Advancing the Design and Implementation of Big Data Warehousing Systems

Carlos Filipe Machado da Silva Costa

UMinho | 2019

Universidade do Minho
Escola de Engenharia

Carlos Filipe Machado da Silva Costa

Advancing the Design and Implementation of Big Data Warehousing Systems

April, 2019

Universidade do Minho
Escola de Engenharia

Carlos Filipe Machado da Silva Costa

Advancing the Design and Implementation of Big Data Warehousing Systems

Doctoral Thesis
Information Systems and Technologies

Work done under the supervision of
Professor Maribel Yasmina Santos (PhD)

April, 2019

"Sometimes life hits you in the head with a brick. Don't lose faith. I'm convinced that the only thing that kept me going was that I loved what I did. You've got to find what you love. And that is as true for your work as it is for your lovers. Your work is going to fill a large part of your life, and the only way to be truly satisfied is to do what you believe is great work. And the only way to do great work is to love what you do. If you haven't found it yet, keep looking. Don't settle. As with all matters of the heart, you'll know when you find it. And, like any great relationship, it just gets better and better as the years roll on. So, keep looking. Don't settle."

(Steve Jobs, 2005)

This page was intentionally left blank

Acknowledgments

To my family...

"When everything goes to hell, the people who stand by you without flinching... they are your family."

(Jim Butcher)

To my friends...

"What greater thing is there for two human souls, than to feel that they are joined for life... to be one with each other in silent unspeakable memories..."

(George Eliot)

To my girlfriend...

"Passion very often makes the wisest men fools, and very often too inspires the greatest fools with wit."

(François de La Rochefoucauld)

To my supervisor...

"If I have seen further it is by standing on the shoulders of Giants."

(Isaac Newton)

This work has been supported by COMPETE: POCI-01-0145-FEDER-007043; FCT - *Fundação para a Ciência e Tecnologia* within the Project Scope: UID/CEC/00319/2013; the SusCity project, MITP-TB/CS/0026/2013; and CCG - *Centro de Computação Gráfica*, providing me the adequate conditions to conclude this doctoral thesis while working as a Big Data Engineer. Moreover, some figures in this document use icons made by Freepik from www.flaticon.com.

This page was intentionally left blank

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

This page was intentionally left blank

Abstract

Current Information Technology advancements have led organizations to pursue high business value and competitive advantages through the collection, storage, processing, and analysis of vast amounts of heterogonous data, generated at ever-growing rates. Since a Data Warehouse (DW) is one of the most remarkable and fundamental enterprise data assets, nowadays, a current research trend is the concept of Big Data Warehouse (BDW), characterizing real-time, scalable, and high-performance systems with flexible storage based on commodity hardware, which can overcome the limitations of traditional DWs to assure mixed and complex Big Data analytics workloads. The state-of-the-art in Big Data Warehousing (BDWing) reflects the young age of the concept, as well as the ambiguity and lack of integrated approaches for designing and implementing these systems. Fulfilling this gap is of major relevance, reason why this work proposes an approach composed of several models and methods for the design and implementation of BDWs, focusing on the logical components, data flows, technological infrastructure, data modeling, and data Collection, Preparation, and Enrichment (CPE). To demonstrate the usefulness, effectiveness, and efficiency of the proposed approach, this work considers four demonstration cases: 1) the application of the proposed data modeling method in several potential real-world applications, including retail, manufacturing, finance, software development, sensor-based systems, and worldwide news and events; 2) the application of the CPE method to process batch and streaming data arriving at the BDW from several source systems; 3) a custom-made extension of the Star Schema Benchmark (SSB), named the SSB+, in which several workloads were developed to benchmark a BDW implemented using the proposed approach, comparing its performance against a traditional dimensional DW; 4) a real-world instantiation based on the development of a BDWing system in the context of smart cities. The results of this research work reveal that the approach can be applied and generalized to support several application contexts, providing adequate and flexible data models that can reduce the implementation time between data collection and data analysis. Moreover, the proposed approach frequently presents faster query execution times and more efficient resource usage than a traditional dimensional modeling approach. Consequently, the proposed approach is able to provide general models and methods that can be used to design and implement BDWs, advancing the state-of-the-art based on a systematic approach rather than an *ad hoc* and use case driven one, which is seen as a valuable contribution to the technical and scientific community related to this research topic.

Keywords - Big Data, Big Data Warehouse, Data Engineering, Data Science.

This page was intentionally left blank

Resumo

Os avanços atuais das Tecnologias da Informação têm levado as organizações a procurar um elevado valor do negócio e vantagens competitivas através da recolha, armazenamento, processamento, e análise de vastas quantidades de dados heterogéneos, gerados a velocidades cada vez maiores. Dado que um DW é um artefacto de dados fundamental nas organizações, uma linha de investigação atual é o conceito de BDW, caracterizando sistemas em tempo-real, escaláveis, de elevado desempenho, com armazenamento flexível, e baseados em *commodity hardware,* sendo capazes de ultrapassar as limitações dos DWs tradicionais de forma a assegurar uma variedade de tarefas complexas de *Big Data analytics*. O estado da arte em BDWing reflete o facto de ser um conceito emergente, bem como a ambiguidade e falta de abordagens integradas para a conceção e implementação destes sistemas. Preencher esta lacuna é significativamente relevante, razão pela qual este trabalho propõe uma abordagem composta por modelos e métodos para conceber e implementar BDWs, focando-se nos componentes lógicos, fluxos de dados, infraestrutura tecnológica, modelação de dados, e na recolha, preparação, e enriquecimento dos dados. Para demonstrar a utilidade, eficácia, e eficiência da solução proposta, este trabalho considera quatro casos de demonstração: 1) a aplicação do método proposto para a modelação de dados em várias potenciais aplicações do mundo-real, incluindo retalho, produção, finanças, desenvolvimento de software, sistemas baseados em sensores, e notícias e eventos a nível mundial; 2) a aplicação do método para recolher, preparar e enriquecer dados (*batch* e *streaming*) provenientes de vários sistemas-fonte; 3) uma extensão do SSB desenvolvida à medida (SSB+), na qual várias *workloads* foram executadas de modo a avaliar o desempenho de um BDW implementado usando a abordagem proposta, comparando-o com um DW dimensional tradicional; 4) uma instância do mundo-real baseada no desenvolvimento de um sistema de BDWing no contexto de *smart cities*. Os resultados deste trabalho revelam que a abordagem pode ser aplicada e generalizada para suportar vários contextos de aplicação, disponibilizando modelos de dados adequados e flexíveis que conseguem reduzir o tempo de implementação entre a recolha de dados e a análise de dados. Além disso, a abordagem apresenta frequentemente tempos mais rápidos na execução de *queries* e um uso de recursos mais eficiente do que uma abordagem dimensional tradicional. Consequentemente, a abordagem proposta pode ser usada para a conceção e implementação de BDWs seguindo uma abordagem sistémica, em vez de uma abordagem *ad hoc* e *use case driven,* o que é visto como um contributo valioso para a comunidade técnico-científica relacionada com este tópico.

Palavras-chave - *Big Data, Big Data Warehouse,* Engenharia de Dados, Ciência de Dados.

This page was intentionally left blank

Table of Contents

This page was intentionally left blank

List of Tables

This page was intentionally left blank

List of Figures

This page was intentionally left blank

List of Abbreviations and Acronyms

In this document, several abbreviations and acronyms are used, which are presented as follows:

3NF	Third Normal Form
ACID	Atomicity, Consistency, Isolation, and Durability
API	Application Programming Interface
BDW	Big Data Warehouse
BDWing	Big Data Warehousing
BI	Business Intelligence
CAP	Consistency, Availability, and Partition tolerance
CPE	Collection, Preparation, and Enrichment
CPU	Central Processing Unit
DBMS	Database Management System
DDL	Data Definition Language
DSRM	Design Science Research Methodology
DW	Data Warehouse
DWing	Data Warehousing
ELT	Extraction, Loading, and Transformation
ETL	Extraction, Transformation, and Loading
GFS	Google File System
GUI	Graphical User Interface
HDFS	Hadoop Distributed File System
IaaS	Infrastructure-as-a-Service
I/O	Input/Output
IoT	Internet of Things
IS	Information Systems
IT	Information Technology

JBOD - Just a Bunch of Disks

JSON - JavaScript Object Notation

MPP - Massively Parallel Processing

NBD-PWG - NIST Big Data Public Working Group

NBDRA - NIST Big Data Reference Architecture

NIST - National Institute of Standards and Technology

NoSQL - Not Only SQL

ORC - Optimized Row Columnar

OLAP - Online Analytical Processing

OLTP - Online Transaction Processing

RAID - Redundant Array of Independent Disks

RAM - Random Access Memory

RDBMS - Relational Database Management System

RDD - Resilient Distributed Dataset

SCD - Slowly Changing Dimension

SF - Scale Factor

SOA - Service-Oriented Architecture

SaaS - Software-as-a-Service

SQL - Structured Query Language

SSB - Star Schema Benchmark

TDWI - The Data Warehousing Institute

TPC - Transaction Processing Performance Council

TPC-DS - TPC Benchmark DS

TPC-E - TPC Benchmark E

TPC-H - TPC Benchmark H

UDF - User-Defined Function

XML - Extensible Markup Language

Chapter 1. Introduction

This chapter introduces the scope and motivation for this doctoral thesis, the research problem, opportunity, and goal, as well as the expected results and the structure of this document. Moreover, the methodology to carry out the research process is also presented in this chapter, including a justification for its adoption, its several activities, and its relationship with the research work and expected results.

1.1 Scope and Motivation

Our world is generating data at unprecedented rates, mainly due to the technological advancements we face, namely in cloud computing, internet, mobile devices, and embedded sensors (Dumbill, 2013; Hashem et al., 2015; Villars, Olofson, & Eastwood, 2011). Collecting, storing, processing, and analyzing all this data becomes increasingly challenging, but organizations who are able to surpass these challenges and extract business value from it, will gain significant competitive advantages. They will be able to better analyze and understand their products, stakeholders, and transactions. Big Data is frequently seen as a buzzword for smarter and more insightful data analyses, but it can be argued that it is more than that, it is about new challenging and more granular data sources, which require the use of advanced analytics to create or improve products, processes, and services, as well as adapting rapidly to business changes (Davenport, Barth, & Bean, 2012).

During the last years, there was an increased interest in Big Data (Google Trends, 2018), and it is sometimes highlighted as fundamental for productivity growth, innovation, and customer relationship, benefiting business areas like healthcare, public sector, retail, manufacturing, and modern cities, for example (Manyika et al., 2011; M. Chen, Mao, & Liu, 2014). The definition of Big Data is ambiguous, and it is difficult to quantify the level at which data becomes big (Ward & Barker, 2013). Therefore, Big Data is frequently defined by its characteristics (e.g., volume, variety, and velocity) and the consequent technological limitations it imposes in organizations, i.e., data that is "too big, too fast, or too hard for existing tools to process" (Madden, 2012). It can be noticed that if Big Data is data that creates technological limitations, then it always existed and it always will. Currently, a paradigm shift is happening in the way we collect, store, process, and analyze data. Organizations need to be aware of these technological trends and strategies that may improve business value. Consequently, Big Data as a research topic is of major relevance to assure that organizations have rigorously justified proofs that emergent techniques and technologies can help them making progress in data-driven business environments.

Moreover, Big Data faces innumerous research challenges mainly divided into four categories: general dilemmas, such as the lack of consensus and rigor in the definition, models, and architectures, for example; challenges related to the Big Data life cycle, from collection to analysis; challenges related to security, privacy, and monitoring; and, finally, organizational change, such as new required skills (e.g., data scientists) or changes in workflows to accommodate the data-driven mindset. Working with Big Data implies knowledge from multiple disciplines and the term data science is frequently highlighted to designate the area responsible for dealing with Big Data throughout the stages of its life cycle, relying on the scientific method (defining hypothesis and validating conclusions) and on knowledge related to areas like machine learning, programming, and databases, for example. Therefore, in this document, data science is referred as the act of extracting patterns and trends from data, using certain data-related techniques, regardless of its characteristics or challenges. These insights can then be communicated or used to create data artifacts or to optimize existing ones, improving business management and performance through data-driven decision-making (C. Costa & Santos, 2017b). In this document, the term data science is used with the meaning afore presented, and terms like data mining, for example, are seen as present in the knowledge of data scientists and, therefore, referred as data science techniques (C. Costa & Santos, 2017b).

As the traditional DW is such a remarkable and fundamental enterprise asset, which leverages data access, analysis, and presentation in appropriate forms to support fact-based decision-making in organizations (Kimball & Ross, 2013), the community starts to question: what is its role in the current era of Big Data? Which considerations for Big Data environments will lead to the redesign of traditional DWs based on relational databases? Which are its main characteristics and how can a BDW be designed and implemented? These questions are of major relevance to understand the role of such recognized data asset in current data-driven environments mainly dominated by volume, variety, velocity, and advanced data analytics, which impose several difficulties to traditional techniques and technologies (Russom, 2014, 2016). Organizations of today's world need to understand if their current DW is limited by the amount, structure, or velocity of data it can process, as well as consider leveraging data science capabilities throughout their daily activities. The BDW is defined by its characteristics, including parallel/distributed storage and processing, real-time capabilities, scalability, elasticity, high performance, flexible storage, commodity hardware, interoperability, and support for mixed and complex analytics. Being a recent concept, related research is emerging, and it becomes critical to study and propose an integrated and validated approach to design and implement both the logical layer (data models, data flows, and interoperability between components) and physical layer (technological infrastructure) of a BDW, a critical gap identified in the literature. The divergence regarding the concept of BDW is alarming, and a prescriptive approach in which models, methods, and instantiations are tightly coupled is needed, providing a cohesive way to build BDWs.

1.2 Research Problem, Opportunity, and Goal

Research related to the concept of BDW is scarce, due to the youth of this topic, and there is no common approach to design and implement it, as the trend mainly consists in finding the best technology to meet Big Data demands (use case driven approach), instead of a data modeling approach (data-driven) (Clegg, 2015). Moreover, there are already some best practices, non-structured guidelines, and implementations in specific contexts, but these do not cover many of the characteristics of a BDW identified in the literature. As works related to the BDW concept are multidisciplinary, certain approaches focus on general guidelines and best practices, while others focus on the technological advancements in storage and analytics, for example. There is no integrated approach focusing on both the logical layer and on the physical layer, in order to implement the characteristics of a BDW with adequate evaluation (e.g., benchmarking, prototypes, and data modeling discussion), thus providing a general-purpose approach, and prescribing models and methods to researchers and practitioners.

In order to identify a relevant research problem, one has conducted a literature review, whose process is detailed in Table 1.1, highlighting the several search engines, keywords, time period, and relevance criteria used throughout this work. The main problem identified in the literature is that there is a significant gap between *"this is what a BDW should be"* and *"this is how it must be designed and implemented",* obviously leading to a use case driven approach primarily concerned with finding the best technology to meet demands. The proposal of a prescriptive approach to design and implement BDWs contributes to the development of new initiatives in a rigorously justified manner, wherein models (representations of logical and infrastructural components), methods (structured practices), and instantiations (prototypes or implemented systems) are tightly coupled and grounded on evaluated practices. Such contribution aims to enrich data-driven approaches in Big Data environments, in which the models and methods are so general that the context of the instantiations becomes as irrelevant as possible, similarly to what usually happens in traditional DWs. Big Data implies severe changes in the way one is used to build traditional DWs, including different techniques and technologies, but this work assumes that it does not need to imply discarding the relevance of data models and methods in favor of a use case driven approach. Consequently, the following research goal is proposed:

> *"Propose a general-purpose approach for designing and implementing BDWs, wherein models and methods are adequately integrated and validated."*

To fulfill the previously mentioned gap, and given the urgent need for extending the knowledge base in BDWing, the proposed approach contains a set of models and methods to guide practitioners working in this area, and also aims to foster future research related to BDWs, by inviting researchers to further evaluate it in several implementation contexts. According to Hevner, March, Park, and Ram (2004), models and methods are framed as Information Technology (IT) artifacts that can be proposed in research processes related to the field of Information Systems (IS), to which this work belongs. Models and methods are seen as relevant artifacts that can be used to extend the current

Table 1.1. Literature review process.

Search Engines	Search Keywords	Evaluation	Time Period
ACM Digital Library Google Scholar IEEE Xplore Scopus Science Direct Web of Science Google (for information about certain technologies)	[“Big Data”], to understand the concept, relevance, challenges, techniques and technologies. Papers or other documents are ordered by relevance, and results are taken into consideration until a series of titles suggest that they do not concern the fundamentals of Big Data enumerated above. This decision is due to the thousands of results retrieved from the search engines. [“Big Data” AND Warehouse(ing)], to review works specifically related to the BDW concept. Papers/documents are ordered by date. As each search engine retrieved only a few hundred articles, all the results are taken into consideration.	1. Titles and abstracts are analyzed. If the work is relevant, then it is saved; 2. Among the saved literature, introductions and conclusions are read. If relevant for discussion, further analysis occurs and the work is cited.	From 01/2010 to 09/2018 Note: Citations of works prior to 2010 occur if it is cited in another work under analysis, or if the work is a reference in the field.

knowledge base with new artifacts. In this research process, the Design Science Research Methodology (DSRM) for IS is used to create these artifacts (Peffers, Tuunanen, Rothenberger, & Chatterjee, 2007), being described in subsection 1.4.

1.3 Research Objectives

As seen in the previous section, the main goal of this doctoral thesis is the proposal of an approach to design and implement BDWs, which should respect the characteristics of a BDW according to the literature, in order to provide prescriptive models and methods for building these complex data assets without ignoring any relevant aspects. Moreover, this doctoral thesis is not focused on “lift and shift” strategies, therefore, the coexistence of the traditional DW with Big Data technologies is not considered here. Consequently, one foresees the use of the proposed approach in the following scenarios: the organization does not currently have a traditional DW and wants to implement a modern data asset, namely a BDW; the organization has a traditional DW and wants to replace it (“rip and replace” strategy); or, finally, the organization relies on a use case driven approach, maintaining a complex and not interoperable federation of different technologies, and wants a data-driven approach with high interoperability between components, well-defined methods, data models, and data flows.

As there is a gap between the understanding of the BDW and the way to implement one, such artifacts are of major relevance to the scientific community and to the practitioners in the area of data engineering and data science, consequently leading to a contribution in which models, methods, and instantiations are tightly coupled and scientifically evaluated. Such contribution provides a structured guide to DW practitioners and promotes future research regarding the concept of BDW, without seeing it as a use case driven approach. Taking this into consideration, the research objectives of this work are defined as follows:

1. Proposal of models and methods:
    a. A model of the logical components and their interoperability, using as general guidelines the National Institute of Standards and technology (NIST) Big Data Reference Architecture (NBDRA) (NBD-PWG, 2015), the Big Data Processing Flow (Krishnan, 2013), the Data Highway Concept (Kimball & Ross, 2013), and the Lambda Architecture (Marz & Warren, 2015). This model also represents how data flows through the different components, rigorously detailing how they interchange data according to the proposed data modeling method;
    b. A method for collecting, preparing, and enriching data flowing to the BDW, including structured, semi-structured, and unstructured data. As previously mentioned, data science techniques (e.g., data mining and text mining) should be taken into consideration, in order to give structure to the data and deliver predictive capabilities. Such concern should be included in the method to propose. This method should also include concerns regarding batch data and streaming data (low latency and high frequency);
    c. A technological infrastructure model, representing how the Big Data technologies can be used, organized, and deployed in a shared-nothing architecture;
    d. A data modeling method that is able to accommodate all types of data regardless of their structure and subject. Obviously, unstructured data does not fit into predefined data

models, therefore, in this case, data mining and text mining techniques are used to extract value from data, giving structure to the relevant findings, and storing those in the BDW. Consequently, the BDW will not only have historical data, but also real-time data and predictive capabilities.

2. Application and evaluation of the models and methods using several demonstration cases:
   a. Evaluate the suitability of the proposed data modeling method when applied to several real-world problems (e.g., retail, manufacturing, finance, sensor-based analysis, and digital media). This objective is focused on making available a set of BDW data models and examples of data modeling guidelines, which practitioners can take into consideration when building their own applications. These examples also complement the smart cities BDW demonstration case presented below, by providing other BDWing contexts;
   b. Design and implementation details regarding batch and streaming data CPE processes. Batch processes do not aim for low latency and high frequency, unlike streaming processes, in which each data point should be loaded into the BDW with a latency between milliseconds and a few seconds. This demonstration case should also consider how several data science techniques (e.g., data mining and text mining) can be efficiently included in batch and streaming data CPE processes, as the approach aims to support the design and implementation of both descriptive and predictive BDWs, as mentioned in the previous objectives;
   c. Benchmark of several workloads and scenarios, including different Scale Factors (SFs) and dimensions size for batch data, use of data partitioning, use of nested attributes, drill across, window and analytics functions, concurrent workloads, and stream processing. This will allow for the evaluation of how a BDW created using the proposed approach handles large scans needed for ad hoc analysis, reporting, and data visualization, compared to a traditional dimensional DW, as well as how it handles streaming scenarios, concurrent workloads, and semi-structured analytics (e.g., analysis using nested arrays, key-value pairs, or geometry objects);

d. Evaluate the suitability of the proposed approach for solving real-world BDWing problems, by implementing a prototype of a BDWing system for smart cities that follows the proposed models and methods. In this case, the SusCity research project will be used, which is focused on the development and integration of new tools and services to improve the efficiency of urban resources, reducing the environmental impact and promoting economic development and reliability (SusCity, 2016). The main goal is to advance the science of urban systems modeling and the data representation supported by the collection and processing of Big Data. This allows the creation of new services that explore economic opportunities and the sustainability of urban systems. The SusCity project has a testbed in Lisbon that includes several data sources (e.g., sensors, census, buildings characteristics, and geolocation data related to mobility), generating data at different velocities (e.g., batch and streaming), with significant volume. Moreover, data science techniques, such as data mining (e.g., clustering and time series forecasting) and data visualization, are crucial to create new services to improve urban systems. Therefore, this research project is used to instantiate the approach, discussing and evaluating the proposed models and methods for collecting, storing, processing, and analyzing Big Data, thus proving the suitability of the approach to solve real-world problems.

It is worth mentioning that throughout this doctoral thesis the invention, adaptation or customization of any Big Data technology is not considered as a contribution. Moreover, this work does not aim to study or focus on data quality mechanisms. Instead, the focus is on the proposal of a cohesive way of building BDWs, a model-oriented and method-oriented approach to assure the adequate interoperability between the different components of a BDW, and evaluate it through demonstration cases that will use state-of-the-art Big Data technologies already developed.

1.4 Research Methodology

Research on IS is mainly characterized by two paradigms: behavioral science, which is focused on the development and verification of theories that explain or predict human or organizational behavior; and design science, which seeks to extend the boundaries of human and organizational capabilities by creating new and innovative IT artifacts, broadly defined as constructs (vocabulary and symbols), models (abstractions and representations), methods (algorithms and practices), and instantiations (implemented systems and prototypes) (Hevner et al., 2004). As the proposed approach is a collection of IT artifacts (models and methods), the use of the DSRM for IS from Peffers, Tuunanen, Rothenberger, and Chatterjee (2007) is suitable to carry out this research process.

The DSRM for IS aims to provide a rigorous way of carrying out design science research and aid the acceptance of this kind of research in the field of IS. Without the existence of a model, the community was facing problems in the evaluation of design science research, and distinguish it from practice activities was difficult. Therefore, the DSRM for IS is seen as a methodology to produce and present design science research, helping researchers to ground their work by referencing a commonly accepted methodology, rather than justifying the research process on an ad hoc basis (Peffers et al., 2007). Figure 1.1 presents the DSRM for IS used in this work, in which the entry point of the research process is considered to be problem-centered, after reviewing the literature and identifying the lack of a prescriptive approach to design and implement BDWs. The research process is as follows:

1. Problem identification and motivation - this activity focuses on defining the research problem and justifying the value of the solution, both already highlighted in this chapter. The problem definition is used to motivate the proposal of an artifact that effectively provides a solution.

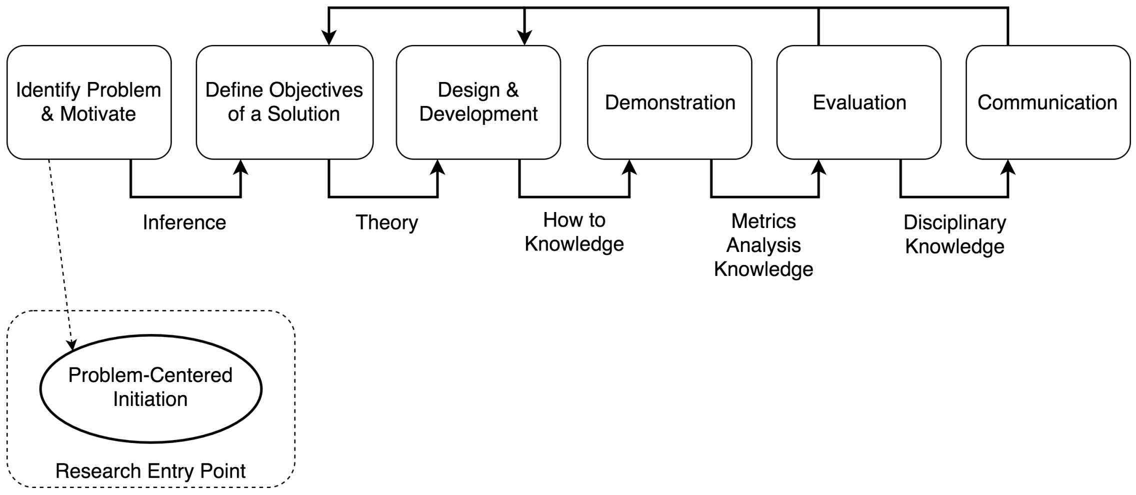


Figure 1.1. Research methodology (DSRM for IS). Adapted from (Peffers et al., 2007).

At this point, the lack of a prescriptive approach to design and implement BDWs covering logical and physical aspects is the identified research problem, according to the literature, and the main motivation, as already highlighted, is to address this gap and help researchers and practitioners in the area. The inputs used in this activity are the state-of-the-art in Big Data and BDW, as well as the relevance to provide a solution for the identified problem;

2. Define the objectives for a solution - this should be inferred from the problem definition. In this doctoral thesis, these objectives are presented in section 1.3, encompassed by the main goal of proposing the BDWing approach. Part of the research objectives are mainly centered around efficiency, as several benchmarking metrics are used to assess the proposed guidelines. In this context, a traditional dimensional DW is frequently used as baseline, and thresholds like executing ad hoc queries in large datasets within a few seconds or tens of seconds is expected, according to the current state-of-the-art regarding storage technologies and analytical mechanisms for BDW implementation. Regarding the SusCity prototype, the evaluation is mainly based on effectiveness, i.e., the artifact is applied to solve the problem, and it either solves it completely, partially or does not solve it in any form, generating a discussion of the results. Finally, there are other research objectives presented in section 1.3 aiming to clarify some of the guidelines provided to practitioners, namely the data CPE and data modeling guidelines. Once there are no similar approaches to design and build BDWs, the proposal cannot be directly compared to any related work;
3. Design and development - in this activity, the artifacts are created, including the models and methods of the BDWing approach. They should comply with the characteristics of a BDW identified in the literature, in order to provide the adequate functionalities. The inputs used in this activity are the state-of-the-art in Big Data and BDW, mostly the knowledge related to techniques and technologies suitable for BDWs, as well as some best practices and non-structured guidelines already present in the literature;
4. Demonstration - the approach is instantiated to demonstrate its usefulness and effectiveness for solving a real problem in the context of smart cities, namely in the SusCity research project. In addition to that, as mentioned, a custom-made extension of the SSB benchmark

(SSB+) is used to provide another demonstration case and to consolidate the evaluation of the approach. Furthermore, two other demonstration cases related to data CPE and data modeling are also developed to consolidate the work regarding some design and development choices. Consequently, in this doctoral thesis, four demonstration cases are developed, in order to apply the proposed artifacts in different contexts. Throughout this activity, it is crucial the knowledge on how to use the artifact to solve the problem;

5. Evaluation - it will be observed and measured how well the approach supports the solution of the problem, when compared to the research objectives. The evaluation of the demonstration cases consists in assessing the proposed approach, mainly in terms of effectiveness, complexity and latency, throughout different phases of the Big Data life cycle (e.g., collection/loading, cleansing, integration, transformation, and analysis). Considerations regarding storage, Random Access Memory (RAM), and Central Processing Unit (CPU) requirements are also relevant for discussion whenever applicable. Finally, the proposed approach should respect the characteristics of a BDW identified in the literature, in order to be evaluated as a satisfactory prescriptive approach. The following points detail the evaluation activity:
    a. The proposed approach should be general-purpose, being suitable for BDWing contexts and focusing on both the physical and logical layers. For its evaluation, an extension of the SSB benchmark (SSB+) is developed and executed (streaming data scenarios, drill across, window and analytics functions, dimensions size evaluation, nested attributes, concurrent workloads, and data partitioning), in order to observe several phenomena under controlled and general-purpose contexts, providing adequate measures mainly regarding latency, CPU usage, and memory and storage constraints. For queries based on large scans of batch and streaming data, the optimal interactivity threshold is within a few seconds (maximum of 10 seconds), based on literature related to the users' tolerance regarding a computer's response time (Nah, 2004; Nielsen, 1993), while the satisfactory interactivity threshold is a few tens of seconds (e.g., 20 or 30 seconds), depending on the business

requirements, data volume, complexity of the queries, number of users, frequency of inserts/updates, and configuration of the infrastructure;

b. In addition to that, the proposed approach is also evaluated using a real-world instantiation that will show how the same can be applied to produce working systems in real-world contexts, in this case, the SusCity research project (SusCity, 2016). This evaluation will focus on the effectiveness of a BDW to support a Web-based interactive data visualization platform providing intensive geospatial analytics and simulations (e.g., buildings retrofitting measures and energy grid performance under different scenarios). In the context of the SusCity project, interactivity requires less than 10 seconds of response time, for the same reasons presented above (Nah, 2004; Nielsen, 1993);

c. Finally, this work also includes two additional evaluation contexts: the first one focuses on the development of adequate data CPE processes following the proposed approach; while the second one focuses on the data modeling aspect of BDWs according to the method proposed in this work, presenting several BDW data models that are suitable for real-world applications. In this case, evaluation will mainly focus on effectiveness and complexity, i.e., assessing if the models and methods are able to support several data CPE workloads and real-world BDW applications, while avoiding some complexity typically found in traditional dimensional DWs (e.g., several types of dimensions, bridge tables, SCDs, and late arriving dimensions).

6. Communication - it mainly involves writing and publishing this doctoral thesis, scientific publications in conferences and journals, books, or other communications to practitioners, if applicable. This activity aims to make widely available the problem and its relevance, as well as the artifact and its usefulness, novelty, rigor, effectiveness, and efficiency.

1.5 Document Structure

After this chapter with the scope, motivation, research goal, objectives, and methodology, this document is structured as follows: Chapter 2 presents the relevance, definition, and challenges in Big

Data contexts, and the techniques and technologies to design and implement Big Data solutions, including relevant Big Data architectures; Chapter 3 describes works related to BDWing, including requirements identification, design changes, guidelines, advancements and implementations in specific contexts; Chapter 4 presents the proposed approach, providing the models and methods for designing and implementing effective and efficient BDWs, which allows for the understanding of the approach in its general form, i.e., without being instantiated in a specific context; Chapter 5 provides several BDW data models and data modeling considerations that practitioners can follow to implement BDW applications, being this chapter particularly useful for clarifying and applying the general data modeling method, and to facilitate the transition from theory to practice; Chapter 6 presents several workloads for collecting, preparing and enriching the data according to the proposed approach, facilitating the understanding of the adequate mechanisms to deal with the data throughout these three stages; Chapter 7 evaluates the performance of BDWs, by benchmarking several design decisions related to the approach proposed in this work, which serves to provide support for its models and methods; Chapter 8 presents a real-world BDW application in the context of smart cities, namely discussing the implementation of the SusCity project from data collection to data visualization, finalizing the discussion of the demonstration cases developed in this work; Chapter 9 concludes with some remarks about the undertaken work and some prospects for future work.

This page was intentionally left blank

Chapter 2. Big Data

The way people interact with organizations and the rate at which the transactions occur create unprecedented challenges in data collection, storage, processing, and analysis. If organizations find a way to extract business value from this data, they will most likely gain significant competitive advantages (Villars et al., 2011). Big Data is often seen as a buzzword for smarter and more insightful data analysis, but it is more than that, it is about new challenging data sources helping to understand business at a more granular level, creating new products or services, and responding to business changes as they occur (Davenport et al., 2012).

We live in a world constantly producing and consuming data, being a priority to understand the value that can be extracted and analyzed from it. Organizations need to understand and analyze relevant data flows, join data analytics with product/process development, and move it closer to the core business (Davenport et al., 2012). This chapter presents the relevance of Big Data in today's world, several attempts to define it, the related challenges, and several techniques and technologies to efficiently design and implement Big Data solutions (C. Costa & Santos, 2017a).

2.1 Big Data Relevance

Over the last years, the interest in Big Data has increased considerably (Google Trends, 2018), particularly after 2012, as can be seen in Figure 2.1. In a McKinsey Global Institute's report, Manyika et al. (2011) argue that Big Data will become fundamental for productivity growth, innovation, and customer relationship among organizations, highlighting its relevance in healthcare, public sector, retail, manufacturing, and personal-location contexts, stating that value can be generated in each one of them. Nowadays, data has a strong presence in the daily activities of almost every industry, alongside labor and capital, as Manyika et al. (2011) demonstrated by estimating that, in 2009, almost all economic sectors in the United States had, at least, nearly 200TB of stored data per

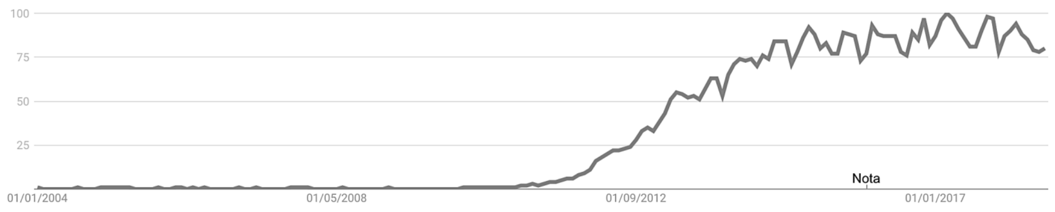


Figure 2.1. Increased interest in Big Data. Reprinted from (Google Trends, 2018).

organization with more than 1,000 employees. Other statistics show that the amount of data available in today's world is growing exponentially (Chandarana & Vijayalakshmi, 2014).

Nevertheless, as human beings tend to resist to changes, there are still the ones who ask themselves: "Why Big Data? Why Now?" (Krishnan, 2013). According to Krishnan (2013), the concept of Big Data is about leveraging access to a vast volume of data, which can help retrieving value for organizations, with minimal human intervention, due to the advancements made in data processing technologies. The author claims that Big Data always existed in several industries, but the appearance of autonomous, fast, flexible, and scalable processes created a new paradigm shift, often resulting in a cost reduction when compared to traditional data processing approaches.

Organizations find themselves facing this new data-driven way to conduct business, and a paradigm shift in their infrastructure and way of thinking is, understandably, a step to consider seriously. However, they need to foresee the value that Big Data can bring to their business (Manyika et al., 2011):

- The use of Big Data can make information more transparent and usable across the organization;
- Business performance can be increased with more accurate and detailed facts, which is possible by collecting and processing more transactional data;
- Better management decisions can be made through data analysis;
- The use of Big Data has the ability to refine and reinvent products and services.

Even so, the evidence that using Big Data intelligently will improve business performance can still be questioned, as McAfee, Brynjolfsson, Davenport, Patil, and Barton (2012) highlight by discussing the inadequacy of business press to demonstrate the real value of being data-driven and testing the hypothesis that data-driven organizations are better performers than traditional ones. The authors interviewed executives in 330 organizations and also gathered performance data about their respective organizations. McAfee et al. (2012) come to an interesting conclusion: organizations that view themselves as data-driven achieved better performance regarding financial and operational goals. The authors highlight more productivity and profitability for top organizations that used data-driven decision-making, even taking into consideration other factors like labor and capital, for example. The results achieved by McAfee et al. (2012) rigorously corroborate the current trend for Big Data value within organizations. The use of Big Data will become inevitable for competitive advantages across most of the industries, from electronic and information industries to finance, insurance, or government. Big Data can leverage increasing productivity and better customer relationship (Manyika et al., 2011), and can potentially be used in several business areas to generate significant value for organizations (Chandarana & Vijayalakshmi, 2014; Manyika et al., 2011; Villars et al., 2011), as Table 2.1 demonstrates.

According to Brown, Chui, and Manyika (2011), other business areas are worth mentioning, such as finance, insurance, and real estate. The authors present an approach that analyzes several business areas by the ease-of-capture Big Data and its potential to generate value. The apparent trend is for organizations to perceive value in data-driven decision-making and start collecting more data, contributing to the continuous growth in data volume. Big Data will have a significant impact in value creation and competitive advantage for organizations, such as new ways to interact with customers or to develop products, services, and strategies, consequently raising profitability. Another area where the concept of Big Data is of major relevance is the Internet of Things (IoT), seen as a network of sensors embedded into several devices (e.g., appliances, smartphones, cars), which is a significant source of Big Data, bringing many business environments (e.g., cities) into the era of Big Data (M. Chen et al., 2014).

Table 2.1. Big Data applied in several business areas.

Business Area	Examples of Application
Healthcare	▪ Personalize medication and understand causes of diseases, using techniques to extract value from vast amounts of data about medical history, medication, and drug manufacturing, for example; ▪ Other Big Data sources can include exercise data or even more unstructured data like medical images.
Environment	▪ Find a correlation between the measured values and the implications for the environment through the collection of data from multiple sensors (e.g., air and water quality, metrology, and gas emissions).
Public sector	▪ Use Big Data to prevent fraud and errors regarding taxes; ▪ Customize actions by segmenting population; ▪ Create more transparency through data availability.
Retail	▪ Event forecasting and customer segmentation, creating personalized products or services; ▪ Location based marketing, sentiment analysis, and cross-selling; ▪ Logistics optimization.
Manufacturing	▪ Demand forecasting for supply planning; ▪ Use of sensors in manufactured products to offer proactive maintenance.
Life sciences	▪ Analyze genetic variations and the effectiveness of potential treatments, using vast amounts of data.

As presented above, Big Data brings competitive advantages to organizations, but there are particular characteristics that define it, although most of the time they are unquantifiable (Ward & Barker, 2013), as will be discussed in the next section. Big Data creates a new paradigm shift in the way we collect, store, process, and analyze data, but organizations can be data-driven and explore the potential of data from innumerous sources without dealing with Big Data techniques and technologies. In this case they are just dealing with new data, data that was not previously processed within the organization, but does not impose severe difficulties in the capabilities of traditional techniques and technologies. In the next section, the definition of Big Data will be discussed according to several perspectives from different authors.

2.2 Big Data Characteristics

At this point, the notoriety and relevance of Big Data is understandable, potentially changing the way organizations operate and create new opportunities based on data-driven approaches. Technological advancements open the way for unprecedented amounts of data generated each day, at ever-increasing rates. In 2011, around 1.8 zettabytes of data were produced in a couple of days, more than it was produced from the beginning of civilization until 2003 (M. Chen et al., 2014). Storage capacity must increase, and new ways of dealing with such amounts of data emerge, but what actually means Big Data?

First of all, there is no widely accepted threshold for classifying data as Big Data. Ward and Barker (2013), in an attempt to clearly define Big Data, present several notorious definitions among the community, highlighting that Big Data is predominantly and "anecdotally" associated with data storage and data analysis, terms dating back to distant times, and also argue that the adjective "big" implies significance, complexity, and challenge, but also makes it difficult to quantitatively define Big Data. Ward and Barker (2013) present several definitions, some defining Big Data by its characteristics, others based on the augmentation of traditional data with more unstructured data sources, and some trying to quantify it. They also present definitions which rely on the inadequacy of traditional technologies to deal with this new type of data, presenting several perspectives from the industry, including Gartner, Oracle, Intel, Microsoft, and IBM, for example. In order to conclude about the similarity among the definitions, Ward and Barker (2013) state that all definitions include at least one of the following aspects: size, complexity, or techniques and technologies to process large and complex datasets.

Dumbill (2013) attempts to provide a definition: *"Big Data is data that exceeds the processing capacity of conventional database systems. The data is too big, moves too fast, or does not fit the strictures of your database architectures. To gain value from this data, you must choose an alternative way to process it".* M. Chen et al. (2014) corroborate this definition by focusing on the fact that traditional software and hardware cannot recognize, collect, manage, or process this new type of data in reasonable time. Krishnan (2013) also agrees with these perspectives, defining Big Data by its complexity, creation speed, and several degrees of ambiguity, whose processing is inadequate for traditional methods, algorithms, and technologies. Although Ward and Barker (2013) are slightly critical both regarding the lack of quantification in Big Data's definition and the use of data storage and analysis in several attempts to define it, in reality, they conclude by stating that the concept of Big Data includes storage and analysis of large and complex datasets, using a set of novel techniques.

The origin of the concept is relatively unknown, and its definition evolved rapidly, thus raising uncertainty. Gandomi and Haider (2015) state that size is the characteristic that first stands out, but others became usual to define Big Data. In 2001, Doug Laney, from Gartner, presented the 3Vs model (Figure 2.2) to characterize Big Data by its volume, variety, and velocity (Laney, 2001). IBM and Microsoft based their definitions of Big Data on this model for at least another 10 years (M. Chen et al., 2014).

According to Gandomi and Haider (2015), volume is a characteristic which indicates the magnitude of data, mentioning that it is frequently reported between Terabytes and Petabytes, citing the survey of Schroeck, Shockley, Janet, Romero-Morales, and Tufano (2012), wherein just over half of the respondents consider datasets bigger than 1TB to be Big Data. However, the authors discuss that data size is relative and varies according to the periodicity and the type of data. It is impractical to define a specific threshold for Big Data volume, as different types of data require different technologies to deal with it (e.g., tabular data and video data), as Gandomi and Haider (2015) exemplify. The volume in the 3Vs model characterizes the amount of data that is continuously generated (Krishnan,

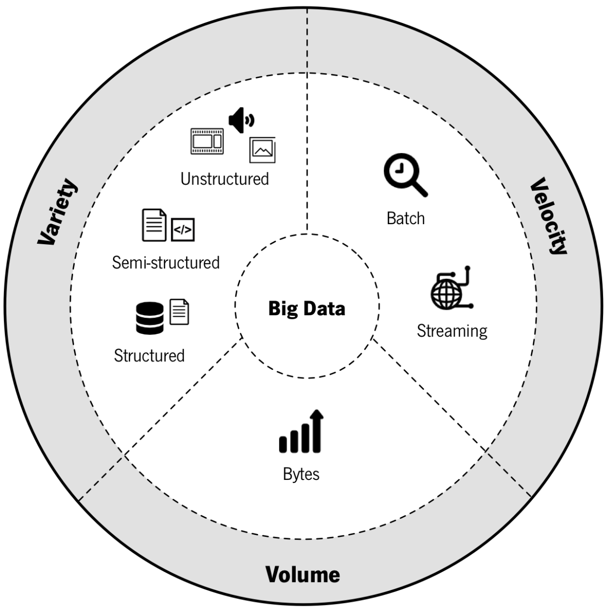


Figure 2.2. The 3Vs model. Adapted from (Zikopoulos & Eaton, 2011).

2013), and the main cause for the ever-increasing volume is the fact that we currently store all our interactions with the majority of services available in our world (Zikopoulos & Eaton, 2011).

Regarding variety, Big Data can be classified as structured (e.g., transactional data, spreadsheets, and relational databases), semi-structured (e.g., Web server logs, Extensible Markup Language – XML, and JavaScript Object Notation - JSON), and unstructured (e.g., social media posts, audio, video, and images) (Chandarana & Vijayalakshmi, 2014; Gandomi & Haider, 2015). Traditional technologies can present significant difficulties to store and process Big Data, such as content from Web pages, click-stream data, search indexes, social media posts, emails, documents, and sensor data, for example. Most of this data does not fit well in traditional databases and there must be a paradigm shift in the way organizations perform analyses to accommodate raw structured, semi-structured, and unstructured data, in order to take advantage of the value in Big Data (Zikopoulos & Eaton, 2011).

The final characteristic in the 3Vs model is velocity, referring either to the rate at which data is generated or to the speed of analysis and decision support (Gandomi & Haider, 2015). Data can be generated at different rates, ranging from batch to real-time (streaming) (Chandarana & Vijayalakshmi, 2014; Zikopoulos & Eaton, 2011). It is relevant to apply the definition of velocity to data in motion, instead of the rate at which data is collected, stored, and retrieved from storage. Continuous data streams can create competitive advantages in contexts where the identification of trends must occur in short periods of time, as in financial markets, for example (Zikopoulos & Eaton, 2011).

Over time, two additional characteristics emerged: value and veracity. Value represents the expected result of processing and analyzing Big Data (Chandarana & Vijayalakshmi, 2014), which usually has low value in its raw state, as this is mainly extracted with an adequate analysis (Gandomi & Haider, 2015). According to Chandarana & Vijayalakshmi (2014), value can be obtained through the integration of different data types to improve business and gain competitive advantages. On the other hand, veracity draws attention to possible imprecise data, since, sometimes, the analysis is based on datasets with several degrees of precision, authenticity, and trustworthiness (Chandarana & Vijayalakshmi, 2014). Gandomi and Haider (2015) corroborate this definition, highlighting the unreliability of certain data sources (e.g., customer sentiments extracted from social media), although recognizing that they can be valuable when adequate techniques and technologies are used.

Other characteristics, not so noticeable according to the literature, are the variability and complexity, introduced by SAS (Gandomi & Haider, 2015). Variability is related to the different rates at which data flows, according to different peaks and inconsistent data velocity. Complexity highlights the challenge of dealing with multiple data sources, namely to connect, match, clean, and transform them. Besides these, Krishnan (2013) also proposes three other characteristics: ambiguity, related to the lack of appropriate metadata, resulting from the combination of volume and variety; viscosity, when the volume and velocity of data causes resistance in data flows; virality, which measures the time of data propagation among peers in a network. Figure 2.3 presents a summary of all these characteristics identified in the literature.

At this point, it seems that trying to quantify any of these characteristics becomes an impossible task. Big Data remains as an abstract concept (M. Chen et al., 2014). It must be accepted that it can be a combination of several characteristics, or a strong presence of only one, but it must be recognized

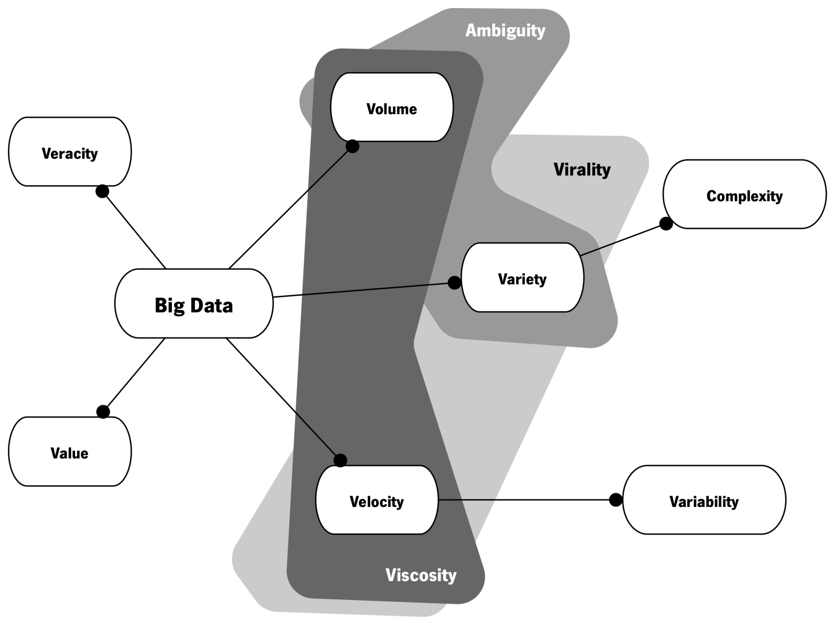


Figure 2.3. Main Big Data characteristics identified in the literature. Adapted from (C. Costa & Santos, 2017a).

as data that makes changes in the way we think about techniques and technologies, if they are inadequate to deal with it. It may be a database or a DW that cannot scale accordingly on a shared-everything architecture (Krishnan, 2013), or data mining tasks that cannot be finished without parallel computing. Again, data is *"too big, too fast, or too hard for existing tools to process"* (Madden, 2012). Defining Big Data by the inadequacy of traditional technologies is relatively dangerous, as advancements are constantly being made (e.g. quantum computers), and such definition implies that Big Data always existed and will continue to exist (Ward & Barker, 2013). The current definitions of Big Data are relatively dependent on the techniques and technologies to collect, store, process, and analyze it. These will evolve over time, and we need to learn to live with it. It will always be a matter of analyzing new technological trends that may benefit business and reconsider new strategies related to data. Currently, a new paradigm shift is happening. It does not need to be a change in all organizations, but scientific progress related to Big Data will continue to exist, in order to assure that organizations have rigorously justified proofs that emerging techniques and technologies can help them making progresses in data-driven environments. The state-of-the-art regarding Big Data techniques and technologies will be presented later in this document. Next section presents the challenges regarding Big Data and the adoption of such initiatives.

2.3 Big Data Challenges

This section presents several challenges regarding Big Data, including general dilemmas, challenges in the Big Data life cycle, issues in security, privacy, and monitoring, as well as required changes in organizations. These challenges also serve to identify relevant research topics across several fields.

2.3.1 Big Data General Dilemmas

General dilemmas may include challenges such as the lack of consensus and rigor in Big Data's definition, models, and architectures, for example. M. Chen et al. (2014) claim that the concept of Big Data is often more commercial speculation than it is a scientific research topic. The authors also mention the lack of standardization in Big Data, such as data quality evaluation and benchmarking. In fact, the lack of standard benchmarks to compare different technologies is seriously aggravated by the constant technological evolution in Big Data environments (Baru, Bhandarkar, Nambiar, Poess, & Rabl, 2013).

Even the ways to fully use Big Data remain an open subject to explore, such as applications in science, engineering, medicine, finance, education, government, retail, transportation, or telecommunications, for example (M. Chen et al., 2014). Discussions such as how to select the most appropriate data within several sources or how to estimate their value remain as Big Data dilemmas (Chandarana & Vijayalakshmi, 2014). Another commonly discussed pitfall is how Big Data helps representing the population better than a small dataset (Fisher, DeLine, Czerwinski, & Drucker, 2012). This obviously varies with the context, but the authors call our attention for not assuming that more data is always better.

2.3.2 Challenges in the Big Data Life Cycle

These challenges are related to technical difficulties in tasks such as Big Data collection, integration, cleansing, transformation, storage, processing, analysis, and governance:

- The need to rethink storage devices, architectures, mechanisms, and networks, in order to achieve more efficient input/output (I/O), data accessibility, and data transmission (C. L. P. Chen & Zhang, 2014);
- Scalability becomes crucial to store and analyze data. Handling increasing amounts of data requires redesigning databases and algorithms to extract value from it (Hashem et al., 2015). Distributed/Parallel computing becomes crucial to deal with Big Data, assuring availability, cost efficiency, and elasticity (M. Chen et al., 2014);
- Assuring data quality and adding value through data preparation becomes challenging in Big Data environments (C. L. P. Chen & Zhang, 2014). Different data sources may have different data quality problems (Hashem et al., 2015). These problems and vast amounts of redundancy can also make data integration more difficult (M. Chen et al., 2014). The heterogeneity resulting from multiple sources augment these challenges, as traditional techniques for data analysis expect homogeneous data (Jagadish et al., 2014). Heterogeneity

brings implications on data integration (Cuzzocrea, Song, & Davis, 2011) and consequences in the analysis of Big Data, as the unstructured nature of data sources presents several challenges regarding transformations to support adequate analytical tasks;

- Visualizing Big Data requires rethinking traditional approaches due to the volume of data, thus combining appearance and functionality is crucial (C. L. P. Chen & Zhang, 2014). Advanced data visualizations are needed to extract value from Big Data (Russom, 2011), having the capability to scale to thousands or millions of data points, handle multiple data types, and be easy to use, in order to satisfy several users. Krishnan (2013) argues that manipulating Big Data is challenging due to its characteristics, namely executing drilldowns or rollups. In these visualizations, data from multiple sources is typically integrated into a single picture. The author indicates that technological evolutions would be made to address the challenge of manipulating Big Data interactively, as discussed later in this document;
- Searching, mining, and analyzing Big Data is a challenging and relevant research trend, including Big Data searching algorithms, recommendation systems, real-time Big Data mining, image mining, text mining, among others (M. Chen et al., 2014). As Gandomi and Haider (2015) claim, size is frequently the main concern in Big Data, but the unstructured nature of data also deserves attention (e.g., text, audio, and video) and imposes significant challenges in these tasks;
- Big Data governance faces challenges regarding control and authority over massive amounts of data from different sources (Hashem et al., 2015). Managing such heterogeneous environment to plan access policies and assure traceability can quickly become almost impossible without adequate governance tools.

Organizations face several challenges in the Big Data life cycle. New business problems require technological innovations in the way data flows across the organization. Surpassing these challenges will depend on the organization’s maturity, since legacy applications and the use of incompatible formats can impose several difficulties to an adequate integration and extraction of value from Big Data. Collecting data, namely gaining access to it, may also be a challenge, as integrating data from multiple sources, including external ones, raises questions about others' intention to share it free of charge (Manyika et al., 2011).

M. Chen et al. (2014) state that the efficiency in data flows is a key factor to assure an adequate Big Data processing. The authors also highlight the challenge of building effective real-time computing models and online applications to analyze Big Data. Other challenges related to processing Big Data may include the reutilization and reorganization of data, which become laborious at scale. The characteristics of Big Data require a paradigm shift in databases and analytical technologies, as dealing with Big Data throughout its life cycle can potentially create severe bottlenecks in networks, storage devices, and relational databases. Technology is evolving to execute these stages in distributed environments, becoming dependent on high storage capacity and processing power.

Even relational databases are evolving to accommodate these trends, increasing query performance and being able to deal with more data variety (Davenport et al., 2012). Combining the benefits of a Relational Database Management System (RDBMS) and the database systems proposed from the need to handle Big Data actually represents a research trend, as well as query optimization in Big Data technologies (Cuzzocrea et al., 2011). Furthermore, advancements are constantly being made in scalable storage and algorithms. Ji, Li, Qiu, Awada, and Li (2012) argue that processing queries in Big Data may take significant time, as it is challenging to sequentially iterate through the whole dataset in a short amount of time. Consequently, the authors highlight the relevance of designing indexes and considering adequate preprocessing technologies. Hashem et al. (2015) identify the need to study adequate models to store and retrieve data, a crucial factor to successfully implement Big Data solutions. Models and algorithms for scalable data analysis also remain an open research issue, as well as the integration and analysis of data arriving continuously from streams. Mining data streams has been identified as an emergent research topic in Big Data analytics (H. Chen, Chiang, & Storey, 2012).

2.3.3 Big Data in Secure, Private, and Monitored Environments

Nowadays, keeping data secure and private is one of the most concerning tasks for organizations (M. Chen et al., 2014; Jagadish et al., 2014). Users want to rest assure that any leaks into the public domain will not occur (Chandarana & Vijayalakshmi, 2014). Sagiroglu and Sinanc (2013), citing a survey from Intel IT Center (2012), claim that security and privacy are frequently mentioned among the Big Data concerns of IT managers. It is relevant to plan a Big Data driven security model for organizations to accurately specify risks and prevent illegal activity or cyber threats. Several considerations are mentioned, such as authentication, authorization, network traffic analysis, data protection laws, and mining data related to security. M. Chen et al. (2014) also discuss the potential for Big Data applications related to security concerns.

Due to the characteristics of Big Data, more risks arise, and traditional data protection methods must be rethought. M. Chen et al. (2014) argue that Big Data applications face multiple challenges related to security, privacy, and monitoring: protection of personal privacy during not only data collection, but also in its subsequent storage and flows; Big Data quality and its influence on the appropriate and secure use of data; the performance of security mechanisms like encryption is largely influenced by the scale and variety of data; and other aspects related to secure communications, administration, and monitoring in environments with multiple users and services. Other relevant challenge, as highlighted by Hashem et al. (2015), is assuring Big Data integrity, i.e., data is only modified by the owner or other authorized entities.

Policies related to data are also relevant today, at a time when there is a significant amount of sensitive data about individuals, such as the one related to their health or finances (Manyika et al., 2011). Legal issues are being raised regarding the easiness to copy, integrate, and recurrently use data by different people. Intellectual property, data ownership, and responsibility regarding inaccurate data deserve proper attention from policy makers (Manyika et al., 2011). Legal and regulatory issues also deserve attention in several aspects (Ji et al., 2012), like analyzing the adequacy of current laws and regulations to adequately protect data about individuals (Hashem et al., 2015). Even the constant tracking on employees within an organization can raise discussions regarding adequate work policies (Michael & Miller, 2013).

Besides these issues, Brown et al. (2011) raise questions about the implications of having data widely and transparently available. Organizations that rely on costly proprietary data to leverage their competitive advantages will face challenges due to the promises of more accessible Big Data sources, as they become widely available in some contexts. The authors also discuss the inherent difficulties for organizations to share data between departments, forming a coherent view of the organization, which is additionally aggravated with Big Data. Organizations need to integrate data from multiple sources and promote collaboration, not only among departments, but also among suppliers and customers (Brown et al., 2011).

Assuring privacy is both a technical and sociological problem (Jagadish et al., 2014). The inadequate availability of location-based data allows the possibility to infer a person's residence, office location, and identity, for example. Moreover, many other data sources can contain personal identifiers, or even if no personal identifiers exist, when data is rich enough, reasonable inferences can be drawn from it (Wigan & Clarke, 2013). Currently, we tend to share more data online, most of the time without knowing the implications. Another relevant topic, briefly mentioned above, is data ownership, due to its value for certain organizations that are currently debating about ways of sharing or selling data without losing control of it (Jagadish et al., 2014). Data ownership is often discussed regarding social media Websites, since the users' data is not owned by the organizations although they store it (Chandarana & Vijayalakshmi, 2014). As Wigan & Clarke (2013) discuss, these organizations tend to assume that they hold the rights of the data, and sometimes the current legislation benefits them, allowing organizations to not permanently delete data, even when users ask for it.

Discussing Big Data security, privacy, and monitoring in cloud environments is also relevant. Organizations frequently recognize that using Big Data technologies in cloud environments helps reducing their IT costs (Ji et al., 2012), although raising concerns about Big Data storage and processing infrastructures. Therefore, one of the challenges lies in assuring adequate monitoring and security without exposing users' data when processing it (Ji et al., 2012).

2.3.4 Organizational Change

Surely Big Data may sound appealing to most organizations, but, frequently, organizational leaders lack the understanding regarding its value and the means to extract it (Manyika et al., 2011). Occasionally, the lack of knowledge on how to use analytics is mentioned as the leading obstacle to become more data-driven (LaValle, Lesser, Shockley, Hopkins, & Kruschwitz, 2011). Within several business areas, organizations need to monitor trends and gain advantages compared to their competitors, but many of them lack the talent, the rigorous workflows structure, and the incentives for adequate Big Data initiatives to better support decision-making (Manyika et al., 2011). Leaders and policy makers must understand how Big Data can create value, as well as critically think about IT capabilities, data strategies, analytical talent, and data-driven approaches. This paradigm shift in organizations requires them to move analytics into the core business and operational functions (Davenport et al., 2012), changing business processes, delivering insights related to customers, products, services, and other transactions. McAfee et al. (2012) present five challenges that organizations will face in management, caused by Big Data initiatives:

- Assure adequate leadership for a Big Data project;
- Find suitable data scientists (Provost & Fawcett, 2013), computer scientists, and other professionals to deal with Big Data, design experiments, and overcome business challenges;
- Understand and adequately use Big Data technology;
- Combine problem-solving people with the right data for decision-making;
- Change organizational culture and rethink how data-driven the organization really is.

Big Data initiatives require a multidisciplinary approach, demanding collaboration to deliver useful results that must be properly understandable by the organization (Jagadish et al., 2014), but to accomplish this, challenging organizational changes must occur.

2.4 Techniques and Technologies for Big Data Solutions

As previously mentioned, the concept of Big Data is often used to sell something (Fan & Bifet, 2013), denoting a lack of common understanding, and opening the way for an almost infinite set of technologies. Unfortunately, this raises significant challenges to understand, adopt, or design techniques to work with Big Data, since they are tightly coupled with a specific technology. The opposite problem also occurs, since most of the time in conceptual models, it is not clear which technology takes place in a certain component of the model. This is mainly due to Big Data's variety, but even discarding unstructured data (e.g., text, video, image, and audio), it seems that almost everyone is trying to sell their solutions to do something with Big Data, without concerns regarding a common way to design and implement solutions.

The era of Big Data can generate significant controversy. For example, Fan and Bifet (2013), citing Boyd and Crawford (2012), claim that it is not necessary to distinguish Big Data analytics from data analytics, as data volume will continue to grow and never decrease. The transition from traditional techniques and technologies represent a radical paradigm shift, which can include abandoning shared-everything architectures (Krishnan, 2013), RDBMSs, common Extraction, Transformation, and Loading (ETL) mechanisms, or the Structured Query Language (SQL), for example. The ambiguity in Big Data's definition, the lack of formal and recognized techniques, and the vast set of available technologies do not help in a peaceful acceptance. It can be argued that the Big Data analytics area needs approaches like the widely accepted work from Kimball and Ross (2013) that focuses on how to store and analyze data in a relational DW. Assuring that businesses do not refrain from progress due to uncertainty or lack of resources is of major relevance.

In order to understand when it is appropriate to rethink traditional techniques and technologies, according to a survey from Russom (2011), organizations tend to replace traditional platforms when:

- Massive performance and scalability are required, such as the need to scale to Big Data contexts with a large volume of data, speed up data collection and queries, or assure concurrent workloads;

- Business users need advanced analytics (e.g., data mining, statistical analysis, text analytics, and ad hoc SQL queries), and the current platform is Online Analytical Processing (OLAP) only;
- Organizations need self-service and rich visualization tools for end users;
- The platform lacks modern capabilities, such as support for a Service-Oriented Architecture (SOA), cloud infrastructures, or in-memory processing.

This section aims to present several techniques to understand and deal with Big Data throughout its life cycle, from collection to analysis, including storage and mining. These techniques mainly represent a collection of guidelines that helps designing Big Data solutions, namely their several components, the relationship between them, and some necessary changes in traditional approaches for dealing with data. Furthermore, in this section, several Big Data technologies are presented, as well as a recent standardization proposal for Big Data architectures, published by the National Institute of Standards and Technology (NIST).

2.4.1 Designing Big Data Solutions

This subsection presents techniques identified in the literature that are adequate to support the design of Big Data solutions. According to C. L. P. Chen and Zhang (2014), citing Marz and Warren (2015) and Garber (2012), a Big Data solution generally contemplates the following principles:

- Present high-level architectures, addressing the distinct role of specific technologies;
- Include a variety of data science tasks, such as data mining, statistical analysis, machine learning, real-time visualization, and in-memory analysis;
- Combine the benefits of different tools for different tasks;
- Bring analysis closer to the data, in order to avoid moving data;
- Distribute processing and storage across different nodes in a cluster;
- Assure coordination between data and processing nodes to improve scalability, efficiency, and fault-tolerance.

2.4.1.1 Big Data Life Cycle and Requirements

There are several considerations throughout the life cycle of Big Data, significantly different from traditional environments. Dealing with Big Data requires new approaches, which are discussed in this subsection.

2.4.1.1.1 General Steps to Process and Analyze Big Data

According to a survey including analysts at Microsoft (Fisher et al., 2012), Big Data analytics tasks can be grouped into five steps: acquire data; choose the architecture based on cost and performance; shape the data according to the architecture; write and edit code; and reflect and iterate on the results. Processing Big Data for analysis typically differs from processing traditional transactional data. As Krishnan (2013) claims, in traditional environments, data is explored, a model is designed, and a database structure is created. However, in Big Data environments, data is first collected and loaded into a certain storage system, a metadata layer is applied, and then a structure is created. There is no need to start by transforming data to properly fit a relational model, as transformations only occur after having everything stored in efficient storage systems. This represents a shift from a traditional ETL approach to an Extraction, Loading, and Transformation (ELT) approach. Figure 2.4 presents the Big Data Processing Flow according to Krishnan (2013).

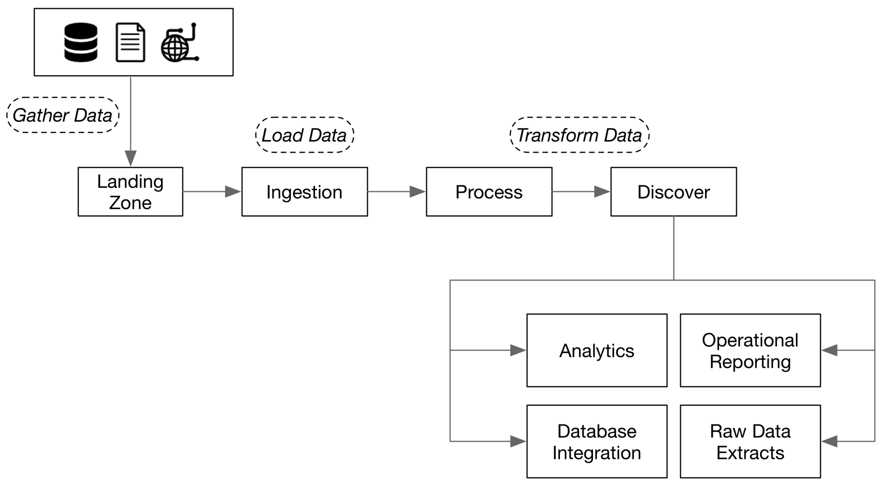


Figure 2.4. An overview of the Big Data Processing Flow. Adapted from (Krishnan, 2013; C. Costa & Santos, 2017a).

The Big Data Processing Flow starts by gathering data from multiple sources, such as Online Transaction Processing (OLTP) systems, multiple files, sensors, and the Web. This data is then stored in a landing zone capable of handling the volume, variety, and velocity of data, which is typically a distributed file system. Data transformations must occur on data stored in the landing zone, fulfilling the requirements of efficiency and scalability, and the subsequent results can then be integrated into analytical tasks, operational reporting, databases, or raw data extracts. In this context, Kimball and Ross (2013) mention relevant best practices regarding the Big Data life cycle:

- Plan a “data highway” with multiple caches - raw source (immediate), real-time cache (seconds), business activity cache (minutes), top line cache (24 hours), and DW or long time series cache (daily, periodic, and annual). Data will flow through these different caches, according to the business needs;
- Use Big Data analytics to enrich data before moving it to the next cache. For example, produce numeric sentiments from mining unstructured tweets. The opposite is also true, so that earlier caches can benefit from the less granular ones. Kimball and Ross (2013) claim that the performance implications of this enrichment should be further evaluated, as data should be moved from the raw source to the real-time cache according to the established time thresholds. Also, we can store multiple data sources, make them available for querying, manipulate them, use them to serve business, and then archive them;
- Adjust the data quality needs according to the latency requirements, i.e., complex data quality jobs take more time to complete than simpler ones focusing on individual values. However, Kimball and Ross (2013) also suggest that value should be added to data as soon as possible, using data integration tasks and including results from data mining, for example. There must be a balance between latency and business value;
- Big Data streaming analytics can be relevant for certain data flows, analyzing data and taking actions as it flows through continuous data streams (Kambatla, Kollias, Kumar, & Grama, 2014). In-database analytics can also be a relevant capability to exploit (Kimball & Ross, 2013).

Begoli and Horey (2012) complement these perspectives, stating that several analytical mechanisms should be included in Big Data solutions, ranging from statistical analysis to data mining and visualization. Moreover, processed data and insights can be made available using open and recognized standards, interfaces, and Web services. Regarding Big Data analytics, there is a vast set of available techniques that can be used to extract value from data. Data mining techniques, such as clustering, association rules, classification, and regression (Han, Pei, & Kamber, 2012) are still present in Big Data environments (Manyika et al., 2011), now with the challenge of distributing them to perform at scale (C. L. P. Chen & Zhang, 2014; Fan & Bifet, 2013). Achieving scalability in these techniques is what makes Big Data analytics different from traditional data analytics. The range of analytical mechanisms and the ambiguous terms to define them may lead to a completely new buzzword: data science. Techniques such as sentiment analysis, time series analysis/forecasting, spatial analysis, optimization, visualization, or unstructured analytics (e.g., text, audio, and video) (Gandomi & Haider, 2015), can all be present in the knowledge base of a data scientist (C. Costa & Santos, 2017b). These techniques are relevant in the Big Data life cycle to extract value from it.

2.4.1.1.2 Architectural and Infrastructural Requirements

The different steps to process Big Data, presented above, must be performed in Big Data environments, according to several requirements identified by Krishnan (2013):

- Absence of fixed data models, to adequately accommodate the complexity and size of data, regardless of its characteristics;
- Scalable and high-performance systems to collect and process data either in real-time or in batches;
- The architecture should support data partitioning due to the volume of data;
- Data transformations use scalable, efficient, and fault-tolerant mechanisms. The results should be stored in adequate systems, such as distributed file systems or non-relational database systems. Data reads should be efficient;
- Data should be replicated and shared across multiple nodes, to support fault-tolerance, multistep processing, and multipartitioning.

Kimball and Ross (2013) corroborate most of the requirements from Krishnan (2013), and add the following capabilities expected from Big Data environments: possibility to implement User-Defined Functions (UDFs) in several programming languages and to execute them over large datasets within minutes; load and integrate data at high rates; execute queries on streaming data; schedule tasks on large clusters; and support mixed workloads, including several ad hoc queries or strategic analysis from multiple users, while loading data in batches or in a streaming fashion.

Big Data solutions should be supported by an adequate infrastructure. Regarding this requirement, organizations can currently rely on cloud computing, either by using private, public, or hybrid clouds (Tien, 2013), in order to provide the underlying resources for massive computations (Hashem et al., 2015). Cloud models, such as Infrastructure-as-a-Service (IaaS), become relevant to accomplish several requirements in Big Data infrastructures, including scalability, commodity hardware, elasticity, fault-tolerance, self-manageability, high throughput, fast I/O, and a high degree of parallelism (Cuzzocrea et al., 2011; Krishnan, 2013). Commodity hardware also plays a relevant role in Big Data infrastructures, namely due to the lower costs of building shared-nothing architectures (Figure 2.5). Google's own papers about the Google File System (GFS) (Ghemawat, Gobioff, & Leung, 2003), MapReduce (Dean & Ghemawat, 2008), and Bigtable (F. Chang et al., 2008) served as inspiration for most of these requirements and for several Big Data technologies that will be presented later.

Kimball and Ross (2013) argue that traditional RDBMSs are not suitable for a wide range of Big Data use cases due to the requirements identified above (e.g., search ranking, sensors, social customer

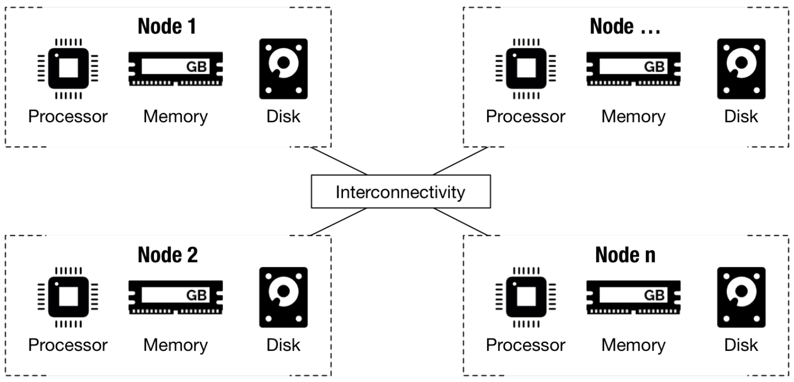


Figure 2.5. A shared-nothing architecture. Adapted from (Krishnan, 2013).

relationship management, document similarity testing, and loan risk analysis). Krishnan (2013) also claims that DWs based on traditional RDBMSs have several design limitations that imply architectural and infrastructural changes to process Big Data, since they cannot be distributed as efficiently as non-relational systems due to Atomicity, Consistency, Isolation, and Durability (ACID) compliance rules, and due to the fact that data partitioning in these systems often does not necessarily mean more scalability or workload reduction. Furthermore, the author mentions the fact that in many of these systems, the CPU and memory are often underused, and the way queries are designed typically increases the workload, such as executing queries with a star schema pattern on a Third Normal Form (3NF) database model, generating significant volume of I/O and inadequate network throughput. Kimball and Ross (2013) present the capabilities that existing RDBMSs vendors are including to extend their solutions for Big Data environments. The authors compare these extended versions with the most commonly recognized open source implementation of MapReduce, namely Apache Hadoop. This comparison is presented in Table 2.2.

2.4.1.2 The Lambda Architecture

The main idea behind the Lambda Architecture (Marz & Warren, 2015) is to think of a Big Data system as a series of layers that satisfy particular needs. As Figure 2.6 shows, the architecture is divided into three main components: batch, serving, and speed layers. In the batch layer, a master dataset stores all the data. As it is sometimes inefficient to read a dataset with possible Petabytes of data every time a query is executed, the architecture contains batch views in the serving layer, which are pre-computations of the master dataset. Instead of scanning the entire master dataset, the results

Table 2.2. Comparison between an extended RDBMS and Hadoop MapReduce. Adapted from (Kimball & Ross, 2013).

Characteristic	Extended RDBMS	Hadoop MapReduce
Proprietary	Mostly Proprietary	Open Source
Cost	Expensive	Less Expensive
Variety	Data must be structured	Does not require structure
Type of operations	Adequate for fast indexed lookups	Adequate for massive scans
Relational Semantics	Deep support	Indirect support (e.g., Hive)
Complex Data Structures	Indirect support	Deep Support
Transaction Processing	Deep support	Little or no support

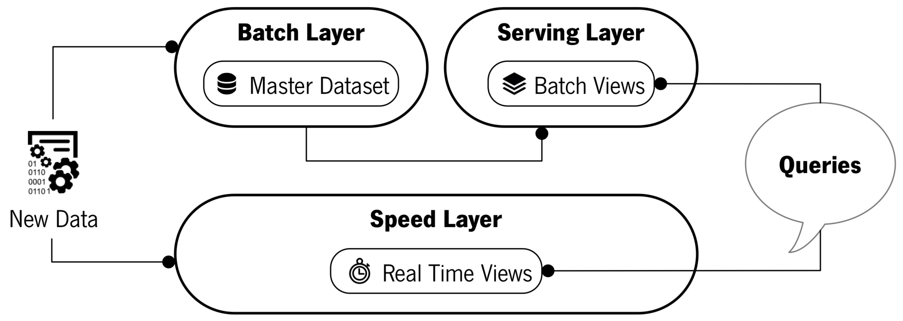


Figure 2.6. The Lambda Architecture. Adapted from (Marz & Warren, 2015; C. Costa & Santos, 2017a).

are returned from batch views with indexing support, thus random reads are possible. Therefore, the batch layer is not only responsible for storing an immutable and constantly growing master dataset, but also for computing functions on the same. As Marz and Warren (2015) highlight, creating the batch views is an high latency operation and should be performed in scalable systems. Then, the serving layer stores these batch views in a distributed database supporting batch updates and random reads.

However, with only these two layers, batch views would be quickly outdated, as new data takes time to propagate from the batch layer into the serving layer. This does not meet the requirements of low-latency (real-time) environments. Consequently, the authors propose the speed layer, which aims to compute functions on data in real-time. Rather than processing all the data at once, like the batch layer, the speed layer only processes recent data. To achieve the smallest possible latency, it does not even look at all the new data at once. Instead, it updates real-time views as new data becomes available, which is described as incremental computation. In order to retrieve current results, queries are answered by looking at the batch and real-time views, merging both results. Consequently, low-latency updates are taken into consideration, and as batch views are updated, real-time views can be discarded, since the authors claim that the speed layer is far more complex than the other two. Marz and Warren (2015) describe how to develop Big Data systems according to the principles of the Lambda Architecture, highlighting several technological aspects, as well as other guidelines:

- Store the rawest data to answer as much questions as possible, obtaining different summarizations and insights. Since Big Data technologies are scalable by nature, they can handle this requirement;
- Store untransformed data, since data integration and quality algorithms can be improved in the future;
- Make the master dataset immutable, i.e., only adding more data, without update or delete operations. By doing this, Marz and Warren (2015) claim that human fault-tolerance and simplicity are assured;
- Within the master dataset, store data as units called facts. They are atomic, timestamped, and uniquely identifiable. The authors describe how to strengthen the fact-based model with information about the types of facts and relationships between them through the use of a graph schema. Moreover, Marz and Warren (2015) also give guidelines about a possible folder and file structure for the master dataset, typically stored in a distributed file system.

2.4.1.3 Towards Standardization: the NIST Reference Architecture

The NIST Big Data Public Working Group (NBD-PWG), namely the Reference Architecture Subgroup, has been working on an open reference architecture for Big Data (NBD-PWG, 2015), in order to create a tool to facilitate the discussion of requirements, design structures, and operations for Big Data environments. According to the authors, the NBDRA is not a system architecture, but rather a common reference, which is not coupled with specific vendors, services, implementations, or any specific solutions. The NBDRA is presented in Figure 2.7, and the proposed taxonomy for its components is as follows:

- System orchestrator - provides requirements regarding policy, governance, architectural design, resources, business requirements, monitoring, and auditing activities. The system orchestrator may include actors such as business leadership, consultants, data scientists, and architects related to information, software, security, privacy, and network;

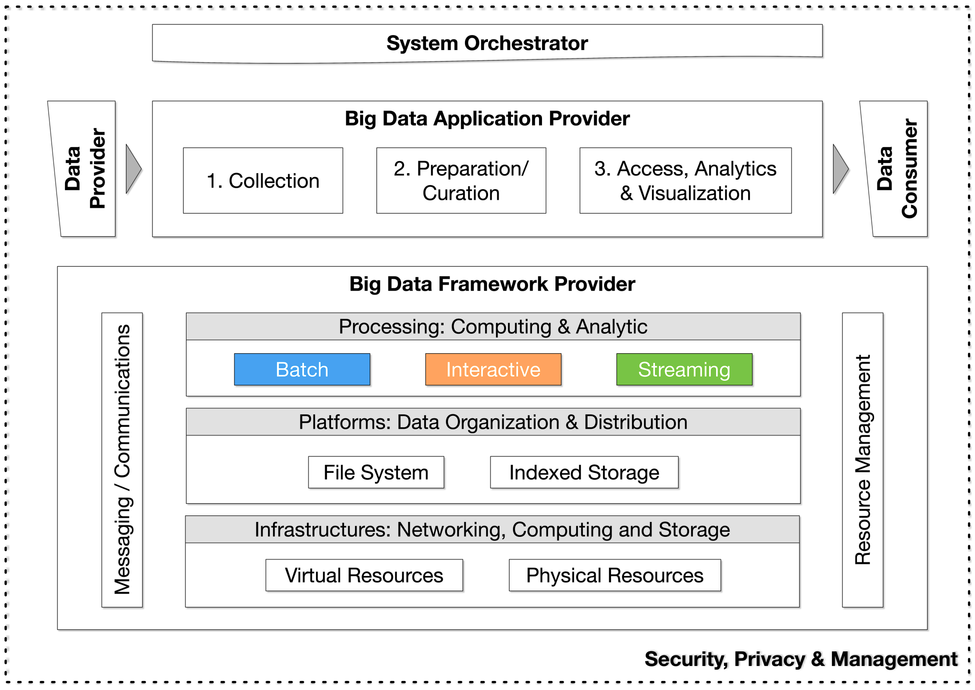

System Orchestrator
Data Provider
Big Data Application Provider
1. Collection
2. Preparation/ Curation
3. Access, Analytics & Visualization
Data Consumer
Big Data Framework Provider
Messaging / Communications
Processing: Computing & Analytic
Batch
Interactive
Streaming
Platforms: Data Organization & Distribution
File System
Indexed Storage
Infrastructures: Networking, Computing and Storage
Virtual Resources
Physical Resources
Resource Management
Security, Privacy & Management

Figure 2.7. The NIST Big Data Reference Architecture. Adapted from (NBD-PWG, 2015; C. Costa & Santos, 2017a).

- Data provider - makes data available through different interfaces, including several data sources (e.g., raw data or previously transformed data). The data provider can be internal or external to the organization;
- Big Data application provider - executes the manipulations in the data life cycle to meet the requirements established by the system orchestrator. In this component, several capabilities are combined to create specific data solutions. While the general activities may remain similar to traditional data processing contexts, Big Data methods and techniques are considerably different due to scalability concerns;
- Big Data framework provider - is composed of general resources or services to be used by the Big Data application provider. This is the role whose changes are more noticeable because of Big Data (NBD-PWG, 2015), due to the relevance of the infrastructure, data platforms, and processing frameworks. Different technologies can be used and hybrid approaches can emerge, providing flexibility and meeting the requirements of the Big Data application provider;

- Data consumer - benefits from the value of the Big Data system. The same type of interfaces used by the data provider can also be exposed to the data consumer, after value has been added to the original data sources.

The NBDRA has two fabrics encapsulating the aforementioned components: a security and privacy fabric, which affects all the components of the NBDRA and interacts with the system orchestrator (policy, requirements, and auditing), the Big Data application provider, and the Big Data framework provider (development, deployment, and operation); and a management fabric responsible for tasks such as provisioning, software management, or performance monitoring, which involves considerations at scale about the system, data, security, and privacy, while maintaining a high level of data quality and accessibility.

The NBDRA contains five components connected by interoperable interfaces (services) and enveloped by the two fabrics mentioned above. It supports a variety of business environments and facilitates the understanding of how Big Data solutions complement existing approaches and differ from them. To develop this proposal, the authors analyzed a wide range of existing Big Data architectures from industry, academy, and government (NBD-PWG, 2015).

2.4.2 Big Data Technologies

This subsection highlights several technologies related to Big Data, including Apache Hadoop and related projects, several distributed databases, and other tools for Big Data analytics.

2.4.2.1 Hadoop and Related Projects

As already mentioned, Hadoop is an open source Apache project based on GFS and MapReduce (Bakshi, 2012). Hadoop contains two main components: the Hadoop Distributed File System (HDFS) and a distributed processing framework named Hadoop MapReduce. Hadoop can store and process vast amounts of data by distributing storage and processing across a scalable cluster of multiple nodes built with commodity hardware. In HDFS, files are divided into blocks distributed and replicated across nodes. HDFS assures many requirements identified above, such as fault-tolerance and availability, for example. Hadoop MapReduce is a programming model and an execution engine for processing large datasets stored in HDFS, based on the divide and conquer method, dividing a complex problem into many simpler problems, and then combining each simpler solution into an overall solution to the main problem. These are called the Map and Reduce steps (C. L. P. Chen & Zhang, 2014). Regarding HDFS, there are two types of nodes in the cluster: a NameNode, which is responsible for storing metadata about blocks and nodes; and a DataNode, which stores data blocks (Bakshi, 2012). Regarding Hadoop MapReduce, there are also two types of nodes, namely a JobTracker that schedules jobs and distributes tasks across slaves called TaskTrackers (C. L. P. Chen & Zhang, 2014).

Over the years Hadoop has evolved considerably, including the transition from MapReduce to YARN (Hashem et al., 2015). YARN rethinks the JobTracker and TaskTracker components, replacing them with a ResourceManager, a NodeManager, and an ApplicationMaster, to solve some problems in Hadoop MapReduce, such as scalability on large clusters or support for alternative programming paradigms (Krishnan, 2013). Apart from that, Hadoop has several related projects, as Figure 2.8 demonstrates, also highlighting their main features (Apache Hadoop, 2018).

Other related projects not present in Figure 2.8 may include: Flume, a service to collect, aggregate, and move large amounts of log data; Oozie, a workflow and coordination system for jobs in Hadoop; HCatalog, a metadata layer for data stored in Hadoop, built on top of the Hive metastore; Sqoop, a connector to integrate data from other existing platforms, such as the DW, metadata engines, enterprise systems, and transactional systems (Krishnan, 2013). There are also more projects that can interact with Hadoop's interfaces or be co-located with it, such as projects for real-time stream processing or interactive ad hoc analysis. Since real-time data processing is becoming increasingly relevant to organizations (Chandarana & Vijayalakshmi, 2014), Storm is a real-time computation system to process streams with high throughput and low latency. Kafka, on the other hand, is a messaging/queuing system to produce and consume messages between processes, in an asynchronous and fault-tolerant manner (Marz & Warren, 2015).

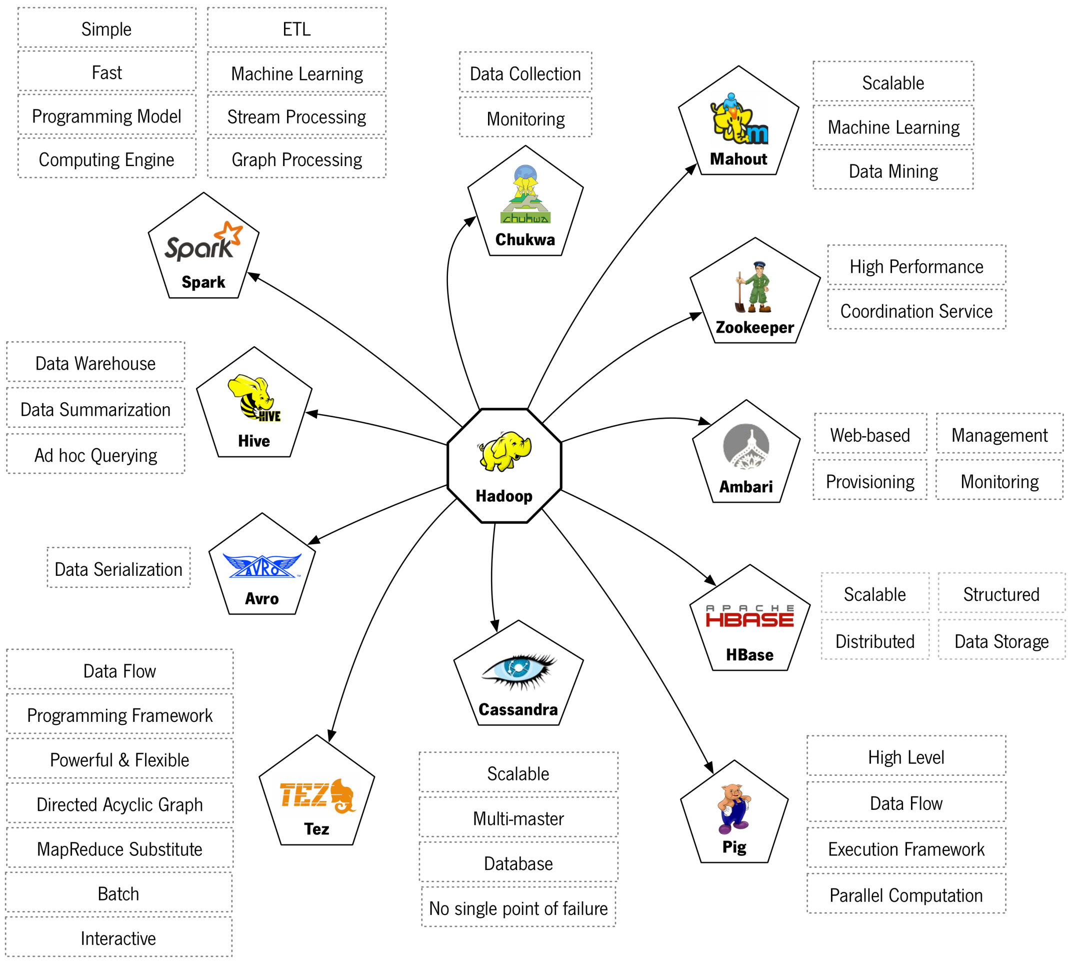


Figure 2.8. The Apache Hadoop ecosystem. Adapted from (C. Costa & Santos, 2017a).

2.4.2.1.1 SQL-on-Hadoop and Interactive low-latency Queries

Interactive and low-latency ad hoc analysis over large datasets is a relevant scenario in organizations. Occasionally, users do not know the queries in advance and need to execute ad hoc queries within seconds, even at scale. There is a trend named SQL-on-Hadoop (Floratou, Minhas, & Özcan, 2014) that is related to the implementation of distributed SQL engines for interactive ad hoc analysis of large datasets stored not only in Hadoop, but also in distributed databases (e.g., Not Only SQL – NoSQL). Many SQL-on-Hadoop systems are available under open source licenses, including: Hive on Tez (Huai et al., 2014); Presto (Presto, 2016); Impala (Kornacker et al., 2015); Drill (Hausenblas & Nadeau, 2013); and Spark SQL (Armbrust et al., 2015). These systems are able to combine data from multiple sources like HDFS files, NoSQL databases, SQL databases, Kafka, among many others, which means that in a single query they can combine not only data from different systems, but also batch and streaming data. Consequently, SQL-on-Hadoop systems play a relevant role in BDWing systems, as it will be discussed later in this work. Moreover, besides SQL-on-Hadoop systems, there are other similar technologies targeting interactive ad-hoc querying, such as Druid, a columnar store that provides real-time aggregation and indexing at data ingestion time (F. Yang et al., 2014).

2.4.2.1.2 Hadoop Security

Still related to Hadoop, there are several security projects. Hortonworks (2016) establishes five pillars for security in Hadoop: administration, authentication, authorization, auditing, and data protection. Kerberos, Apache Knox, and Apache Ranger are highlighted as projects related to these five pillars, in order to assure a secure Hadoop environment. Kerberos can be used to authenticate users and resources within Hadoop clusters. Apache Knox complements Kerberos, by blocking services at the perimeter of the cluster and hiding the cluster's access points from end users, thus adding another layer of protection for perimeter security. Finally, Ranger provides a centralized platform for policy administration, authorization, auditing, and data protection (e.g., encrypted files in HDFS).

2.4.2.2 Distributed Databases

Database technology has evolved significantly towards handling datasets at different scales and supporting several applications that may have high needs for random access to data (M. Chen et al., 2014; Hashem et al., 2015). NoSQL databases have become popular mainly due to the lack of scalability in RDBMSs, since this new type of databases provides mechanisms to store and retrieve a large volume of distributed data (Hashem et al., 2015). The relevant factors that motivated the appearance of NoSQL databases were the strictness of the relational model and the consequent inadequacy to store Big Data. NoSQL databases are seen as distributed, scalable, elastic, and fault-tolerant storage systems. They satisfy an application's need for high availability even when nodes fail, appropriately replicating data across multiple machines (Kambatla et al., 2014). Relational databases will certainly evolve and some organizations (e.g., Facebook) are using mixed database architectures (M. Chen et al., 2014). Combining the benefits of both storage systems is a current research trend, as already mentioned (Cuzzocrea et al., 2011). A recent term is emerging, NewSQL, which combines the relational data model with the benefits of NoSQL systems, such as scalability (Grolinger, Higashino, Tiwari, & Capretz, 2013). NoSQL and NewSQL databases are mainly designed to scale OLTP-style workloads over several nodes, fulfilling the requirements of environments with millions of simple operations (e.g., key lookups, reads/writes of one record or a small number of records) (Cattell, 2011).

This phenomenon changed the way databases are currently designed. While a RDBMS complies to ACID properties (Krishnan, 2013), a NoSQL database, as a distributed system, typically follows the considerations of the Consistency, Availability, and Partition tolerance (CAP) theorem: "any networked shared-data system can have at most two of three desirable properties" (Brewer, 2012). These properties include: consistency, equivalent to a single up-to-date copy of the data; high availability of that data; and tolerance to network partitions. As Brewer (2012) claims, the CAP theorem served the purpose of leveraging the design of a wider range of systems and trade-offs, in which the NoSQL movement is a clear example. The fact that two of the three properties should be chosen was always misleading, states the author, since it tends to simplify the "tensions among properties". These properties are more continuous than binary and, therefore, they can have many levels. CAP only prohibits perfect availability and perfect consistency in the presence of network partitions.

Consequently, the CAP theorem serves the purpose of considering combinations of consistency and availability that fit in a certain scenario. The author highlights that choices between consistency and availability can vary within a certain system and according to specific data or users, for example. Brewer (2012) clarifies this misconception and discusses the relationship between ACID and CAP, stating that choosing availability only affects some of the ACID's guarantees. These design considerations are intrinsic to NoSQL databases, and each may be designed differently regarding these choices.

There are several NoSQL databases, so enumerating and evaluating all of them becomes a nearly impossible task. Over 120 NoSQL databases were known in 2011 (Tudorica & Bucur, 2011). Currently, it is estimated that the list of NoSQL databases has more than 225 elements (NoSQL, 2018). Taking this into consideration, NoSQL databases are typically divided into four data models, which are described as follows, along with several examples:

- Key-value model - values are typically stored in key-value pairs. The key uniquely identifies a value of an arbitrary type. These data models are known for being schema-free, but may lack the capability to adequately represent relationships or structures, since queries and indexing are assured through the key (Grolinger et al., 2013). Each key is unique and queries are tightly coupled with keys (M. Chen et al., 2014).
    - Examples: Redis; Memcached; BerkeleyDB; Voldemort; Riak; and Dynamo.
- Column-oriented model - a columnar data model can be seen as an extension of the key-value model, adding columns and column families, providing more powerful indexing and querying due to this addition (Krishnan, 2013). This design was largely inspired by Bigtable (M. Chen et al., 2014; Grolinger et al., 2013), but that does not mean that all column-oriented databases are fully inspired by it (e.g., Cassandra adopts design aspects from both Dynamo and Bigtable).
    - Examples: Bigtable; HBase; Cassandra; and Hypertable.
- Document model - suitable for representing data in document format. JSON is here frequently used. It can contain complex structures, such as nested objects, and it also typically includes secondary indexes, thus providing more query flexibility than the key-value data model (Grolinger et al., 2013).
    - Examples: MongoDB; CouchDB; and Couchbase.
- Graph model - based on the graph theory, in which objects can be represented as nodes, and relationships between them can be represented as edges (Krishnan, 2013). Graphs are specialized in handling interconnected data with several relationships (Grolinger et al., 2013).
    - Examples: Neo4j; InfiniteGraph; GraphDB; AllegroGraph; and HyperGraphDB.

Regarding NewSQL, as the name implies, these databases are based on the relational model (Grolinger et al., 2013), offering either a pure relational view of the data (e.g., VoltDB, Clustrix, NuoDB, MySQL Cluster, ScaleBase, and ScaleDB) or similar (e.g., Google Spanner). According to Grolinger et al. (2013), sometimes, interactions with these databases occur in terms of tables and relations, but they might use different internal representations (e.g., Key-value store). Different NewSQL databases support different SQL compatibility levels, such as unsupported clauses or other incompatibilities with the standard. Similarly to NoSQL, NewSQL databases can scale accordingly by adding more nodes to the cluster.

2.4.2.3 Other Technologies for Big Data Analytics

By describing Hadoop and its related projects, several technologies for Big Data analytics were already inherently identified: streaming analytics (e.g., Spark Streaming and Storm); data mining and machine learning (e.g., Spark and Mahout); DWing (e.g., Hive); interactive ad hoc analysis (e.g., Hive on Tez, Impala, Presto, Drill, and Spark SQL) (Santos et al., 2017; Soliman, 2017); data flow (e.g., Pig). However, no data visualization tools were presented yet.

Regarding Big Data visualization, several mashup tools can be highlighted, such as Datameer, FICO Big Data Analyzer (former Karmasphere), Tableau, and TIBCO Spotfire (Krishnan, 2013). These mashup tools can integrate data from multiple sources into a single picture. As Krishnan (2013) highlights, there is also the possibility of visualizing Big Data with statistical tools, like R or SAS, for example, taking advantage of their capabilities. Other tools are also briefly mentioned in the literature, such as Jaspersoft Business Intelligence (BI) Suite and Pentaho Business Analytics (C. L. P. Chen & Zhang, 2014). Certainly, many other visualization tools exist and may be adequate for Big Data visualization, such as Excel and Power BI, JavaScript libraries, or Python's plot capabilities.

Besides data visualization, there are other tools to extract, load, transform, and integrate data before analytical tasks. Talend Open Studio for Big Data is an example of such tool (C. L. P. Chen & Zhang, 2014). Moreover, apart from the aforementioned tools related to Hadoop for data mining and machine learning, other alternatives identified in the literature may include: MADLib and EMC Greenplum (Begoli & Horey, 2012); R, MOA, WEKA, and Vowpal Wabbit (Fan & Bifet, 2013); data mining tools from SAS or IBM (Krishnan, 2013); Rapidminer; and KNIME (M. Chen et al., 2014). Some of these tools like R and WEKA are not scalable by default, and they are also used in traditional data mining and machine learning environments, where processing large training sets is not a significant concern. Over time, these tools were extended with several connectors for scalable Big Data stores and packages for distributed processing (e.g., SparkR, RHadoop, RHive, distributedWekaBase, distributedWekaHadoop, and distributedWekaSpark), but by default, without these extensions, they are better suited for small to moderate datasets. This does not mean that they are not useful in Big Data mining, quite the opposite, but the volume of data that serves as training and testing sets should be considered (preprocessing large datasets can be useful in these cases). The same principle applies to other non-distributed algorithms implemented in any other language like Python or Java, for example. It should be remembered that one of main challenges regarding the Big Data life cycle is to scale the algorithms to extract value from data (Hashem et al., 2015).

This page was intentionally left blank

Chapter 3. Big Data Warehousing

The DW concept has a long history, and the need to access, analyze, and present data in appropriate forms to support fact-based decision-making exists in organizations for a long time (Kimball & Ross, 2013). A DW is a repository that consolidates information about the organization, leveraging a vast range of analyses developed by several users. Traditionally, it is a database that maintains an historical record of the organization, which is periodically extracted from OLTP sources. The DW is designed to access multiple records at a time and it is optimized to support analytical tasks (e.g., predefined or ad hoc queries, reports, OLAP, and data mining). OLAP is a common analytical task associated with the DW, mainly consisting in multidimensional structures capable of executing several tasks according to the desired view of the data (Santos & Ramos, 2009). Summarizing, the DW concept is commonly defined as a "subject-oriented, integrated, non-volatile, and time-variant collection of data in support of management's decisions" (Inmon & Linstedt, 2014), as well as a "single version of the truth" (Kimball & Ross, 2013).

Since the last decades, traditional DWs are recognized as the enterprise data asset, but the evolution of advanced analytics (e.g., data mining, statistics, and complex queries), increasing data volume, and real-time needs to analyze fresh data are driving changes in DW architectures (Russom, 2014). Nowadays, the DW is evolving, being extended and modernized to support advancements in technologies and business requirements, in order to prove its relevance in the era of Big Data. DW modernization is on top of the priorities for professionals, and surveys show that DWs are evolving dramatically and there are several opportunities to improve and modernize them, since organizations view them as relevant to today's businesses (e.g., analytics, data-driven decision-making, operational efficiency, and competitive advantages) (Russom, 2016).

However, in this modernization process, some challenges arise, such as inadequate governance of data, lack of skills, cost of implementing new technologies, and difficulties in conceiving a modern solution that can ingest and process the ever-increasing amounts or types of data. According to Russom (2016), the average DW stores between 1TB and 3TB of data and it will store between 10TB and 100TB until 2018. Organizations need to consider the modernization of their DW architectures when some of the following questions arise (Chowdhury, 2014):

- Is the current platform limited by the amount of data to process?
- Is the DW a useful repository for all the data that is generated and acquired? Or is some data being left unprocessed due to current restrictions?
- Do we want to analyze non-operational data and use new types of analytics?
- Do we need to ingest data quicker?
- Do we need to lower the overall cost for analytics?

Therefore, among the community, the concept of BDW is emerging. This chapter presents works related to the concept of BDW, including: identification of characteristics, requirements, and guidelines for design change and implementation; proposals of DWs on NoSQL databases; advancements and benchmarks in storage technologies for BDWs; optimizations in OLAP, query processing, and execution mechanisms for BDWs; and some implementations in specific contexts. The following sections are organized by the main topics identified in the literature, and the content within each section is sorted first by date then by author name, unless there are more than one publication for the same author. In this case, they will appear together in the text.

3.1 Characteristics and Design Changes for Big Data Warehouses

This section presents several works related to the characteristics and the need for design changes in DWs to fully support Big Data environments. Research in this topic is still in its infancy and there is no common approach to design BDWs. Consequently, among the related work, there are authors who discuss this need and the general changes that have to occur, giving non-structured guidelines to design BDWs or to revisit traditional modeling techniques. There are also works that discuss logical architectures or propose implementation of traditional DWs with Big Data extensions.

- Kearney (2012) states that organizations can create significant value by modernizing their DWs with Big Data technologies to analyze data, but this demands Massively Parallel

Processing (MPP) architectures. The author suggests IBM Netezza, a DW appliance developed by IBM.

- Kobielus (2012) shows the relevance of Hadoop for the next generation DW, due to its diverse set of possible roles, such as ETL, data staging, or preprocessing of unstructured data. MPP, in-database analytics, mixed workloads, and flexible storage are also mentioned as main features in these DW architectures, which aim to provide a complete view of the truth about structured and unstructured data.

- Baboo and Kumar (2013) highlight the need to study storage options and use of Big Data in DWs. The authors state that when DWs can adequately handle a high volume of data and real-time needs, organizations will be able to have further insights and make better business decisions. The authors provide an overview of what is Big Data analytics and its advantages, calling the attention for future research related to the DW.

- Cuzzocrea, Bellatreche, and Song (2013) recognize DW and OLAP over Big Data as an emerging research topic. Among several issues related to OLAP over Big Data, some concerns associated with the design of DWs are highlighted, such as the size of the fact tables and innovative ways to compute aggregations, which becomes even more relevant in Big Data environments. Cuzzocrea and Moussa (2017) also discuss some challenges for multidimensional database modeling in the age of Big Data, calling the reader's attention to several challenges such as: schema-less or dynamic schema data; dimensionality problems (cubes with hundreds of high cardinality dimensions); the need for intelligent recommendation systems for data partitioning and computation of summarized data (e.g., materialized views); real-time processing; and sophisticated data visualization.

- Foo (2013) states that in the era of Big Data, organizations have available a set of techniques and technologies such as Hadoop, stream processing, and high-performance analytics, which can deliver fast insights. This leads to implementing a federation of multiple repositories and technologies to serve specific purposes. The traditional DW is complemented with these new technologies and, therefore, interoperability between them becomes crucial.

- Goss and Veeramuthu (2013) describe the current DW solution in a semiconductor manufacturing organization, and highlight the need for new solutions based on Big Data concepts for better data transparency across the organization, experimental and automated data analysis, or advanced simulations. The authors consider different solutions, such as Big Data appliances, Hadoop, or massive in-memory databases. They conclude by appealing to vendors to work together, ditching proprietary infrastructures and offering plug-and-play components.

- Kimball and Ross (2013), although mostly focused on traditional DW modeling, provide relevant best practices to plan a DW in Big Data environments, such as follows:

  - Consider complex analytics, not only reporting or ad hoc query; avoid legacy environments, preventing possible technological changes as much as possible; and promote the use of sandbox results, where data scientists can work freely;
  - Plan data highways, i.e., different caches with different latency requirements, as shown in the previous chapter; think about extracting facts, even from unstructured content; be aware of data quality and value; and implement streaming mechanisms;
  - Still approach a modeling problem as dimensions and facts, and integrate structured and unstructured data.

- Krishnan (2013) study the need to redesign traditional DWs, in order to address significant challenges (e.g., data types, data volume, performance, fault-tolerance, infrastructural cost, and user requirements). The author states that these next generation DWs will include data from several sources and will be a collection of multiple techniques and technologies, such as RDBMSs, Hadoop, NoSQL, data mining, text mining, reporting, visualization, among other. The author also discusses some real-world examples that used multiple techniques to integrate Big Data, and concludes by claiming that there is no solution to fit all contexts.

Examples of possible logical architectures are presented, as well as the advantages and disadvantages of each one of them (Krishnan, 2013):

- Architecture based on external integration, wherein traditional technologies are maintained, and a new platform for processing Big Data is integrated with them through a data bus. Tasks are executed in two distinct platforms, and when data is being explored, the data bus assures adequate integration between them, through appropriate metadata processing. Generally, this approach provides a modular and heterogeneous design, but implementing the data bus and maintaining an adequate integration between the two platforms may become complex;
- Hadoop/NoSQL and RDBMS architecture: this approach is similar to the one presented above, but instead of a complex data bus which integrates data at the time of exploration, the RDBMS and Hadoop/NoSQL are integrated through a connector that exchanges data between the two systems. However, if the connector does not perform adequately, performance becomes severely affected;
- Big Data appliances: these are black box solutions which handle rigorous and complex workloads associated with Big Data and current RDBMSs. There are several physical architectures according to each vendor (e.g., Teradata, IBM, and Oracle), but the logical architecture typically consists in a group task between Hadoop, NoSQL, and the RDBMS, in order to solve several challenges associated with Big Data. These solutions are mainly configured according to the user's requirements and making subsequent changes to the architecture can be difficult, as stated by Krishnan (2013);
- Data virtualization architecture: it hides the details about how data is stored, since the same becomes available to users as if it was stored in a single location, hiding implementation details. This approach can provide easier maintenance for analytical workloads, but a lack of governance may occur in multiple data silos, as well as decreases in performance.

- Mohanty, Jagadeesh, and Srivatsa (2013) compare the BDW with a traditional DW, highlighting some significant differences, such as the capability to perform exploratory analysis (e.g., sandboxes), deliver fast insights, and prove business hypothesis based on multiple sources of ever-increasing data, in a low-latency and scalable way. The authors present a conceptual BDWing architecture, which mainly consists in the identification of several techniques and technologies discussed in the literature (e.g., real-time and Hadoop), illustrating their coexistence with the traditional enterprise DW.

- Sun, Hu, Ren, and Ren (2013) discuss mainstream implementations of different architectures. First, the authors present the architecture dominated by MPP databases (e.g., Greenplum), which can use MapReduce capabilities in their database engines. Second, the architecture dominated by MapReduce is presented, where Hive is given as an example, providing a SQL interface on top of MapReduce. Finally, an integrated architecture is discussed, wherein the advantages of the other two are fully harnessed (e.g., HadoopDB, Vertica, and Teradata). The authors envisage future research, such as an adequate integration between data models and query processing, and pre-computation or indexing of multidimensional data.

- Chowdhury (2014) states that traditional infrastructures are not able to capture, manage, and process Big Data within reasonable time. The author describes Big Data technologies based on Hadoop, including IBM solutions, which can be used to augment existing DWs built on top of traditional databases. Several examples of IBM solutions are mentioned, showing their relevance to complement traditional DWs.

- T. K. Das and Mohapatro (2014) highlight the need to explore the capabilities of Hadoop, in order to handle Big Data and then integrate it into an existing DW. Therefore, Hadoop is seen as a mean to achieve efficient ETL processes for unstructured datasets with significant volume. An interface between Hadoop and a DW is illustrated, and the authors also state that the DW can be built on top of Hadoop, but there are no specific details about the interface or implementation, such as data flows, models, or proposed technologies. Specific results are also not mentioned.

- Golab and Johnson (2014) review recent research problems regarding data stream warehousing, motivating the need for a DW that is updated in near real-time, rather than during downtimes, also describing possible system architectures. The authors state that this concept aims to deal with data volume and velocity, contemplating not only issues in common DWs (e.g., storing and querying significant amounts of historical data), but also dealing with stream processing issues, such as handling ordered data, consistency, and near real-time response, as well as supporting alerts, materialized views, and complex analytics, for example. Golab and Johnson (2014) present three types of approaches:

    - Start with a Database Management System (DBMS) and extend it with the ability to load and query data arriving in near real-time;
    - Start with a data stream engine and then add persistent storage;
    - Start with a technology such as Hadoop and add stream processing.

  The authors highlight several optimizations needed in a data stream warehousing, such as fast ETL, efficient data layouts, maintenance of materialized views (incremental or recomputation), concurrency control, and scalability. According to them, several open problems deserve attention, including the exploration of hybrid architectures, the use of data mining and machine learning in a data stream warehouse, and managing the complexity of having multiple sources.

- Inmon and Linstedt (2014) extend the Data Vault methodology to design, manage, and implement a DW. According to the authors, Data Vault 1.0 was mainly focused on data modeling, while this second version extends it with agile techniques from software development and minor changes to ensure that modeling techniques work with Big Data requirements (e.g., unstructured data and NoSQL). They claim that in Data Vault's 2.0 architecture, platforms such as Hadoop currently fit in as an ingestion and staging area for any data that can proceed to the DW, or as a place to perform data mining and text mining, storing their subsequent results into relational database engines. Inmon and Linstedt (2014) state: “eventually, it will be a system capable of housing both relational and non-relational data simply by design”.

In their approach, the authors show that they can provide platform integration between an RDBMS and NoSQL platforms using hash keys, allowing for cross-system joins between them. Consequently, their idea is to allow current organizations to augment their infrastructure, maintaining current RDBMS engines.

- O’Leary (2014) discusses the concept of Big Data lake, comparing it with a traditional enterprise DW. This concept is an analogy to a water lake, where data streams fill the lake and several users examine it, diving in and taking samples, regardless of its lack of structure. In contrast, O’Leary (2014) sees the DW as a costly add-on to the enterprise, typically based on a single source to accommodate a particular set of queries, in a more structured manner.

  The author also claims that some challenges arise in a Big Data lake, given that the lack of structure causes problems to many statistical and machine learning packages, which sometimes are not designed for distributed environments. Moreover, in a Big Data lake, data duplication, redundancy, and inconsistency may raise significant problems. Finally, the author also presents some examples where artificial intelligence can be applied to the Big Data lake, such as follows: generate tags to facilitate data usage and definition; extract additional information from different data sources (e.g., temporal patterns); give structure to unstructured data (e.g., extracting sentiments from Twitter data); improve data quality; and discover new business insights using machine learning.

- O’Sullivan, Thompson, and Clifford (2014) present several data modeling considerations for Big Data deployments, including BDWs. The authors focus on both transactional and unstructured data, presenting some schema considerations for an adequate integration between Hadoop and RDBMS-based DWs. The work also highlights an interesting set of future needs for BDWing, including the evolution of data models and modeling methods, and the technologies in which these models are deployed.

- Russom (2014) discusses the results of a survey from The Data Warehousing Institute (TDWI) about the evolution of DW architectures. Many professionals from several industries answered the survey. Russom (2014) identifies several priorities for DW architectures, including: successful DW architectures should focus on both physical (e.g., server planning) and logical layers (e.g., data models); analytics is the main driver to evolve traditional DWs, as well as Big Data and real-time operations; an architecture can have a mix of approaches and standards; and Hadoop or NoSQL are great additions to traditional DWs, but it is not expected that these new technologies replace the old ones completely.

  Russom (2016) presents a report of several practices and strategies for DW modernization, resulting from a survey similar to Russom (2014). Several practices are discussed according to the responses of organizations, including the modernization of DWs by augmenting or replacing existing platforms. According to the survey, for some organizations, the adoption of new data platforms through a cloud or Software-as-a-Service (SaaS) paradigm provides another relevant feature: elasticity with lower costs. Furthermore, 32% of the 473 respondents state that they do not plan to replace their current DW primary platform, while 9% already replaced it, and 43% plan to replace it within 3 years. According to Russom (2016), "rip and replace is real and will become more common", such as migrating from a traditional RDBMS to a newer one, to a new DBMS, or to Hadoop, although according to the author, the latter was only found in a few rare cases, since Hadoop typically emerges as a complementary DW platform. Furthermore, it is highlighted that RDBMSs were still preferred among organizations. The survey concludes by highlighting several priorities to modernize the DW:

    - Add capacity for growing data, users, and analyses, satisfying the requirements of scalability and velocity;

- Deliver new and improved analytical capabilities, along with reporting and data integration;
- Evaluate if the technology adequately satisfies the business requirements before adopting it, taking performance and cost into consideration;
- Complement the traditional DW with other platforms, migrating data and balancing workloads, which requires thinking about a large-scale architecture and how data flows through different platforms;
- Consider Hadoop for several roles in a DW environment (e.g., data staging, ETL, and massive parallel execution engine), in order to complement the traditional DW, and not necessarily to replace it.

▪ Clegg (2015) discusses the challenge that Big Data presents to DW architectures, stating that it would be a mistake to discard decades of architectural best practices based on the assumption that storage for Big Data is not driven by data modeling. The author argues that a significant amount of data for analytics and reporting will remain relational. However, building an adequate architecture has become complex, due to the variety of available techniques and technologies (e.g., DW appliances, Hadoop, NoSQL, and real-time analytics). Therefore, DW architectures are entering in a new phase, since Big Data has finally fractured the traditional enterprise DW, states Clegg (2015), due to the use of Hadoop for data mining and batch operations, data marts for domain-specific applications, or NoSQL for real-time and time series data, most of the time with a combination of cloud and on-premises solutions. Vendors typically claim to have the solution to an organization's specific problem. Therefore, organizations moved from an integrated DW to a federation of different technologies addressing different use cases. According to Clegg (2015), Gartner called this the Logical DW (Beyer, 2011). The author states that we moved away from a data-driven view of the DW to a use case driven approach, and the danger of uncoordinated data silos emerge, meaning that much of the analysis takes place outside the main data store.

Consequently, replacing parts of the DW architecture with Hadoop and scalable databases leads to a "lift and shift" replacement strategy. Previously, data modeling was the main concern, but nowadays, the concern seems to be finding the right technology to meet demands. The author highlights the need to design a grand architecture and plug requirements into it, according to valuable use cases. This is a period of transition for DW architectures, being unknown if stability will be reached soon, but use case driven approaches seem to be the best strategy for now, states Clegg (2015).

- Golov and Rönnbäck (2015, 2017) discuss the anchor modeling strategy for highly normalized MPP databases in Big Data contexts, which allows for high-performance ad hoc queries, as demonstrated using systems like HP Vertica. The authors also present the limitations regarding single cluster uses and ETL issues, which can be overcome with some guidelines provided by the authors.

- P. Hu (2015) studies the cooperation between Hadoop and a traditional DW, in order to solve the performance issues of the latter. The author uses Sqoop for data collection and transmission, and relies on HDFS and Hive for storing data, although no detail is provided regarding how data flows through the system or how it is stored. A logical architecture is presented, where it can be seen that unstructured data should be stored in Hadoop, and structured data should be stored in the traditional DW, assuring communication between them. However, no explicit details are given regarding how this communication occurs, and although the author states that a prototype proves the feasibility of the proposed architecture, the evaluation method and results are not clearly presented. It would be interesting to discuss the performance of an Hadoop and RDBMS architecture Krishnan (2013) based on a connector between the two. The cooperation between Hadoop and a traditional DW, specifically in ETL processes, is also a relevant research topic discussed in other contributions (Houari, Rhanoui, & Asri, 2017).

- Jukic, Sharma, Nestorov, and Jukic (2015) also focus on how Big Data can augment and enrich the analytical capabilities of traditional DWs. Big Data is seen as a source for the DW, and Hadoop as a part of the ETL tools.

- Jukic, Jukic, Sharma, Nestorov, and Arnold (2017) explore and evaluate the use of columnar database technology and fully denormalized fact tables. Evaluating this approach using Greenplum, an MPP database, the authors arrive to the conclusion that a fully denormalized approach can bring considerable improvements to ETL processes and OLAP queries, namely better performance due to the lack of join operations. ETL processes also become simpler, since they avoid complex concepts like Slowly Changing Dimensions (SCDs). Although the full denormalization of fact tables, i.e., completely flattening the dimensions and facts into a single table, is arguably a well disseminated guideline in Big Data contexts, the reality is that such approach is also discussed in traditional DWing contexts (J. P. Costa, Cecílio, Martins, & Furtado, 2011), in order to avoid the processing costs of join algorithms and the additional random and sequential I/O operations when joins cannot be processed in-memory, while often assuring minimal network data exchange operations, which is relevant in distributed systems like Hadoop and other related projects.

- Tardío, Mate, and Trujillo (2015) present a methodology to avoid failure in Big Data projects, in which they identify common problems, best practices, and methods, aiming to increase the success of new initiatives. They propose a methodology to manage, analyze, and visualize Big Data, validating the approach through a case study based on electricity consumption. The proposed methodology consists of five phases: define data stages; acquire and manage data sources; add value to data; select and implement a BDW; develop visualizations for Big Data. Then, in order to apply it, the authors choose the technology to carry out the project. In their case, Hadoop was chosen, since it was more flexible regarding the structure of data. To conclude the five phases, (Tardío et al., 2015) take the following steps:

    1. Define data stages using the concept of data highway (caches) from Kimball and Ross (2013). In this phase, information requirements must be defined, as

well as time constraints from collection to analysis, data quality requirements, and query latency;

2. Collect raw data and load it into the Big Data file system (e.g., HDFS). According to the requirements, one can choose batch load (e.g. HDFS commands and MapReduce-based ETL) or streaming load. In the case of streaming load, the data can be analyzed in real-time (e.g., Spark Streaming) or stored for later processing. As the authors do not have the goal of real-time analysis, they just store the data;
3. Define a multidimensional model to add value to the previously stored data, identifying entities and relationships. The authors consider dividing the problem into facts and dimensions, and implementing models such as the star schema (Kimball & Ross, 2013). They also highlight the need to iteratively discover the multidimensional data model by exploring the raw data, linking it with the information requirements previously established. According to them, Pig or Hive can be used to query raw data according to the multidimensional schema. Tardío et al. (2015) highlight the need for a model which is flexible to further changes and enrichment (e.g., adding new data sources or using data mining);
4. Implement a BDW that supports most BI tools and query latency requirements. First, a BDW repository is chosen according to latency requirements, which can be high (e.g., Hive) or low (e.g., MPP databases like HP Vertica, or in-memory tools like Power Pivot or Qlikview). Second, the multidimensional model is implemented taking into consideration the features of the selected repository. Finally, the data is loaded into the BDW. In their case, Hive was selected in a combination with in-memory BI tools to support OLAP and dashboard applications. No details are given regarding the physical implementation of the data model and its efficiency. Moreover, it is curious that the authors propose Power Pivot or Qlikview as in-memory tools, since they are not frequently mentioned as scalable solutions in Big Data environments;

5. Develop visualizations for Big Data, using the BDW as the source. In this case, the authors used Qlikview and Excel.

Tardío et al. (2015) acknowledge that the manual effort required to apply their methodology remains high, despite all the advantages of having a systematic approach to conduct Big Data projects.

- Q. Yang and Helfert (2017) discuss the suitability of a three-layered DWing architecture on Hadoop, including: the real-time data layer built using Flume and HDFS, wherein log data is dumped without too much concern regarding its structure; the reconciled data layer, being responsible for data preparation and data storage, using Hive to deploy a star schema DW; and finally, the derived data layer, including several pre-computations similar to OLAP cubes, which are then stored in databases like HBase.

- Ali (2018) presents a real-time BDWing and Analytics framework with a demonstration case based on a communications service provider, which involved offloading the ETL from an Enterprise DW to a Big Data platform. Despite the fact that some of the frameworks' constructs and guidelines are specifically related to the context of a communications service provider, there are some guidelines and design considerations that can be useful for the design of BDWs. The framework proposed by the author allows for the ingestion of streaming and offline data from RDBMSs, files, and other transaction systems in the telecommunications context. It is divided into three main components: the real-time persistent data hub, which consists of several integrators and connectors (JDBC, files connector, Kafka, and Apache NiFi) to fetch data from multiple sources, which will eventually land in the BDW; the BDW (implemented using Spark and HDFS), a key component of the framework that manages raw data in a Hadoop data lake, mainly using JSON and compressed Optimized Row Columnar (ORC) files as the main formats; and, finally, the active data analysis platform (implemented using Apache Ignite, Spark Streaming, and Storm) is considered by the author the core component of the framework, and it is further divided into three layers that preprocess raw data, assure data modeling and visualization, respectively.

The output of the active data analysis platform is stored back into the BDW, in order to be consumed by reporting and campaign purposes, merging several insights and structured attributes to target (new) subscribers with pertinent campaigns.

- Golfarelli and Rizzi (2018) conduct a literature review to discuss more than 20 years of DWing techniques, architectures, and methodologies, already calling the reader's attention to some emerging Big Data needs and systems, such as distributed architectures, data partitioning, and data replication supported by proprietary Big Data appliances, Hadoop, Hive, Presto, among many other technologies. Regarding some methodologies related to DWs in Big Data contexts, the authors highlight some research focus being given to OLAP on NoSQL databases.
- Tria, Lefons, and Tangorra (2018) present a framework for evaluating methodologies to design BDWs, defining a set of criteria like application, agility, ontological approach, paradigm, and logical modeling. The authors also provide ways of dividing methodologies into classes (e.g., automatic, incremental, and non-relational), as well as a way to define the characteristics being addressed by the methodology (e.g., value, variety, and velocity).

3.2 Data Warehouses on NoSQL Databases

Although NoSQL databases are mainly designed to scale OLTP applications (Cattell, 2011), that did not prevent the appearance of works that propose a DW supported by NoSQL systems and data models, which are presented in this section.

- Chai, Wu, and Zhao (2013) claim that scalability and efficiency have been key issues in RDBMS-based DWs. Nowadays, the continuous data growth is seen as a bottleneck to these systems, and the authors propose a DW based on document-oriented databases, wherein the ETL process is conducted through MapReduce. The authors conclude that their approach achieves better scalability, flexibility, and efficiency than an RDBMS-based DW.

- Liu and Vitolo (2013) extend the capabilities of graph databases and develop a Graphical User Interface (GUI) to visualize graphs. They propose the concept of "graph cube" to achieve the fundamentals of a graph DW. The authors state that their work motivates for further technical advancements.

- Gröger, Schwarz, and Mitschang (2014) propose a flexible integration between data typically stored in a traditional DW and unstructured data, based on a graph structure to link these two types of data. The authors evaluate multiple scenarios regarding volume and complexity, in which the largest graph has 3,000,000 nodes, achieving less than 10s in the execution of complex queries, which, for example, might be finding the name of all employees with several links. They use 3 machines, each one running a specific storage system, since their prototype was supported by three different storage systems.

- Tria, Lefons, and Tangorra (2014) claim that BDWs differ from traditional DWs, and their data model should be based on a more flexible design. Therefore, they propose a design methodology based on the key-value model, which considers entities and relationships. Tria et al. (2014) propose a set of rules to transform data to the proposed key-value model, instead of using star or snowflake schemas found in traditional DWs. Performance was not evaluated, and the authors also envisage the use of document and column-oriented models.

- Chevalier, El Malki, Kopliku, Teste, and Tournier (2015) study multidimensional DWs on NoSQL models, in order to support OLAP, namely with column-oriented and document-oriented models. HBase and MongoDB are used in their experiment. A set of rules to map data to those models is proposed, and the authors evaluate loading and execution times to pre-compute aggregates for different levels of detail. They use a 3-node cluster and 3 datasets (1GB, 10GB, and 100GB) generated from the TPC Benchmark DS (TPC-DS), a decision support benchmark proposed by the Transaction Processing Performance Council (TPC). The loading times ranged from around 2m to 132m. According to Chevalier et al. (2015), HBase computed all the aggregates in 1,700s, while MongoDB finished in 3,210s, so HBase has a slight advantage, according to the authors, although no certain recommendations are given.

Furthermore, several directions for future work are identified, such as the identification of queries that benefit NoSQL models, and the comparison of relational and NoSQL models.

- Dehdouh, Bentayeb, Boussaid, and Kabachi (2015) propose three approaches which allow the implementation of BDWs on column-oriented databases, each one differing in the structure of fact tables and dimensions (normalized, denormalized in a single table, and denormalized in a single table using column families). The authors propose a set of rules to map data from a multidimensional model to their data structures in a column-oriented database. HBase was evaluated in a 25-node cluster, using a SQL interface called Apache Phoenix. The dataset used in the experiment consists in 6 billion tuples retrieved from the SSB benchmark. The queries consist in aggregating a measure based on different dimensions and attributes. Depending on the query, the two experiments conducted by Dehdouh et al. (2015) show execution times ranging from around 1,000s to over 2,000s for the normalized model, and around 250s to 600s for the denormalized models, with a small advantage when using column families.

3.3 Storage Technologies, Optimizations, and Benchmarking for Big Data Warehouses

The way data is stored, either physically or logically, plays a relevant role on how users interact with the BDW. Consequently, there are several works that propose optimizations to existing technologies or new database systems adequate for the typical workloads in a BDW. In this section, Table 3.1 presents several approaches, including their research contribution, main characteristics, and achieved results.

Table 3.1. Research on storage technologies, optimization, and benchmarking for BDWs.

Work	Research contribution	Main characteristics	Evaluation and results
(Thusoo, Sarma, et al., 2010)	Hive, an open source DW solution built on top of Hadoop.	Supports SQL-like queries (HiveQL) and UDFs; includes a metastore with schemas and statistics.	In 2010, at Facebook, Hive had stored 700TB of data, and it was receiving 5TB daily. The cluster scaled accordingly to the workloads, including reporting and ad hoc analysis.

Work	Research contribution	Main characteristics	Evaluation and results
(H. Wang, Qin, Zhang, Wang, & Wang, 2011)	LinearDB, which joins the efficiency of parallel databases and the scalability and fault-tolerance of MapReduce.	Modifies the traditional star/snowflake schema to achieve better scalability, and has a specific query mechanism to take advantage from it.	Cluster: 5 nodes. Dataset: 120M rows (30GB). Benchmark: SSB. Results: it was faster than PostgreSQL, ranging from around 40s to just over 100s. LinearDB also achieved adequate scalability.
(Guo, Xiong, Wang, & Lee, 2012)	Mastiff, a MapReduce-based system to achieve high loading speed and query performance on time-based data.	Uses optimized table scans and a column-based query engine.	Cluster: 20 nodes. Dataset: 200GB from the TPC Benchmark H (TPC-H) and 30GB from a network monitoring system. Benchmark: TPC-H. Results: it was able to load data and perform queries faster than Hive, HadoopDB and GridSQL.
(Qu, Rappold, & Dessloch, 2013)	Surpass join inefficiency in MapReduce-based DWs.	Frequently used dimension columns are pre-joined with fact tables.	Cluster: 6 nodes. Benchmark: TPC-H. Results: reduced the storage footprint, since data was not fully denormalized, but the performance improvements were not stable as the data volume increased.
(Alsubaiee et al., 2014)	AsterixDB, a platform suitable to use cases related to Big Data (e.g., Web DW and social media).	Has a flexible NoSQL-style data model and a specific query language; it is scalable and includes several data types; AsterixDB can query data stored internally or externally.	AsterixDB performed well against Hive, a commercial parallel DBMS, and MongoDB, running some of the tested queries in less time than the aforementioned systems.
(Bissiriou & Chaoui, 2014)	Improve the performance of HadoopDB.	A fast and space-efficient file format (RCFile) is introduced, as well as a new SQL-to-MapReduce translator and a new column-oriented database.	The authors did not benchmark their approach.

Work	Research contribution	Main characteristics	Evaluation and results
(Floratou et al., 2014)	Compare the performance of Hive and Impala as SQL-on-Hadoop systems.	3 benchmarks are used, and the comparison includes the two systems using recent file formats (ORC and Parquet).	Cluster: 21 nodes. Benchmarks: TPC-H, TPC-DS, and custom I/O tests. Results: Impala delivered a significant performance advantage over Hive (on MapReduce and on Tez) when the dataset fitted into memory, due to Impala's I/O and query efficiency. Execution times ranged from around 10s to 1,000s on more intensive queries.
(S. Hu et al., 2014)	DualTable, which aims to preserve Hive's query performance when data updates are frequent.	Combines the streaming read efficiency of HDFS and the random write capabilities of HBase.	Benchmarks: TPC-H and a workload from a real application. Results: successfully improved Hive's performance. The authors did not compare their approach with Hive's ACID capabilities, since this feature was not ready at the time.
(Huai et al., 2014)	Advance Hive's storage and runtime performance.	Update Hive's existing file formats to ORC, improving storage capacity and data access; optimize resource usage through an efficient query planner and execution engine.	Cluster: 11 nodes. Benchmarks: TPC-H and TPC-DS. Results: significant improvements in storage and query efficiency.
(Sureshrao & Ambulgekar, 2014)	Study several MapReduce-based storage structures.	Row, column, and hybrid structures are presented, as well as RCFile, Mastiff, and ORC.	Advantages and disadvantages were presented, but performance was not evaluated.
(Almeida, Bernardino, & Furtado, 2015)	Evaluate storage technologies for BDWs.	MySQL Cluster and Hive are compared.	Cluster: 1, 2, and 4 nodes. Dataset: ranging from 1GB to 24GB. Benchmark: SSB. Results: scalability issues were identified in MySQL cluster, unlike Hive. According to the authors, MySQL cluster is best suitable to OLTP.
(Arres, Kabachi, Boussaid, & Bentayeb, 2015)	Improve MapReduce performance through a new approach to allocate data blocks.	Related data blocks are collocated in a particular form to improve query performance.	Cluster: 10 nodes. Dataset: 920GB. Benchmark: TPC-H. Results: Query execution time was reduced. For the presented queries, the execution times ranged from around 7,000s to 8,000s.

Work	Research contribution	Main characteristics	Evaluation and results
(Chao, Li, Liang, Lu, & Xu, 2015)	DataMPI, an approach to improve Hive's performance.	Uses a message passing interface.	Cluster: 8 nodes. Dataset: up to 40GB. Benchmarks: Intel HiBench and TPC-H. Results: significantly improved Hive's performance (30% to 32% on average).
(Barkhordari & Niamanesh, 2017)	Atrak, a MapReduce-based DW.	Improves data locality with a unified data format.	Cluster: 50 nodes. Dataset: 100TB. Benchmark: TPC-DS. Results: Atrak presented performance improvements over systems like Hive and Spark SQL.
(Chou, Yang, Jiang, & Chang, 2018)	Evaluation of a system architecture for power meter data analysis with Hive, Impala, and Spark SQL.	Besides presenting a system architecture, most of the work is focused on comparing the performance of Hive, Impala, and Spark SQL.	Cluster: 8 nodes. Dataset: 56,000,000 to 1,120,000,000 records. Benchmark: custom-made. Results: Impala presented better results in query processing, followed by Spark and Hive, respectively. Spark also demonstrated performance benefits in ETL processing when compared to Hive.

3.4 Advancements in OLAP, Query, and Integration Mechanisms for Big Data Warehouses

Research related to analytics on BDWs has become increasingly relevant. The community is focused on aspects such as combining the benefits of RDBMSs and non-relational systems, proposing query optimizations in HiveQL, as well as how to store and process multidimensional structures (e.g., OLAP cubes) in these new systems (Cuzzocrea et al., 2011; Cuzzocrea, 2013, 2016). In the previous section, research related to storage systems for BDWs was described. Some of the approaches also propose query planners and executors to improve the performance of these storage systems. However, in this section, the focus is not on the storage layer, but on OLAP, query, and integration mechanisms to improve analytical tasks in BDWs, i.e., approaches which focus on advancing analytical and integration mechanisms for improved interactions with BDWs.

The community has been vastly contributing to the improvement of analytical tasks over BDWs, either by embedding predictive models directly into the database using SQL UDFs in parallel database architectures with high throughput (around 2M records per second) (K. K. Das, Fratkin, Gorajek, Stathatos, & Gajjar, 2011), or by proposing low-latency query engines to process Hive's data as it constantly increases with the number of users, which is the case at Facebook, supporting queries that scan 5PB of compressed data (Murthy & Goel, 2012). Therefore, reducing latency becomes critical for exploratory analysis, in cases where creating a vast set of pre-aggregation mechanisms is significantly inconvenient. Consequently, efficient query processing, real-time ETL mechanisms, and scalable OLAP on Big Data are research trends related to BDWs, as shown in Table 3.2.

Table 3.2. Research on OLAP, query, and integration mechanisms for BDWs.

Work	Research contribution	Main characteristics and highlights
(Asif, Dobbie, & Weber, 2013)	Improve real-time data integration.	Use of algorithms for efficient joins between a stream and a vast volume of data stored on disk.
(Weidner, Dees, & Sanders, 2013)	Achieve sub-second query execution times.	In-memory OLAP is used in environments with Terabytes of data. Execution times are significantly fast, usually less than 1s.
(Cuzzocrea & Moussa, 2014)	Study parallel OLAP cubes in Big Data environments.	OLAP based on relational technology is used, namely the Mondrian server.
(Cuzzocrea & Moussa, 2018; Cuzzocrea, Moussa, & Vercelli, 2018)	Support the DW maintenance process for near real-time OLAP, making use of big summary data (e.g., materialized views).	Inspired by the Lambda Architecture, the authors propose an approach for managing and refreshing big summary data in near real-time OLAP contexts. To evaluate the approach, the authors use the TPC-H benchmark and create a set of materialized views on top of the original dataset.
(Dehdouh, Bentayeb, Boussaid, & Kabachi, 2014)	Aggregation mechanism based on OLAP cubes.	The authors propose a columnar NoSQL cube using Apache Phoenix and HBase. Execution times are around 20,000s for a 1TB dataset, in a 15-node cluster.
(Ferrández et al., 2014)	Extend the traditional query mechanisms with question answering capabilities.	A question answering framework that combines external unstructured data with structured data stored in a DW.
(Lebdaoui, Orhanou, & Elhajji, 2014)	Address the integration of Big Data into the DW in shorter time periods.	The volume of data changes is divided to increase the rate of data integration and to refresh the DW more often, while preserving data integrity.
(Beheshti, Benatallah, & Motahari-Nezhad, 2015)	A framework to support scalable graph-based OLAP Analytics.	Summarization and multiple granularity levels are used to facilitate the analysis of graphs with significant volume.

Work	Research contribution	Main characteristics and highlights
(Li & Mao, 2015)	A real-time ETL framework to avoid congestion between OLAP queries and OLTP updates.	An external dynamic storage area and a replication mechanism are proposed to avoid blocking issues, reducing OLAP response times and assuring adequate real-time accuracy.
(Song et al., 2015)	A Hadoop-based OLAP system to process Big Data.	Adopts a multidimensional model based on dimensions and measures. Shows performance advantages in data loading and OLAP over other evaluated systems.
(H. Wang et al., 2015)	Improve BDWs through a new query processing framework.	Join operations are partially pushed both to a preprocessing phase and a postprocessing phase. Fact tables are rearranged so that dimensions' hierarchies are compressed to eliminate the need for typical star/snowflake joins in query processing.
(C. Xu et al., 2015)	Octopus, a computation engine to bridge the gap between data scientists and the DW.	A SQL-like query language is used to integrate both database queries and machine learning algorithms. It can be used to interact with different data sources and execute machine learning algorithms on that data; Octopus optimizes the amount of data movement, and it was able to outperform Spark 1.4 in an analytical scenario using a 9-node cluster.
(Tian, Özcan, Zou, Goncalves, & Pirahesh, 2016)	A hybrid approach to join data stored in HDFS and a traditional DW.	Study of several algorithms to join data stored in HDFS and a DW, in order to identify the most adequate hybrid warehouse architecture.

3.5 Implementations in Specific Contexts

Several business products rely on the value that can be extracted from the DW through analytical techniques, such as ad hoc queries, dashboards, reports, or data mining, for example. Therefore, among the literature, there are some approaches that present specific applications of a DW in Big Data environments, often referred as a BDW. In this section, these approaches will be presented, as well as their respective contributions to the topic of BDWing.

- Thusoo, Shao, et al. (2010) present the DW and analytics infrastructure at Facebook, which includes Scribe, Hadoop, and Hive as the fundamental components of log collection, storage, and analytics, which combined make available a DW that can handle 10TB of compressed data every day. At Facebook, Hive is used for reporting, ad hoc queries, and analysis.

- Brulé (2013) explores the use of Hadoop, NoSQL, MPP DWs, stream processing, and predictive analytics in the energy and production industry, highlighting several potential use cases for their application, in order to augment an industry which is typically based on physics-based models and simulations, as the author claims.

- S. Wang et al. (2014) aim to improve the performance of a DW about biological data, conducting an experimental evaluation to compare a key-value model in HBase with a data model in MySQL cluster and MongoDB. The authors demonstrate that the key-value model outperformed the others, and can be used to retrieve results based on relevant biological questions.

- Bondarev and Zakirov (2015) present a demonstration case about student performance, using Sqoop to import data from a relational DW to Hive, maintaining a snowflake schema and using it to create visual analyses. There are also other BDW applications in the education sector, such as the implementation case from Santoso and Yulia (2017) demonstrating the use of Hadoop as a Big Data tool for the data ingestion/staging phase to enhance an RDBMS-based system.

- Chennamsetty, Chalasani, and Riley (2015) propose a system to provide insights from historical data about patients, as the Healthcare industry can produce vast amounts of data. Hive is used to store the data, supporting further analytics like data visualizations about patients. This work highlights Hive's capabilities to support a BDW. Sebaa, Chikh, Nouicer, and Tari (2018) also provide a Hive-based implementation to improve healthcare resources distribution (optimal allocation of resources), presenting a constellation-based data model and data partitioning considerations.

- Ramos, Correia, Rodrigues, Martins, and Serra (2015) and Martins et al. (2015) propose an augmentation of the traditional DW, namely using automatic techniques to collect data from the Web, and store it in a NoSQL database (MongoDB), in order to complement the hotel's internal data stored in a traditional DW. Ramos et al. (2017) also propose a BDWing system for the hospital sector, whose main focus is related to other healthcare works previously presented in this section.

- Vardarlier and Silahtaroglu (2016) propose a system to help universities in the decision-making process, collecting data from several sources and storing it in a BDW, to further apply machine learning algorithms. Although the authors defined the proposed system as a BDW, they do not clearly discuss its characteristics, techniques, or technologies and, therefore, it is not conclusive if it is an augmentation of the traditional DW or a solution that uses SQL-on-Hadoop, for example.

3.6 Final Remarks

The research related to the BDW is mainly divided into five topics, as seen in previous sections: the characteristics and design changes for DWs in Big Data environments; DWs on NoSQL databases; storage technologies, optimizations, and benchmarking for BDWs; advancements in OLAP, query, and integration mechanisms for BDWs; and implementations in specific contexts. Figure 3.1 presents the distribution of the works related to BDWing discussed in this document, grouped by the main topic. The results indicate that research regarding the characteristics and design of BDWs is more predominant, discussing characteristics, design changes, guidelines, logical architectures,

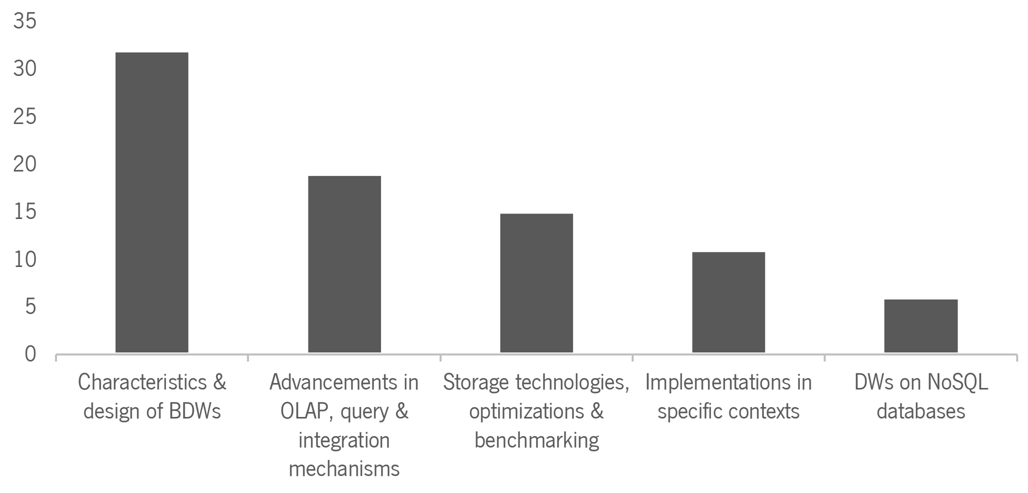


Figure 3.1. Number of works related to research on BDW, grouped by the main topic.

techniques, and technologies. Works related to analytics and storage for BDWs are the second and third more predominant topics, respectively, proposing and evaluating several approaches to improve BDWs and their analytical capabilities. Moreover, some works also present implementations in specific contexts, while others propose DWs supported by NoSQL databases.

Furthermore, according to the literature review, several characteristics can define a BDW:

- Parallel/distributed storage and processing of large amounts of data;
- Scalability (accommodate more data, users, and analyses);
- Elasticity to provide a more efficient way of scaling-out and scaling-in depending on the organizational needs;
- Flexible storage, including semi-structured and unstructured data;
- Real-time capabilities (stream processing, low-latency, and high-frequency updates);
- High performance with near real-time response;
- Mixed and complex analytics (e.g., ad hoc or exploratory analysis, data mining, text mining, statistics, machine learning, reporting, visualization, advanced simulations, and materialized views);
- Interoperability in a federation of multiple technologies;
- Fault-tolerance, mainly achieved through data partitioning and replication;
- And the use of commodity hardware to reduce the costs of implementation, maintenance, and scalability.

Moreover, Hadoop and NoSQL databases are mentioned either as a replacement of the traditional DW or as a way to augment its capabilities (e.g., ETL, data staging, and preprocessing of unstructured data), thus forming a federation of different technologies that enable the aforementioned characteristics. Figure 3.2 presents a conceptual model of the BDW, which illustrates these characteristics and strategies discussed above.

Designing a BDW should focus both on the physical layer (technological infrastructure) and on the logical layer (data models, data flows, and interoperability between components). Augmenting the

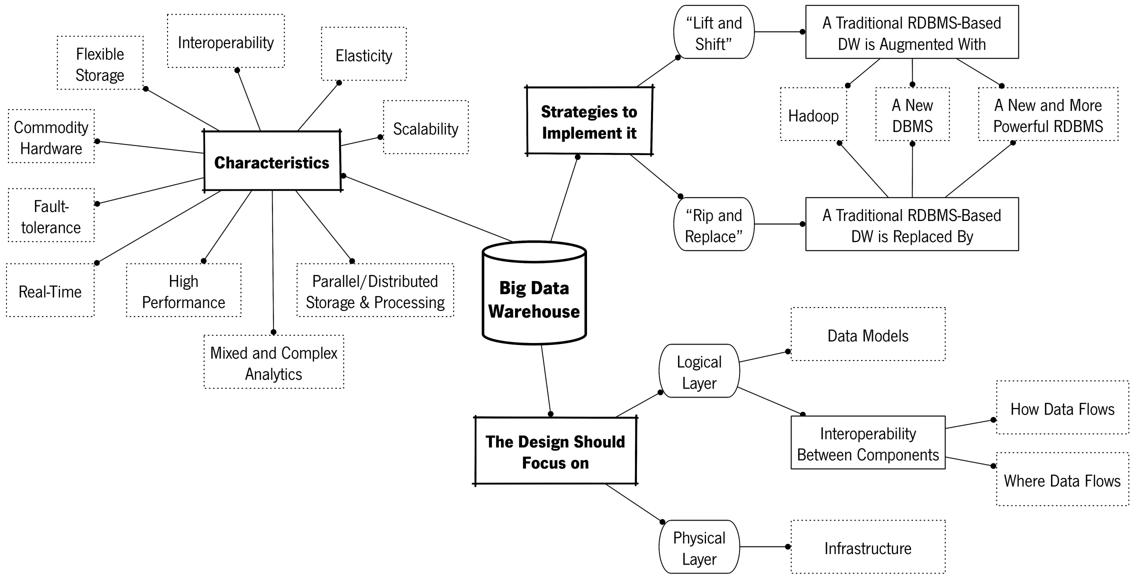


Figure 3.2. A conceptual model of the BDW.

capabilities of traditional DWs with new technology is a valid approach and, arguably, currently preferred among organizations, strategy that is known as "lift and shift". However, "rip and replace" strategies will become more common, wherein traditional DWs are fully replaced due to their limitations in Big Data environments (Russom, 2014, 2016). The "lift and shift" strategy creates a federation of several technologies and may represent a change of perspective from a data-driven view of the DW to a use case driven view (Clegg, 2015). Therefore, data modeling was previously the main concern, being now replaced by finding the right technology to meet demands, leading to the risk of uncoordinated data silos.

Current research, although of significant value, only contributes to specific characteristics of a BDW, advancing some existent technology, proposing a new one, or developing a specific implementation for a particular use case. The phenomenon among research on Big Data is noticeable: most are concerned with "selling" their technique or technology. There is a lack of prescriptive research on BDW, since there is no integrated approach to design BDWs, as formerly existed in traditional DWs, like the well-known approaches from Kimball or Inmon. This is mainly the result of shifting from a data-driven view of the DW to a use case driven view, and also due to the young age of Big Data as a research topic. However, as Clegg (2015) claims, it would be a mistake to discard decades of architectural best practices based on the assumption that storage for Big Data is not relational nor driven by data modeling principles or guidelines.

The fact that there is a significant number of works related to SQL-on-Hadoop proves that the data structures known for many years are more relevant than ever, although modified and optimized for Big Data contexts. Of course, there is unstructured data that does not adequately fit into these data structures, but there are also techniques to extract value from that data, and then use it to fuel a BDW (e.g., data mining, text mining, and machine learning). The problem identified in this literature review is that there is a significant gap between “this is what a BDW should be” and “this is how one designs and implements it”, which then leads to a use case driven approach, primarily concerned with choosing the right technology to meet demands. As approaches are use case driven, the knowledge and guidelines that can be retrieved from one implementation to the others are only possible because the circumstances are the same, thus not creating gradual and iterative knowledge, crucial for fundamental advancements in the area.

Among the works discussed in this chapter, there are already some best practices and general guidelines of major relevance, but they do not focus on both the physical layer (technological infrastructure) and the logical layer (data models, data flows, and interoperability between components) to implement the characteristics of a BDW, with adequate and detailed demonstration, discussion, and evaluation. This is of major relevance for the scientific and technical community related to BDWing, since it consequently leads to a contribution in which models (representation of data structures and components), methods (structured practices) and instantiations (prototypes and implemented systems) are tightly coupled. Such approach can lead to a prescriptive contribution to design and implement BDWs according to their characteristics of parallel/distributed storage and processing, scalability, elasticity, real-time, high performance, mixed and complex analytics, flexible storage, interoperability, fault-tolerance, and commodity hardware.

This page was intentionally left blank

Chapter 4. An Approach for the Design and Implementation of Big Data Warehousing Systems

The approach proposed in this work is a prescriptive contribution that researchers and practitioners can either use for building BDWs or for considering as background knowledge for future research. It significantly extends the current scarce and scattered contributions regarding BDWing, as it includes prescriptive models and methods that can be used as a guide for designing and implementing these complex analytical systems. The approach is based on the “rip and replace” strategy (Russom, 2016), discarding traditional RDBMS-based DWs and replacing them with state-of-the-art Big Data techniques and technologies. It is an approach that aims to address the characteristics of a BDW (Figure 3.2), focusing on both the logical and physical layers. This chapter describes the proposed approach, presenting its prescriptive models and methods, namely: a model of logical components and data flows; a method for data CPE processes; a model for the technological infrastructure; and a method for data modeling focusing on data storage and analytics.

4.1 Model of Logical Components and Data Flows

The logical components included in the proposed approach (Figure 4.1) are defined according to the components present in the NBDRA (NBD-PWG, 2015), since it aims to comply with current standards and trends in the Big Data community. Obviously, the same presents some significant modifications and also extends the NBDRA with new components, since the latter is a general architecture for Big Data solutions and, therefore, not specifically designed towards BDWs. The approach here proposed also takes into consideration relevant guidelines provided by previous published works, such as the Big Data Processing Flow proposed by Krishnan (2013) and the Data Highway Concept proposed by Kimball and Ross (2013). Furthermore, the approach often encourages compliance with three of the main principles of the Lambda Architecture (Marz & Warren, 2015): first, one should store data at the highest level of detail (e.g., raw data in the distributed file system component), since it may serve future analytical purposes not previously planned; second, whenever possible, one should model data structures to store a set of immutable events, avoiding updates to existing data, in order to simplify the BDWing system; finally, data at different speeds certainly has different requirements and, therefore, different logical components for batch and streaming data must be taken into consideration. The model of logical components and data flows (Figure 4.1) is divided into six main components: data provider; data consumer; Big Data application provider; Big Data framework provider; system orchestrator; and security, privacy, and management.

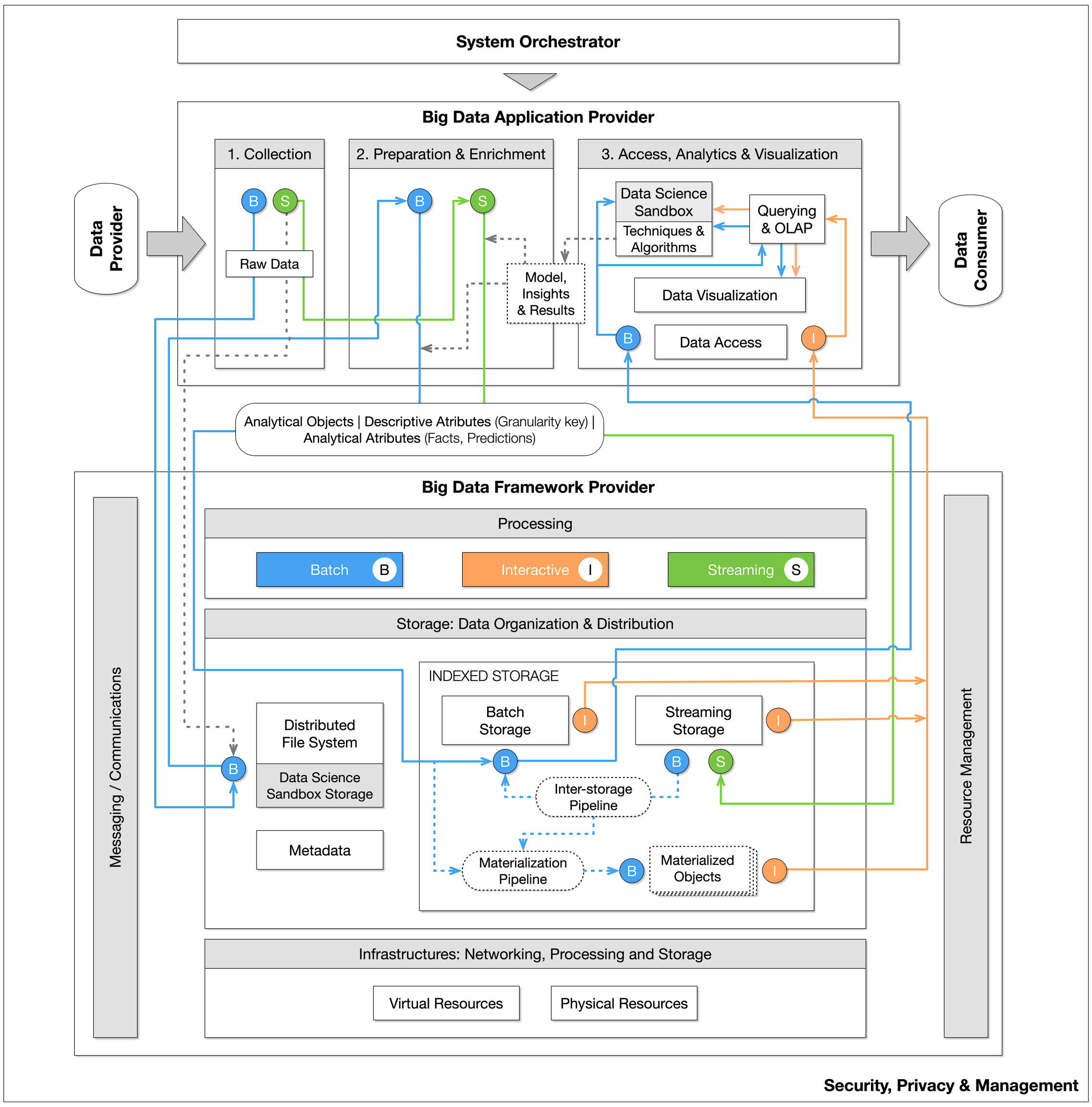

System Orchestrator
Big Data Application Provider
1. Collection
2. Preparation & Enrichment
3. Access, Analytics & Visualization
Data Provider
Data Consumer
Raw Data
Data Science Sandbox
Techniques & Algorithms
Querying & OLAP
Model, Insights & Results
Data Visualization
Data Access
B
S
I
Analytical Objects | Descriptive Atributes (Granularity key) | Analytical Atributes (Facts, Predictions)
Big Data Framework Provider
Processing
Batch
Interactive
Streaming
Storage: Data Organization & Distribution
INDEXED STORAGE
Batch Storage
Streaming Storage
Distributed File System
Data Science Sandbox Storage
Metadata
Inter-storage Pipeline
Materialization Pipeline
Materialized Objects
Infrastructures: Networking, Processing and Storage
Virtual Resources
Physical Resources
Messaging / Communications
Resource Management
Security, Privacy & Management

Figure 4.1. Model of logical components and data flows.
Dashed components are seen as optional depending on the implementation goals.

4.1.1 Data Provider and Data Consumer

The data provider component represents each actor introducing new data into the BDW (e.g., person, sensor, computer, smartphone, and Web stream). Therefore, this component represents several data sources, external or internal, online or offline, collected automatically or manually. Among the responsibilities of the data provider, the following can be highlighted: assure adequate data privacy and security; enforce access rights; make data available through suitable interfaces; and provide adequate metadata. In contrast, a data consumer represents an end-user or an external system that can perform the following actions: search and download data; analyze data (e.g., execute ad hoc queries, and train/test data science models); construct or consume reports and other data visualization mechanisms (e.g., dashboards); and include data in business processes. To access the data available in the implemented BDWing system and protected by the security, privacy, and management component (e.g., authentication and authorization mechanisms), the data consumer is able to use the interfaces made available by the Big Data application provider through demand-based interaction, where the data consumer initiates the interaction and waits for a response (NBD-PWG, 2015).

4.1.2 Big Data Application Provider

The Big Data application provider component is responsible for assuring three relevant stages in the data that flows throughout the different BDW components:

1. Collection - in this stage, the data is collected from data providers, arriving at the BDWing system at the rawest state possible. The data can arrive at the system through two different velocities, batch or streaming. Data arriving in batches is immediately stored in a distributed file system, since one of the main challenges in Big Data is variety (different structures, types, and sources), and this file system is a component that allows the storage of any variety and volume of data. This data will then be processed in the next stage. In contrast, if the data is arriving in a streaming fashion, it does not need to be stored yet, and it flows to the preparation and enrichment stage. However, an alternative route in the streaming flow can

exist, storing streaming raw data in the distributed file system, as Figure 4.1 shows, being this data available for further tasks, such as disaster recovery or training of data science models on unmodified streaming data;

2. Preparation and enrichment - the batch data previously stored is extracted from the distributed file system with the goal of being prepared and enriched to provide analytical value. The same happens with the streaming data, although it arrives at this stage directly from the collection stage, as previously explained. The preparation and enrichment of data can include all sorts of cleansing, integration, transformation, and aggregation processes. New attributes can also be created and derived from the raw data, without any limitation, as well as the extraction of hidden patterns in unstructured data (e.g., image, video, and text). These processes can not only take as input the new data arriving at the system, but also read the data already stored in the BDW, establishing comparisons and trends, for example. Besides finding patterns in unstructured data using data science techniques (e.g., text mining), these processes can also include predictions from previously trained data science models based on problems such as classification, regression, clustering, and time series forecasting. It must be remembered that the goal of this stage is to prepare and enrich data to serve the current business goals and expectations, whether they are based on facts or on predictions made by previously trained data science models (flow marked with a dashed circle in Figure 4.2). The way to implement these processes differs according to the velocity of the data (batch or streaming), since different velocities typically require different paradigms and technologies, but the essential steps are similar. In the proposed approach, batch and streaming data follow different routes, but are prepared and enriched with the same goals in mind: fuel the analytical objects to which the data belongs, structuring data according to their granularity key, descriptive attributes, and analytical attributes, whether they are facts or predictions extracted from the raw data being collected. The method for data modeling detailed in section 4.3 explains these concepts, detailing how data should be modelled in the BDW. After this stage, the data is stored in its corresponding indexed storage component: batch data is stored in the batch storage; and streaming data is stored in the streaming

storage. Figure 4.2 illustrates the proposed method for CPE processes, summarizing the steps described above;

3. Access, analytics, and visualization - this is the third and final stage of the data in a BDWing system. In this stage, the data consumers have access to the data in two fundamental ways: batch and interactive access. A batch access can be typically used for complex visualization

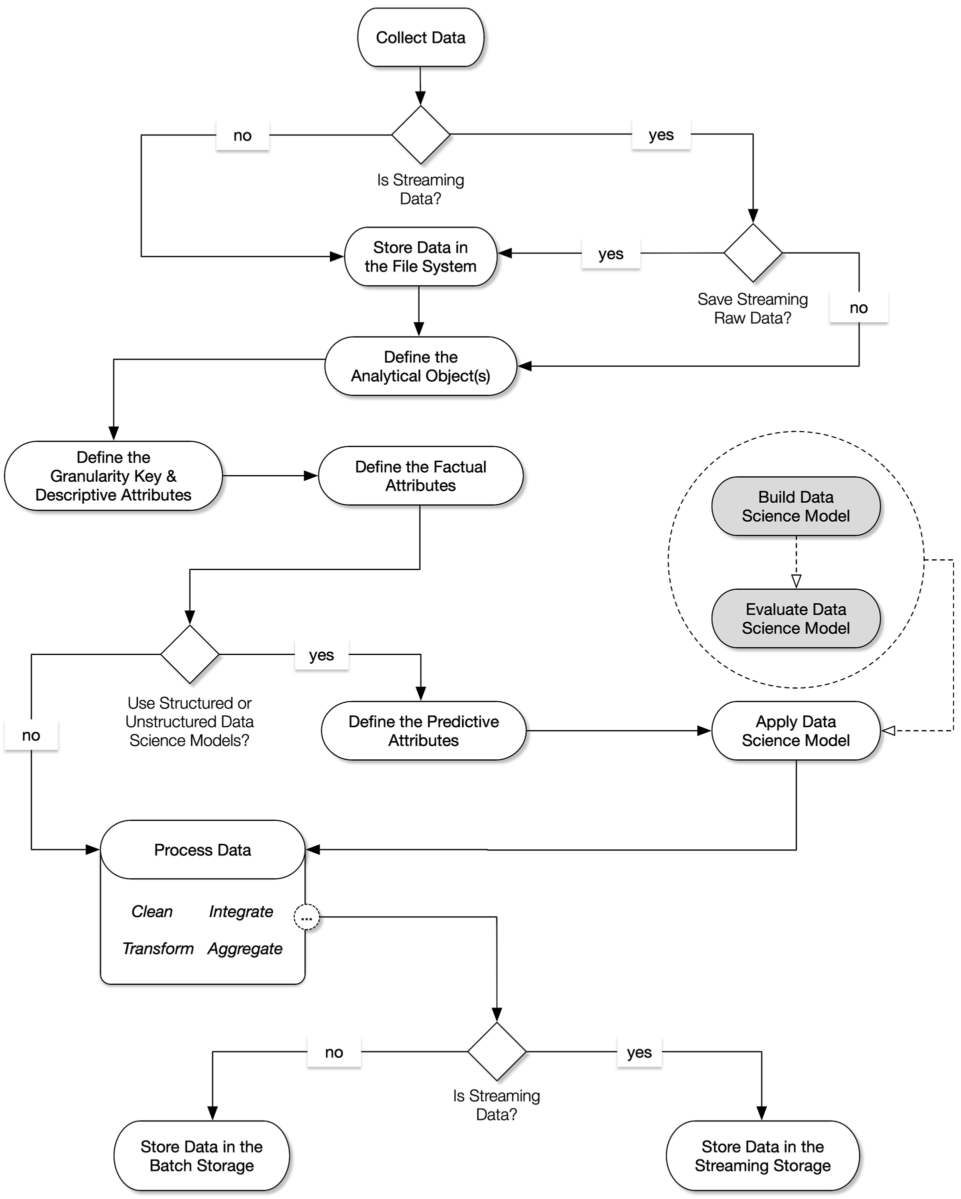

Collect Data
no
yes
Is Streaming Data?
Store Data in the File System
yes
Save Streaming Raw Data?
no
Define the Analytical Object(s)
Define the Granularity Key & Descriptive Attributes
Define the Factual Attributes
Build Data Science Model
Evaluate Data Science Model
yes
Use Structured or Unstructured Data Science Models?
no
Define the Predictive Attributes
Apply Data Science Model
Process Data
Clean
Integrate
Transform
Aggregate
...
no
yes
Is Streaming Data?
Store Data in the Batch Storage
Store Data in the Streaming Storage

Figure 4.2. Method for CPE processes.

tasks (e.g., deep and complex reporting) or for training and testing data science models, which typically require intensive computation. The data science sandbox (see Figure 4.1) is a crucial component of the BDW, where data scientists can explore the data stored in the distributed file system and indexed storage components, create models, extract useful insights, and use the results to improve or create new analytical objects. Therefore, preparation and enrichment processes can also start with batch jobs originated from the data science sandbox, without the need for a collection stage, basically meaning that the data science sandbox is just considered as another data provider in this context. These are tasks that do not necessarily require an interactive response time and, therefore, can be seen as batch-oriented tasks. In contrast, tasks such as ad hoc querying, OLAP, and exploratory data visualization require an interactive behavior to keep the data consumer engaged in the current analysis and exploration of data. The tasks enumerated previously are relatively similar to the ones performed in a traditional DW (e.g., reporting, data visualization, and exporting data to run data mining algorithms). Obviously, there are some particularities to take into consideration in Big Data environments (volume, variety, and velocity), but there is no need to propose a specific method to perform these tasks, since Figure 4.1 is already self-explanatory.

4.1.3 Big Data Framework Provider

This logical component includes the several subcomponents related to the resources that are necessary to provide an adequate distributed storage and processing platform for the BDW, as well as an adequate communication with its external actors (e.g., data providers and data consumers). Therefore, this component is mainly related to infrastructural concepts, such as messaging/communications, resource management, infrastructures (physical or virtual), data processing paradigms (batch, interactive, and streaming), and data organization and distribution paradigms (distributed file system and indexed storage).

4.1.3.1 Messaging/Communications, Resource Management, and Infrastructures

The messaging/communications component represents the need to assure reliable queuing and data transmission between nodes in a cluster that scales horizontally (NBD-PWG, 2015). It must be remembered that scalability is one of the main characteristics of BDWs. Messaging/communications techniques must also assure adequate fault-tolerance when nodes fail. In the design and implementation of BDWs, this component is relatively transparent to the stakeholders involved in the project, since it is directly related to the technologies chosen to implement the several logical components of the BDWs (section 4.2). Each technology may implement different messaging/communications techniques, and, depending on the application context, one may prefer a specific technique that better meets the requirements of the project. This is something that stakeholders should be aware, but it is often transparent to the team installing and configuring these technologies.

Regarding the resource management component, it represents a relevant concern included in the technologies that assure distributed storage and processing, which are an adequate way of achieving scalability in a BDW. These technologies must efficiently manage the resources available in the cluster, namely CPU and memory. The inadequate management of these resources may severely impact the performance of the BDW. One of the main concepts regarding this component is data locality, since data is too big to be moved from the storage nodes to the processing nodes through the network (NBD-PWG, 2015). Therefore, the processing needs to be closer to the storage (C. L. P. Chen & Zhang, 2014), typically co-locating the processing and storage nodes in the cluster. Similarly to the messaging/communications component, the resource management component is directly related to the technology chosen to implement a specific logical component. Each technology may use different techniques to manage resources.

Finally, the logical component related to the infrastructures highlights all the physical and/or virtual elements necessary to run the tasks assigned to each component of the BDW, including the network to transfer data, the CPU and memory to provide adequate data processing, and the storage to provide data persistence. Physical resources represent the hardware used across the different nodes in a horizontally scalable cluster. In contrast, virtual resources are frequently used to achieve an elastic and flexible allocation of physical resources, typically referred to as IaaS. Although Big Data technologies can be deployed on virtualized environments, the majority of them are designed to run directly on physical commodity resources (NBD-PWG, 2015), providing efficient I/O by distributing multiple CPUs, memory units and disks across a cluster of commodity machines based on a shared-nothing architecture.

4.1.3.2 Processing

In a BDW built using the proposed approach, there are three types of processing according to the different levels of latency, namely batch, interactive, and stream processing. Generally, the boundaries between these three types of processing are not clear. However, in this work, they are defined as follows:

- Batch processing - this type of processing involves latencies ranging between several minutes and hours. Examples of batch processing may include: the periodic CPE of vast amounts of historical data from data providers; the processing of deep and complex reports or ad hoc queries; and the training of data science models, which involves complex and processing-intensive data mining and text mining algorithms. These tasks are ideal for running in the background without the need for user intervention;

- Interactive processing - this type of processing is used to provide query execution times ranging from milliseconds to a few tens of seconds, depending on the infrastructure and data volume. There are a few data organization, distribution, and modeling strategies used in this work that allow for this level of latency even with ever-increasing amounts of data. Such strategies include: data denormalization; data partitioning; and inter-storage and materialization pipelines (see subsection 4.1.3.3 and section 4.3);

- Stream processing - in this work, this type of processing only concerns the latency in data CPE processes, meaning that data consumers do not have direct access to the data being streamed. Instead, streaming data is stored in the streaming storage component and it is immediately available to the data consumers through interactive processing supported by this data storage component. Regarding the levels of latency, streaming data should arrive at the streaming storage in milliseconds or a few seconds, in order to be immediately available to data consumers. However, when preparing and enriching streaming data, sometimes, it is useful to create micro batch jobs to perform specific operations, such as small aggregations, window operations, application of data science models, and merging streaming data with batch data to establish trends. Micro batches can be seen as a significantly small batch of data records, instead of handling them individually. The size of a micro batch job is often customizable in several streaming technologies. When this type of operations is needed, the data arrives in a streaming fashion to the BDWing system, but may only be available after a few tens of seconds or even minutes, depending on the size of the micro batch. Micro batches can also help improving the throughput of the data flow, only requesting an insert operation on the streaming storage component when a micro batch is completed, instead of creating a request for each data record.

4.1.3.3 Storage: Data Organization and Distribution

Data organization and distribution is a crucial aspect in the proposed approach. It is designed with the goal of providing a flexible and scalable data storage solution that is aware of data volume, variety, and velocity, without necessarily discarding a data modeling method. In this context, the storage design philosophy presented here is based on two relevant components: the distributed file system, which is an unstructured data storage solution, wherein data does not necessarily need to have a specific schema nor does it need to be modelled in a specific way; and the indexed storage component, wherein the data must comply with specific structures, although they are based on a flexible modeling technique suitable for BDWs, as detailed in section 4.3. These two components are also related to the logical component that provides all the metadata for the data stored in the file system and in the indexed storage components (e.g., file locations, data types, descriptions, and relevant timestamps).

4.1.3.3.1 Distributed File System

Making the analogy to the traditional DWs, the distributed file system can be seen as an empowered staging area, wherein raw data can not only be stored for later preparation and enrichment, but also for training data science models based on structured or unstructured data, since a file system adequately supports schema-less data sources. Therefore, data scientists can use this file system as a sandbox to explore the data and discover hidden patterns, providing useful insights to support the decision-making process. Taking a closer look at Figure 4.1, it can be observed that data scientists can also use this component to store the results of queries submitted to the indexed storage component, and use these results to create or improve data science models based on several techniques and algorithms. These models and insights can then be included in further data CPE processes, combining them with data arriving at the BDW and storing the result in the analytical objects (section 4.3) stored in the indexed storage component.

This approach provides adequate flexibility to freely explore the data in its raw state, to combine it with previously stored data, if applicable, and to make sure that the sandbox findings flow to the indexed storage component, which assures that the analytical requests from data consumers are fulfilled. Take as an example a company that sells jewelry in several countries. This company collects data from its point of sale systems using batch processing, as well as unstructured text from social media using stream processing. Data scientists use the unstructured data stored in the distributed file system to train a text mining model for extracting sentiments regarding the different types of jewelry in several countries. In the indexed storage, the company stores an analytical object for the sales data and an analytical object for the sentiments expressed regarding the different types of jewelry in each country. Meanwhile, after days of querying the data stored in the indexed storage component and saving the findings in the distributed file system, a certain data scientist can start to classify a sale as “expected” or “unexpected”, which results from the comparison between the jewelry being sold and the sentiments expressed for that product in the country in which the sale is being made. This is an example of the usefulness and flexibility of the distributed file system in a BDWing system built using the proposed approach, in order to complement or create analytical objects stored in the indexed storage component.

4.1.3.3.2 Indexed Storage

Contrasting with the distributed file system, the indexed storage is a component oriented towards data modeling, i.e., data needs to be structured according to a specific data model. However, in this work, the data model based on analytical objects offers significant flexibility, while maintaining a structured schema suitable for querying and OLAP. This modeling method is further discussed in section 4.3. In this subsection, the focus is on the logical components responsible for storing these analytical objects. The indexed storage component is divided into two main storage types, namely the batch storage and the streaming storage. Nevertheless, the data modeling approach is the same for the two types of storage, storing all the data in analytical objects and their descriptive and analytical attributes.

The batch storage component represents a repository of analytical objects that are refreshed less frequently, since the data only arrives in a batch-oriented fashion and, therefore, the time interval between updates is usually several minutes, hours, days, weeks, or months, for example. In contrast, the streaming storage component stores analytical objects that are refreshed frequently, since the data arrives through streaming mechanisms and, therefore, updates are usually happening with time intervals of milliseconds, seconds, or a few minutes (for large micro batches). A relevant component related to these two storage types is the inter-storage pipeline, which is responsible for transferring data between the streaming storage and the batch storage. Consequently, the same analytical object may exist simultaneously in these two storage components. This may happen if the technology being used to support the streaming storage has fast random access to data, but it is not optimized for fast sequential access. In contrast, if the technology is the same for both storage types, so either balanced or more optimized for fast sequential access and not for fast random access, frequently, the inter-storage pipeline may only need to execute background jobs to distribute data in a more efficient way, such as, for example, merging many small files originated by the streaming process into one larger file, since small files can become a problem in Hadoop (Mackey, Sehrish, & Wang, 2009). It must be remembered that, internally, indexed storage systems also persist data as files. The inter-storage pipeline is optional, depending on the infrastructure being deployed to support the BDW, since technology is constantly evolving, and there is increasing interest in exploring storage systems that adequately support both fast sequential access and fast random access in Big Data environments, as discussed in section 4.2.

Other optional subcomponents included in the indexed storage component are the materialization pipeline and the materialized objects. A materialized object is an object that stores the results of a query executed over one or more analytical objects. The materialization pipeline is the logical component that assures this materialization process. Materialization can be significantly helpful for improving execution times in contexts where the data consumer consistently submits similar queries to the BDWing system. Moreover, materialization also helps storing the results of deep and complex requests like long-running reports, which otherwise will take a significant amount of time to complete. Materialization may typically represent a trade-off between data timeliness and response times, but there are several contexts where the data consumers do not need the most recent data available in the BDW. Nevertheless, materialized objects can be refreshed when a new batch of data arrives at the system or when the inter-storage pipeline runs a background job (Figure 4.1). Consequently, the materialization pipeline can either re-process the whole materialized object, or perform an incremental change by reading it and complementing it with new data.

To conclude this subsection regarding the indexed storage component, it is relevant to highlight that the analytical objects stored either in the batch storage or in the streaming storage can be organized and distributed using two relevant concepts: partitioning and bucketing/clustering (Thusoo, Sarma, et al., 2010). These two concepts can largely influence query performance in certain contexts. When relying on an indexed storage that makes use of partitioning, all the data of an analytical object is stored as many small pieces of data inside the storage system, dividing a large dataset into many small and more manageable parts that can be accessed individually, without the need to search the entire dataset. An example of partitioning is storing an analytical object like sales transactions using a separate storage location for each year, month and/or day. If one needs to analyze the sales of the last month, the indexed storage system only needs to scan the partition corresponding to the respective month. Partitioning can be significantly helpful when data consumers have a well standardized access to data, such as querying the data stored in analytical objects for a certain period of time (e.g., year, month, and day) or place (e.g., country, region, and city). Consequently, partitioning improves the performance of queries when the typical filtering attributes are used to partition the dataset. Partitioning can also be significantly useful when data is loaded in periodic batches or in batches corresponding to certain places, since a partition can be assigned for each batch.

Bucketing/clustering represents a technique to make sure that a range of records are stored in the same group/bucket or sorted in a certain way, according to the attribute(s) used for bucketing/clustering. The way it is physically implemented differs according to the technology, i.e., some storage technologies may group a range of values in the same file, while others can order the values and make sure they are stored in a sorted fashion. Following the example of sales transactions, if a certain organization has several sales employees, using a bucketing/clustering technique with the identification of the employee, the indexed storage can store the transactions of the same employee in the same bucket, or make sure the transactions are sorted according to the identification of the employee. Partitioning and bucketing/clustering can be used together, and query performance can be significantly impacted when adequate strategies are taken into consideration (E. Costa, Costa, & Santos, 2018).

4.1.4 System Orchestrator and Security, Privacy, and Management

The system orchestrator is seen as an overarching role, including several actors (humans and/or software) that manage and orchestrate the daily operations of the BDWing system. The system orchestrator aims to configure and manage other components of the architecture, in order to sustain the workloads that are being constantly executed. Its tasks include: assign/provision the Big Data Framework Provider (subsection 4.1.3) to physical or virtual nodes; provide GUIs for the specification and management of workloads; and monitor the system and its workloads through the security, privacy, and management component, taking into account the requirements and constraints, such as business requirements, policies, architectural design choices, and resources, for example (NBD-PWG, 2015).

In the proposed approach, the security, privacy, and management component represents an overarching concern that is related to all other components in the BDWing system. Managing such complex system typically involves several considerations at a massive scale, while the system performs multiple tasks in a production cluster with several nodes. Among the tasks concerning this component, the following can be highlighted (NBD-PWG, 2015):

- Policy, metadata, and access management (authentication and authorization);
- Provide adequate encryption capabilities at networking or storage levels (if needed);
- Provide adequate auditing capabilities;
- Disaster recovery in case of data loss;
- Provide adequate monitoring mechanisms for the resources and performance of the system;
- Make available adequate platforms for resource allocation and provisioning, as automated as possible;
- Configure and manage the installed software.

4.2 Model of Technological Infrastructure

While the model of logical components and data flows represents an artifact for the design of BDWing systems, the model of the technological infrastructure represents an artifact for their implementation, focusing on the technologies that can instantiate each logical component, as well as focusing on the hardware that can be used to deploy the BDWing system. In this section, Figure 4.3 presents the model of the technological infrastructure, including several state-of-the-art technologies for each logical component of the BDW presented in Figure 4.1. Therefore, a direct association can be made between the two figures, aiming to provide a coherent view and simplicity in the design and implementation phases of BDWing initiatives. The colors (blue, orange, and green) are used according to the types of processing depicted in Figure 4.1 (batch, interactive, and streaming, respectively). The technologies presented in Figure 4.3 must be seen as suggestions made by this work, which is based on several Hadoop-related projects, and not as a preference over any other technology that researchers and practitioners may find suitable for implementation. This is the reason why the model illustrates that there is space for other possibilities. For each logical component, several suitable technologies are presented, which must be seen as alternatives or complementary ones, and not as mandatory in all implementations. Finally, Figure 4.3 also presents how these technologies are

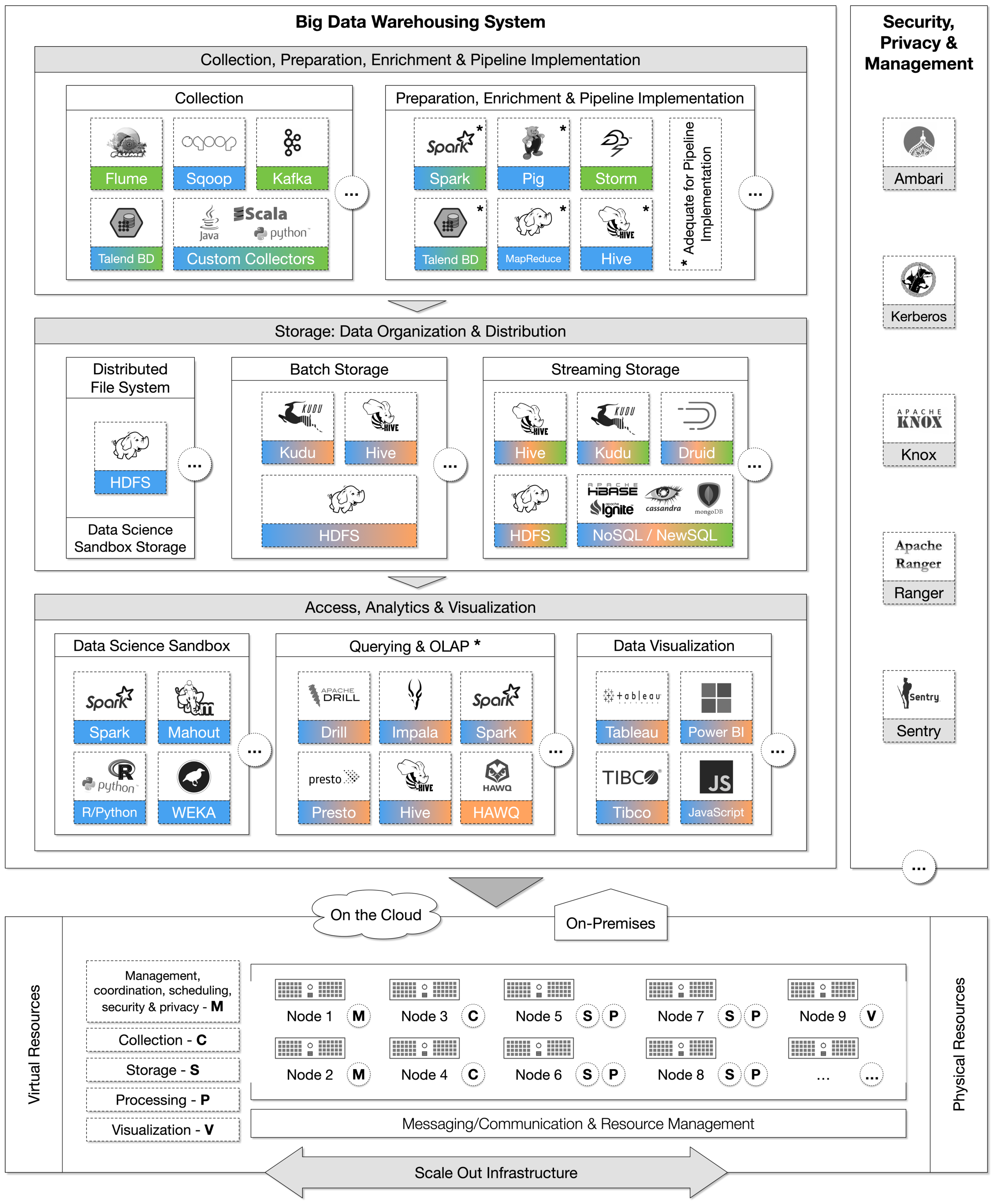


Figure 4.3. Model of the technological infrastructure.

supported by a scale-out infrastructure, deployed on-premises or on the cloud, either using physical or virtual resources.

Starting with data collection, Flume and Kafka are suitable technologies that can be used to collect data in a streaming fashion. In contrast, Sqoop can be used to move batches of data from relational databases into HDFS. There are also ETL tools oriented towards Big Data contexts (e.g., Talend Big Data), which include components for both batch and streaming data collection. However, frequently, these tools, in their open source versions at least, just provide an integrated GUI for submitting tasks to systems such as Flume, Kafka, and Sqoop. Therefore, technically, the technologies mentioned above still have to be deployed on the infrastructure. Furthermore, for specific data collection scenarios, one may need to implement custom collectors developed using well-known programming languages, such as Java, Python, and Scala, either for batch or streaming scenarios.

For data preparation and enrichment using batch processing, Hadoop-related projects like Pig, Hive, and Spark are adequate technologies. Native MapReduce code, although complex, can also be used for this purpose, as well as Talend Big Data. Regarding preparation and enrichment via streaming, Storm, Spark Streaming, and Talend Big Data can be used. Nevertheless, as mentioned above, Talend, in its open source version, typically makes use of the other components to assure adequate distributed processing, since its native components may not be scalable. Since these tools include a vast set of storage connectors and data processing components, some of them are also adequate to support the implementation of the inter-storage and materialization pipelines (namely the technologies marked with an asterisk in Figure 4.3). The technology to choose for this purpose will obviously depend on the choice of the storage technologies.

Storage technologies are one of the crucial aspects of the BDWing system, and maybe one of the most difficult to understand. In regard to the Big Data technologies for the distributed file system, HDFS is a not-so-complicated choice, since it provides a way of storing all kinds of data, structured or not. The dilemma relies on the indexed storage component, i.e., on the batch storage and on the streaming storage. One may take one of the following approaches: the first being based on infrastructural simplicity, which releases management burden for system orchestrators; the second being a hybrid approach, which can assure more efficient refresh processes, but can also impose more challenges concerning the management of the infrastructure.

Assuming one aims for infrastructural simplicity, the same storage technology is reused as many times as possible. Therefore, since HDFS is used as the distributed file system, it can also be used both for historical and streaming storage. Hive uses HDFS to store the data, so, technically, using Hive tables to store the analytical objects is as complex as using raw HDFS files. Consequently, using HDFS with file formats oriented towards analytics like Parquet and ORC (Huai et al., 2014), or using Hive tables stored in these formats, represent the approach with maximum infrastructural simplicity. However, this approach may sacrifice data refresh rates, since streaming mechanisms will have to group data records in larger micro batches, in order to avoid creating multiple small files, as these can cause concerns in Hadoop (e.g., larger metadata footprint in RAM and unsatisfactory NameNode performance) (Mackey et al., 2009), as briefly mentioned in section 4.1. This phenomenon occurs because HDFS and Hive are currently oriented towards fast sequential access and not towards fast random access (more details related to streaming scenarios are provided in section 7.3). The problem is that increasing the micro batch size also increases the interval between data collection and its availability for querying in the BDWing system. Despite this, there are many streaming contexts where it is not an issue if data is only available a few minutes later after its collection.

Taking this into consideration, to achieve shorter time intervals between the collection of data and its availability for querying, one can use storage systems oriented towards fast random access, such as NoSQL databases. Another advantage of these systems is the capability to perform random reads or updates on data, which can be useful for certain BDW applications. For instance, the use of these systems enables efficient update operations on records, in cases where it is not feasible to model data as a set of immutable events. However, since these databases are mainly used for OLTP-based workloads (Cattell, 2011), they typically do not perform as well as the fast sequential access systems for OLAP-based workloads. Consequently, choosing NoSQL databases solves the small files problem in Hadoop, but may also bring more infrastructural complexity and slower query execution times for OLAP-based workloads (results provided in section 7.3).

Among NoSQL databases, one can also highlight the relevance and possible use of in-memory NoSQL databases like Redis (Redis, 2018), or even NewSQL databases like Apache Ignite (Apache Ignite, 2018), if the chosen querying and OLAP system supports these technologies. In fact, some of these technologies may provide faster query execution times, as they sometimes have more optimized in-memory architectures. Again, the modularity of the approach allows for these flexible implementation choices without changing any significant architectural construct or data modeling guideline. Another adequate technology for streaming scenarios is Druid (F. Yang et al., 2014), a columnar store that can be used to support interactive and concurrency-heavy applications focusing on slicing-and-dicing, drilling down, and aggregating event data. Druid achieves this by aggregating and indexing time-based data as soon as it arrives to the system, providing sub-second queries over vast amounts of streaming data (Correia, Santos, Costa, & Andrade, 2018). Another adequate use case for Druid is the storage of materialized objects due to its on-the-fly aggregation mechanisms. Although Druid can be used for the batch storage component as well (Correia et al., 2018), this work highlights its use for streaming and materialization scenarios containing aggregated data indexed by temporal attributes, as this can be recognized as its main design focus. As many other Big Data technologies, Druid has its limitations (e.g., lack of support for random access operations), and practitioners should perform a preliminary analysis when choosing storage technologies, as the ecosystem is rapidly evolving.

Furthermore, there are other technologies aiming to provide a middle ground between fast sequential access and fast random access, which is the example of Kudu, being able to support both scenarios without the need for different storage systems (Lipcon et al., 2015). Kudu can be co-located with other components of the Hadoop ecosystem and, therefore, can be used together with HDFS. Using the same storage system for both batch data and streaming data can also reduce infrastructural complexity, although HDFS should continue to be used as the distributed file system. Furthermore, as previously stated, technology is evolving rapidly, and with the community advancing Hive transactions and streaming support (Apache Hive, 2018), streaming scenarios and update operations in Hive are becoming more streamlined. Currently, in the implementation of BDWing systems, practitioners should spend some time studying how these systems work, as well as their advantages and disadvantages, in order to implement an adequate and stable storage system for the BDW, since there is no optimal solution for all implementation contexts.

Regarding data access, analytics, and visualization, there are several technologies that can be used for specific tasks. Spark MLlib and Mahout are two machine learning and data mining libraries that make use of distributed processing to extract patterns from a large volume of data. R, Python, and WEKA, for example, can also be used for this purpose, but one should be aware of their limitations in Big Data environments, as previously discussed in section 2.4.2.3. However, during the last years, these technologies began to include processing components that are able to establish connections to distributed systems such as Spark and Hadoop. Technically, any machine learning and data mining technology able to process large amounts of data and with adequate connectors to Hadoop-related systems can be used in a BDW data science sandbox. Still in this context, technologies such as Tableau, Microsoft Power BI, or TIBCO Spotfire can be used to visualize data. Moreover, more customized visualizations can be created with custom-made JavaScript applications (e.g., intensive geospatial analytics – see the SusCity data visualization platform in section 8.4). The data visualization tool being implemented needs to provide adequate connectors for the querying and OLAP technologies. However, in certain scenarios wherein direct access is required, bypassing the querying and OLAP engine is acceptable through the use of native storage drivers (e.g., HDFS, Hive, NoSQL/NewSQL, Kudu, and Druid), or through the development of custom-made Web services (e.g., REST Web services), in order to avoid some incompatibilities, or to assure higher concurrency and efficiency for certain scenarios demanded by data consumers (e.g., concurrent custom-made Web data visualizations).

The querying and OLAP systems are crucial for BDWing, since they provide an interactive SQL interface to query the data stored in the batch storage and in the streaming storage. These systems are frequently mentioned as SQL-on-Hadoop systems, although they also support other data sources like NoSQL databases. There are several alternatives, some of which can be highlighted: Hive (on Tez) (Huai et al., 2014); Drill (Hausenblas & Nadeau, 2013); Impala (Kornacker et al., 2015); Spark SQL (Armbrust et al., 2015); Presto (Presto, 2016); and HAWQ (L. Chang et al., 2014). Benchmarking several SQL-on-Hadoop systems is an advisable step when implementing a BDWing system (Santos et al., 2017), in order to evaluate if their response times, scalability, and SQL compatibility meet the established requirements. Furthermore, evaluating their connectivity with the storage and data visualization systems is of major relevance to implement an adequate and interoperable BDWing system.

Such complex technological infrastructure needs to be secured and properly managed, assuring the fulfillment of the security and privacy policies, as well as making available a set of mechanisms to monitor the behavior of the infrastructure and act accordingly, if necessary. In this context, Ambari can be used to provision, manage, and monitor a Hadoop cluster supporting the BDWing system. Regarding security, there are several technologies that can be used depending on the specific requirements: Kerberos can provide secure authentication for users and resources; Knox can provide perimeter security, hiding the details of the cluster's access points and blocking services; Sentry can be used to define adequate authorization policies to access data; and Ranger, which is similar to Sentry, provides a centralized platform for policy administration, authorization, auditing, and data protection (HDFS encryption). There are other ways of assuring data security and privacy, such as using specific encryption mechanisms or access control lists made available by different technologies.

To conclude this section, there are some relevant guidelines that should be taken into consideration when deploying an adequate infrastructure for BDWing:

1. Plan the infrastructure to mainly scale horizontally (scale-out), in order to reduce costs and leverage the full potential of emergent Big Data technologies like Hadoop. "Because Hadoop uses industry-standard hardware, the cost per Terabyte of storage is, on average, ten times cheaper than a traditional relational DW" (Krishnan, 2013);
2. Co-locate storage and processing nodes in the cluster, in order to avoid moving data from one node to another, causing bottlenecks in the network (C. L. P. Chen & Zhang, 2014). As can be seen in Figure 4.3, storage and processing nodes are always co-located. This means that querying and OLAP technologies should be installed in all the storage nodes, thus data is not moved across nodes when the data consumers submit a request;

3. Implement a Just a Bunch of Disks (JBOD) configuration for each storage node. If a Redundant Array of Independent Disks (RAID) configuration must be used, implement a RAID-0 strategy (W. Xu, Luo, & Woodward, 2012);
4. Implement at least a 1-gigabit Ethernet network infrastructure (Shvachko, Kuang, Radia, & Chansler, 2010).

4.3 Method for Data Modeling

This section presents the data modeling method to design the data structures stored in the indexed storage component of the BDW. It discusses how data should be modelled according to specific data structures denominated as analytical objects, which include descriptive and analytical attributes (and families). Moreover, other concepts are also presented and discussed in this section, such as materialized objects, granularity keys, atomic values, collections, partition keys and bucketing/clustering keys. All these concepts are presented in the general data model (Figure 4.4). Finally, this section also discusses the concept of complementary analytical object, proper ways of joining and uniting batch and streaming analytical objects, strategies to handle dimensional data (outsourced descriptive families), and some data modeling best practices.

4.3.1 Analytical Objects and their Related Concepts

In this work, an **analytical object** is defined as an isolated subject of interest for analytical purposes. Analytical objects are highly denormalized and autonomous structures that are able to answer queries without the constant need to join dimension and fact tables. The benefits of full denormalized structures in terms of performance and ETL simplicity is a topic periodically discussed and evaluated by the DWing community (Jukic et al., 2017; Santos et al., 2017; Santos & Costa, 2016; J. P. Costa et al., 2011), as previously seen in section 3.1. Typical analytical objects found in organizations may include: sales; purchases; inventory management; employee vacations; employee performance; (potential) customer interactions; customer complaints; transactions during the manufacturing process; among many others. In order to identify an analytical object, one just needs to identify a subject of interest in a specific analytical context. They might be found in traditional business

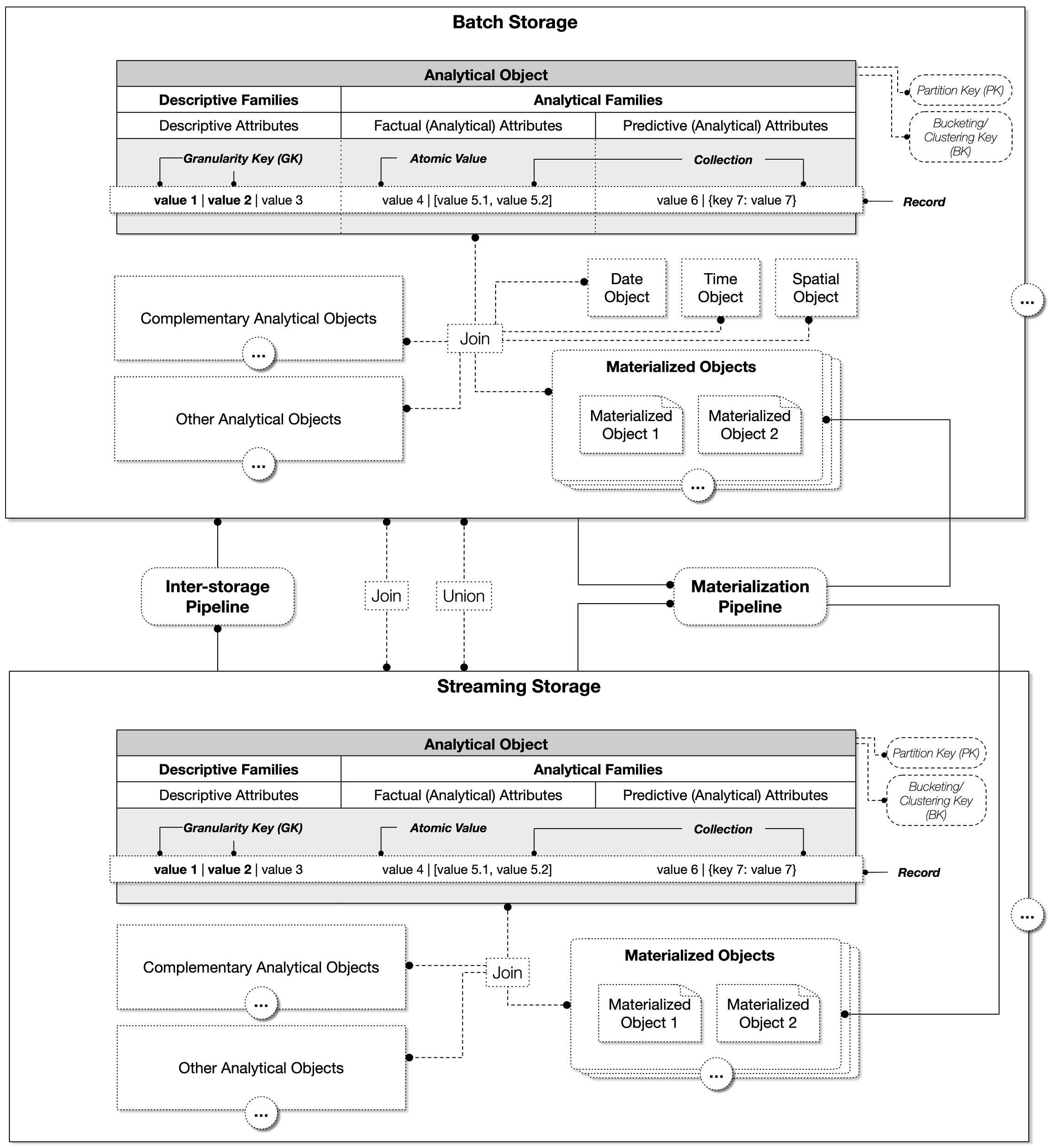


Figure 4.4. General data model.

processes or in new organizational contexts, such as social media interactions and initiatives, recommendation systems, or sensor-based decision-making, for example. An organization can identify analytical objects by either looking at the data currently being produced (data-driven), or by looking at its current goals and start collecting data to fuel these analytical objects (requirements-driven).

An analytical object includes **descriptive** and **analytical families**, as well as **descriptive** and **analytical attributes**, respectively. Families are just a logical representation to group related attributes, and there is no need to physically implement them in the storage system. Descriptive attributes provide a way of interpreting analytical attributes through different perspectives, using aggregation or filtering operations, for example. One can associate them with the attributes found in the traditional dimensions of a DW (Kimball & Ross, 2013). Natural keys (e.g., product code, employee code, and customer code) can also be included as descriptive attributes, if the practitioner foresees an application for these attributes (e.g., specific analyses or update operations on records). In contrast, **analytical attributes** provide numeric values (sometimes embedded in complex/nested data structures that also contain text data) that can be analyzed through the use of the different descriptive attributes (e.g., grouped or filtered), including **factual** and **predictive attributes**. Factual attributes represent numeric evidences of something that happened in a specific record of the analytical object, and can be associated with facts in a traditional fact table (Kimball & Ross, 2013). Predictive attributes provide insights retrieved from the application of data science models and, therefore, they do not represent numeric evidences of something that happened, but rather an estimate of what happened or a prediction of what can happen in a near future. Predictive attributes are a crucial concept to adequately integrate predictive capabilities in the BDW, and can also store relevant patterns extracted from unstructured data (e.g., text, images, and video).

A **record** of an analytical object stores all the values corresponding to an event associated with that object, taking into consideration its different attributes. Descriptive and analytical attributes can contain atomic values or collections. **Atomic values** are stored as simple data types, such as an integer, float, double, string, or varchar. **Collections** store more complex structures like arrays, maps, or JSON objects. These complex and nested data structures, together with a flexible denormalized model without rigid relationships between tables, allow the exploration of the full potential provided by Big Data storage systems.

The **granularity key** is a relevant concept associated with an analytical object. The granularity key is tightly coupled with the analytical object, identifying the level of detail of the data that will be stored in each record. The granularity key of an object is defined by one or more descriptive attributes that uniquely identify a record, although this constraint does not have to be physically implemented in the storage system through a primary key, since some Big Data storage systems may not support such concept. One only needs to assure that each record complies with the granularity key of the object, which defines its level of detail.

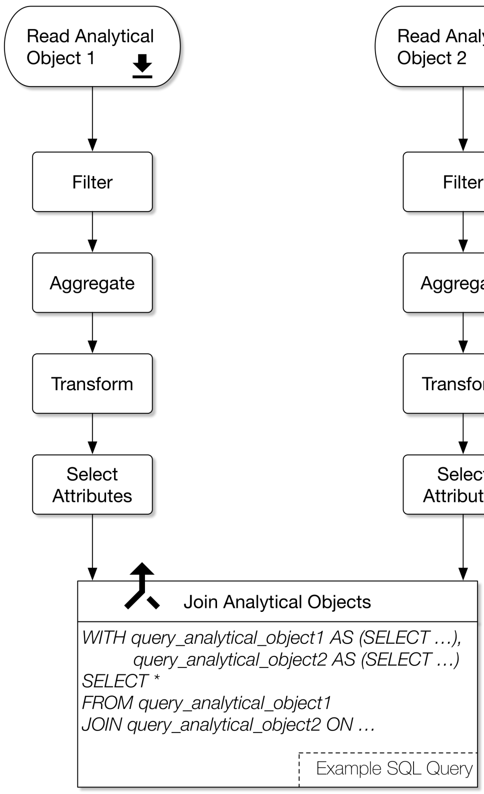
Take as an example an analytical object *"sales"*. Its granularity key can be defined solely by the unique identifier of the sales order. In this case, each record stores the general data about the sales order. The data about products sold in this order can be stored in a collection, or not stored at all, if for some reason there is no interest in that analysis. However, if the granularity key of the analytical object *"sales"* is defined by the identifier of the sales order and the identifier of the product, one record per product will be stored. There is no rigorous rule for preferring collections over redundant data stored across records, and vice versa. System orchestrators should consider their current preferences, skills, and technological or infrastructural constraints (e.g., some querying technology may not support collections, or the size limitations in collections may not be suitable for that context). This will depend on the implementation context. In the proposed approach, the granularity of the analytical object is never considered a limitation, nor does one apply any specific rule or guideline.

As discussed in subsection 4.1.3.3.2, an analytical object can be partitioned and bucketed/clustered by specific descriptive attributes (technically, using analytical attributes is perfectly possible as well, although not as usual). The attributes that are used to partition the analytical object form the **partition key**, which fragments the analytical object into more manageable parts that can be accessed individually. This work does not provide a rigorous rule to partition analytical objects, but encourages system orchestrators to use time and/or geospatial attributes as the partition key (E. Costa et al., 2018), since data can be typically loaded and filtered in hourly/daily/monthly batches for specific places (e.g., cities, regions, and countries). This will obviously depend on the implementation context, but this is typically an adequate strategy for several contexts. Another advantage of partitions can be highlighted in scenarios wherein data should be updated (e.g., perform a batch update because some records were modified or were previously incorrect), which allows practitioners to recompute just the required partitions instead of the entire analytical object. In contrast, the attributes used as **bucketing/clustering key** assure that a range of records are stored in the same group/bucket or sorted in a certain way. However, system orchestrators need to plan this strategy according to frequent access patterns requested by data consumers. The proposed approach highlights the relevance of this concept, but does not aim to provide any rule in this area, due to the fact that it may vary significantly according to the implementation context.

4.3.2 Joining, Uniting, and Materializing Analytical Objects

In the proposed approach, analytical objects can complement each other. Although there are no physical relationships implemented in the storage system, Big Data querying technologies (e.g., SQL-on-Hadoop systems) are able to join different datasets given specific attributes. Therefore, an analytical object may contain, in its descriptive attributes, the attributes that correspond to the granularity key (or part of it) of another object. In this case, an object is considered a **complementary analytical object** if its granularity key (or part of it) is included in another analytical object (e.g., the *"customer account"* object in section 5.2*,* whose part of the granularity key is referenced by another object, and the *"product"* object in section 5.1, whose granularity key is fully referenced by another object). Such integration allows for the association between two analytical objects through a join operation. Another type of association can be made using descriptive attributes that do not correspond to the granularity key of the analytical objects. In this case, analytical objects can be joined using regular descriptive attributes, such as a simple date, for example. A date may not define the granularity key of an analytical object, but it can be used as a join attribute between analytical objects. If many to many associations are identified between analytical objects, one can use collections to solve this issue, i.e., one analytical object contains a collection in its descriptive attributes that stores the association with many records of another analytical object. Once again, it must be highlighted that there is no physical relationship between analytical objects, neither it is mandatory to prepare and enrich data to create these associations between analytical objects. Technically, analytical objects can be joined by any attribute without practitioners being concerned with foreign key relationships and indexes. Certainly, there are many contexts in which analytical objects are analyzed independently, without ever needing to join them. However, whenever necessary, the approach offers support for it.

At this point, a question may begin to emerge: *"If a new denormalized approach is being proposed to solve the complexity in join-dependent data models, how can one perform efficient join operations between analytical objects if they can potentially store Gigabytes, Terabytes, or Petabytes of data?"*. To answer this question, Figure 4.5 presents the process of joining analytical objects, which highlights the need to execute all the required operations in each analytical object through the use of subqueries, or relying on efficient query optimizers to adequately and automatically process both sides of the join operation before the join itself occurs. Then, and only then, the results of these subqueries (or pre-join processing from query optimizers) are joined accordingly. This approach vastly reduces the complexity of join operations, since each subquery on each side of the join is already as aggregated (or filtered) as possible. Figure 4.5 provides an example SQL query showing how to perform this type



```
WITH query_analytical_object1 AS (SELECT …),
         query_analytical_object2 AS (SELECT …)
SELECT *
FROM query_analytical_object1
JOIN query_analytical_object2 ON …
```

Figure 4.5. Process of joining analytical objects.

of join operations. If the *"WITH"* keyword is not compatible with the current querying and OLAP technology, one can also make use of subqueries in the "*FROM*" or *"JOIN"* clause. The same concepts are also valid for union operations.

This process of joining analytical objects should be applied in each join operation, not only including complementary analytical objects, but also **materialized objects** and analytical objects in different storage systems (see Figure 4.4). Since analytical objects can have a significant number of records, joining them can become a time-consuming task, even when using the join approach presented in Figure 4.5. It is in this context that materialized objects are useful and efficient. Complex and long-running queries can be materialized through the materialization pipeline (see Figure 4.4), giving origin to the materialized objects, which can be further joined with other analytical objects. The materialization pipeline also assures the update of materialized objects with new data. Materialized objects can be stored either in the batch storage or in the streaming storage, depending on the access patterns of data consumers (e.g., using NoSQL databases for the streaming storage can provide adequate random access capabilities for specific analytical scenarios). Summarizing the concept of materialized objects, it can be concluded that they are able to store the results of time-consuming queries, increasing the performance of the BDWing system, since several data consumers can consume this materialized object much faster than the original analytical objects. Consequently, materialized objects may be analogous to OLAP cubes in traditional DW environments, containing pre-aggregated data meant to be consumed in a faster and more efficient way.

Besides join operations, this work also considers the use of union operations, typically useful to combine analytical objects stored in the batch storage with analytical objects stored in the streaming storage. Uniting analytical objects in different storage systems enables the visualization of batch and streaming data using a single query. Also relevant is the fact that queries can take advantage of union operators while the inter-storage pipeline does not transfer the records from a streaming analytical object to the corresponding batch analytical object.

4.3.3 Dimensional Big Data with Outsourced Descriptive Families

In certain contexts, data still remains highly relational and dimensional, i.e., different analytical objects will share common descriptive families. One adequate example is *"sales transactions",* which can be analyzed using several descriptive families like *"customer", "product", and "supplier",* for example. Besides that, these descriptive families can be included in several other analytical objects, such as *"customer complaints", "purchases", "inventory management",* among others.

As discussed previously in this section, the proposed approach allows the use of joins between analytical objects. However, it does not include the concept of dimensions. Typically, flat structures are preferred to avoid the cost of join operations and to achieve better performance, as demonstrated in Chapter 7. However, completely flat structures vastly increase the storage size of the BDW when compared to dimensional structures (e.g., star schema). The problem becomes really severe if multiple analytical objects share the same descriptive families, because the increase in storage size can get out of control, especially if these descriptive families have a significant number of attributes. Obviously, one may be able to sacrifice storage space, which is cheaper than processing power, in exchange for better performance. Taking into consideration the insights provided in Chapter 7, it may be advantageous to use a flat analytical object that is 3 times bigger than the corresponding star schema. However, if one considers contexts with several flat analytical objects that share the same descriptive families, the BDW size can grow in a rate that the organization cannot sustain. Furthermore, there are certain contexts in which star schemas can outperform flat analytical objects (see subsection 7.2.3).

For these reasons, and supported by the results presented in Chapter 7, one promotes the following guidelines for modeling dimensional Big Data using the concept of **outsourced descriptive family**:

1. A descriptive family should be outsourced to a complementary analytical object if one or a combination of the following conditions is verified:
    a. The descriptive family is frequently included in other analytical objects (phenomenon that is relatively similar to the conformed dimensions concept in Kimball's

approach), avoiding extreme redundancy in the BDW, especially if the descriptive family has a considerable number of attributes. Otherwise, outsourcing frequently reused descriptive families with few attributes may not be compelling;

b. The descriptive family has low cardinality, i.e., its distinct records will form a low-volume complementary analytical object that easily fits into memory, enabling the capability to perform map/broadcast joins in SQL-on-Hadoop engines (see subsection 7.2.1 and 7.2.3);

c. The frequency of data ingestion of the complementary analytical object is equivalent to the other analytical objects it is related to. For example, if one is using the BDW to store and process streaming data from social networks, having a *"user"* complementary analytical object is only practical if the users' data is also streamed to the BDW as soon as a customer signs up for the social network, otherwise the BDW will suffer from problems such as the late arriving dimensions phenomenon in dimensional DWs (Kimball & Ross, 2013). If such design requirement is not possible to fulfil for some reason, then flat analytical objects are preferred in these contexts;

d. The descriptive family alone can provide considerable analytical value when analyzed independently, forming a real analytical object. For example, *"customer"* may serve as a complementary analytical object when outsourced from a descriptive family of another object, but it can also be used independently to measure customer performance if it contains analytical attributes related to average sales, average returns, current reviews, among other factual or predictive data.

2. Complementary analytical objects resulting from the outsourcing of descriptive families should use natural granularity keys, as maintaining surrogate keys is not practical in most of the BDW storage technologies, both for batch and streaming scenarios (e.g., lack of proper support for auto-increments). Searching for the surrogate keys corresponding to the natural keys flowing through CPE workloads also becomes very inefficient and unpractical, especially in streaming workloads;

3. The records of complementary analytical objects resulting from the outsourcing of descriptive families should also be designed to be immutable, whenever possible, similarly to the records of regular analytical objects. If such is not possible or applicable for the requirement being fulfilled, these complementary analytical objects should at least be either efficient to update or easy to recompute using a CPE workload (fully or partially using partitions), in order to avoid dealing with complex SCD-like scenarios. Despite this guideline, practitioners should feel free to create mutable complementary analytical objects (as well as regular analytical objects) whenever the technologies storing the batch/streaming object support proper updates. Again, BDWing technology is evolving in this matter, and this guideline must not be seen as absolutely mandatory if performance is not severely compromised (see subsection 5.2.1 for further discussion on this topic);
4. By simply outsourcing descriptive families to complementary analytical objects, only descriptive attributes are considered. This means that the resulting complementary analytical objects do not hold any analytical families and attributes, and, therefore, any analytical value. Although this is possible in the proposed approach, it somehow violates the principle that analytical objects should be autonomous structures that can answer some queries without the need for any join operations. This principle will not be true for a complementary analytical object *"customer"* that will only be used to complement other analytical objects, for example, as previously exemplified in this section. Consequently, one encourages practitioners to use the concept of "aggregated facts as dimension attributes" in Kimball's approach (Kimball & Ross, 2013). Although not mandatory, this technique allows practitioners to include analytical attributes (facts or predictions) in these complementary analytical objects, meaning that these attributes can not only be used for filtering or labelling records, but also to perform calculations, as one is modeling an analytical object after all, and not only a traditional dimension. Using this strategy, the *"customer"* analytical object can be used to independently answer specific queries, such as *"what is the average revenue generated by certain customers?",* without needing to query both the *"sales"* analytical object and the *"customer"* analytical object. Following this example, the *"customer"* analytical object can

even include predictive attributes, such as a cluster label based on the customer's value to the organization (see subsection 6.5.1). Obviously, similarly to what Kimball and Ross (2013) state, these pre-aggregations create more burden in the processes that make data flow to the system, but also provide more analytical value and, sometimes, eliminate the need for complex and costly queries. Such trade-offs still hold true in the proposed approach.

4.3.4 Data Modeling Best Practices

This subsection presents several best practices that can be applied to a BDW data model, in order to clarify some questions that may arise in its design and implementation, including the use of null values, the preparation of spatial and temporal attributes, and the modeling of records as immutable events.

4.3.4.1 Using Null Values

The use of null values in the BDW is not forbidden, and for certain cases is even advisable. However, there are some relevant practices that must be taken into consideration. Regarding analytical attributes, one advises the use of *null* to indicate the absence of a value, since null values are often ignored in querying, OLAP, and visualization technologies, which do not take them into account when performing aggregations on data. If numbers like *0* or *-999,* for example, are used to indicate the absence of a value, every time an aggregation is performed, filters need to be applied first to ignore these values, since they affect an average/sum calculation.

In contrast, regarding descriptive attributes with a text data type, the use of *"Unknown"* or *"Not Applicable"* is more user-friendly and appropriate when using these attributes to aggregate analytical attributes. However, there are certain data types in which the use of null values is still preferable (or the only solution) to indicate the lack of values in descriptive attributes, namely types such as boolean, arrays, or maps.

4.3.4.2 Date, Time, and Spatial Objects vs. Separate Temporal and Spatial Attributes

The date and time objects presented in Figure 4.4 include several temporal attributes that complement the analytical objects stored in the BDW. Including these attributes (e.g., *"is holiday", "is weekend", "month",* and *"year"*) in the analytical objects can severely increase their storage size and consequently affect the stability and performance of the BDWing system. These objects are considerably small and will not significantly affect the performance of the BDW by requiring a join operation, as seen in Chapter 7.

One encourages the use of the date and time objects to store a vast set of temporal attributes that can be used by the analytical objects. An adequate practice would be the use of standard dates (e.g., *"yyyy-mm-dd"*) and standard time representations (e.g., *"hh:mm"*) in all analytical objects, which would then allow to join them with the date and time objects.

Moreover, with this approach, practitioners can also use several UDFs to interact with the single date or time attributes stored in the analytical objects, in order to create new attributes not present in the date and time objects. Extracting attributes at runtime may not significantly impact the query execution time, sometimes just showing insignificant increases. Nevertheless, one does not discourage the use of separate temporal attributes (e.g., *"day", "month", "year", "hour",* and *"minutes"*), quite the contrary, since they are still significantly useful in certain contexts. One particular example is the specification of partition keys, given that, frequently, only simple data types like strings or integers can be used in the partition key. Therefore, if one needs to use *"month"* as the partition key, there may be the need to have a separate temporal attribute *"month"*. Concluding, the use of the date and time objects or the use of separate temporal attributes depends on the implementation context, and system orchestrators should evaluate the most adequate solution for the context.

Regarding the use of spatial objects, they prove to be significantly useful for standardizing spatial attributes across the analytical objects of the BDW, such as assuring that a city and a country have the same exact meaning (and characteristics) throughout the entire data model. However, practitioners should be careful with large and detailed spatial objects (e.g., *"building number"*, *"street name"*, and *"coordinates"*), because join operations can certainly create performance bottlenecks in Big Data contexts. Therefore, one should prefer maintaining these highly detailed characteristics (e.g., *"building number"* and *"coordinates"*) in the analytical object in a denormalized form, while creating less granular spatial objects like *"city"*, for example, which can also include the corresponding countries in a denormalized form (see subsection 5.3.2). However, highly detailed spatial objects are acceptable in scenarios wherein one can predict their growth, because the number of records they can have is already known or expected *a priori.*

4.3.4.3 Immutable vs. Mutable Records

As previously discussed, one encourages practitioners to model analytical objects as a set of immutable events. As Marz and Warren (2015) discuss, simpler implementations can be achieved by eliminating the complexity associated with update operations, which can sometimes raise concurrency issues. This modeling style will probably suite most of the analytical scenarios in organizations, since the granularity of each analytical object can be rethought to treat each record as an immutable event.

Take as an example an analytical object to store customer complaints (Figure 4.6). A certain organization knows that a customer complaint has several states over time. A possible approach, which allows the records to be updated, is to have one analytical object that stores a customer complaint in each record. When a recently opened customer complaint arrives at the BDW, it is stored in a record with the status *"open"*, not having a due date yet. In the meanwhile, this record will have to be updated when the customer complaint is *"finished"*. In contrast, another approach is to model the analytical objects according to a set of events related to customer complaints. When a recently opened customer complaint arrives at the BDW, a record is created containing the status and the date associated with that status. When the status of the customer complaint changes, new data arrives at the BDW, and a new record for each state change is stored. This second approach assures that each record is immutable, eliminating the need for update operations.

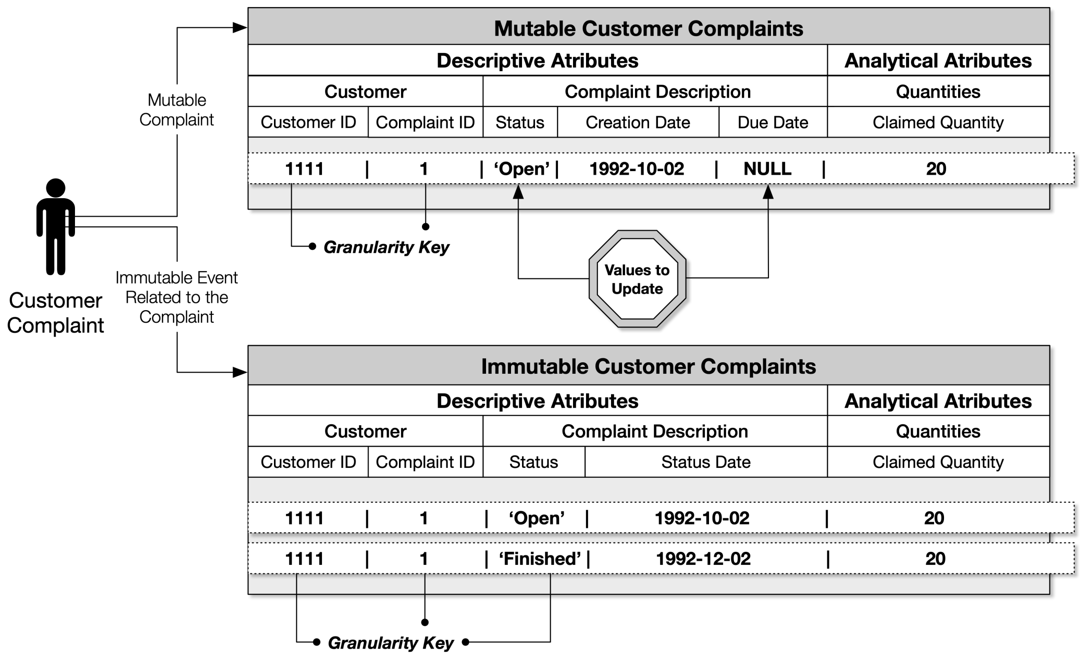


Figure 4.6. Example of immutable and mutable records.

Despite the fact that queries need to be structured in different ways, the two analytical objects presented in Figure 4.6 are able to answer the same analytical questions. Furthermore, one can argue that the immutable analytical object is more oriented towards ad hoc querying, wherein data consumers can discover relevant patterns and delays among processes related to customer complaints. However, the proposed approach does not forbid the use of mutable analytical objects, considering that practitioners plan the BDW technological infrastructure according to the random access trade-offs and limitations of the several technologies presented in section 4.2. Modeling analytical objects as a set of immutable events is a suggestion, not a rigorous rule, since updates can be performed on storage systems that adequately support random access operations, as previously discussed in subsection 4.1.3.3.2 and further explored in subsections 5.1.3, 5.2.1 and 5.2.4. As previously discussed, technology is constantly evolving, and these trade-offs or limitations may not be an issue in certain implementation contexts. The proposed approach does not aim to restrict any use of specific functionalities, giving practitioners an adequate flexibility regarding data modeling. However, one highlights the need to assure that the logical components, data flows, infrastructure, and data model are all properly integrated and aligned to serve the business goals.

4.3.5 Data Modeling Advantages and Disadvantages

This modeling approach based on denormalized and nested data is seen as a crucial step to achieve a flexible storage in the BDW. When compared to the relational data modeling approaches found in traditional DWs, this work trades less redundancy and smaller DW sizes for the following advantages:

1. Assures better performance in query execution, due to the lack of constant join operations between dimensions and fact tables imposed by traditional dimensional and 3NF data models;
2. Provides a flexible denormalized model without the need to perform complex surrogate key maintenance and lookups for each insert, allowing for simpler and more efficient batch and streaming CPE processes, by avoiding known-problems such as SCDs and late arriving dimensions (especially in streaming scenarios);
3. Preferably focuses on modeling analytical objects as a set of immutable events and, therefore, there is no need to frequently deal with concepts such as SCDs (Kimball & Ross, 2013). However, as explored in subsection 5.2.1, this does not mean that mutable objects are forbidden, and when using them, some of the SCDs considerations still hold true;
4. Avoids other traditional dimensional data modeling, ETL, and DW maintenance problems like having to consider several types of dimensions (e.g., mini dimensions, junk dimensions, shrunken dimensions, and bridge tables), which in Big Data contexts are arguably unnecessary, as saving some storage space and achieving less-redundant data models, may come at the cost of spending a considerable amount of time in data modeling, implementing ETL processes, and maintaining the DW (not to mention performance costs), which may be a compelling reason why, nowadays, practitioners pursue more flexible analytical contexts. Consequently, despite some data redundancy, in several contexts, the proposed approach provides simpler data models than a dimensional or 3NF DW, reducing the time needed from collection to analytics;
5. Highlights nested structures as relevant constructs in certain BDW data models and applications, which can be significantly useful in certain contexts (see Chapter 5 and Chapter

8), such as storing geospatial objects for intensive geospatial analysis, and solving many to many relationship issues typically found in relational databases (e.g., a customer complaint may have several responsible employees, which are also responsible for several customer complaints).

Nevertheless, the proposed data modeling method has some characteristics that may be considered as disadvantages when compared to the aforementioned methods to design DWs, which include:

1. The total size of certain BDWs (typically the ones whose data sources are highly dimensional with frequently reused dimensions) may increase drastically due to extreme denormalization, reason why the approach introduces the concept of date/time objects, spatial objects, complementary analytical objects, and outsourced descriptive families. Consequently, practitioners should take into consideration the guidelines provided in subsections 4.3.4.2 and 4.3.3, as well as the data models explored in Chapter 5, mainly in section 5.1 and 5.3, as the original data sources tend to be highly dimensional, being the same dimensions reused frequently by different business processes/analytical subjects. Without these strategies, the resulting BDWs would be significantly larger than the DWs based on star schemas or 3NF data models. Nowadays, storage size is cheap, but may often lead to unnecessary concerns and costs regarding systems administration, which can be avoided by using the constructs discussed above, whenever practical and applicable;
2. If the data source fueling an analytical object is based on a relational database, the CPE workloads for that object may need to include a considerable amount of join operations, either being performed in the source (as a SQL query for example), or being performed in the technology supporting the workloads. However, in Big Data contexts, many of the data sources are non-relational (e.g., sensor data, NoSQL databases, spreadsheets, XML files, and JSON files), making the proposed method for data modeling significantly more compelling and simpler for BDWs.

Chapter 5. Big Data Warehouses Modeling: From Theory to Practice

After the presentation of the general data modeling method in section 4.3, this chapter explores its use in several BDWing contexts, since more practical examples and real-world applications may be required for practitioners to master some of the proposed data modeling guidelines. Consequently, this chapter aims to provide several examples of BDWing applications using the proposed data modeling method, in order to clarify some of the guidelines provided previously, and to evaluate their suitability in a broader scope of analytical applications focused on: traditional enterprise setups with human resource management, purchases, sales, promotions, goods returns, inventory management, and production process; financial market; retail; code version control systems; media events (broadcast, printed and Web news); and air quality measurement systems.

5.1 Multinational Bicycle Wholesale and Manufacturing

As already seen, Big Data can be defined as data whose characteristics impose severe difficulties to traditional DWing platforms. Frequently, there may be a misconception regarding the need to satisfy all Big Data characteristics to deploy a BDW, such as the need to process vast amounts of unstructured data arriving at theoretically unlimited velocities. However, in this section, one will present how a BDW can be modelled to encompass traditional business processes like human resources management, sales, purchases, production, among others.

Obviously, traditional DWs have long been the backbone for analytics over traditional and structured business processes, but this section provides a way of modeling such complex scenario in a BDW created using the proposed approach, in order to provide more data modeling simplicity, less ETL effort without complex dimension maintenance and surrogate key lookups, and more processing efficiency by reducing the constant need to join several tables, while being fully compliant with a shared-nothing and open source vision of what a BDW should be. Such benefits can attract organizations that are starting their analytical platforms based on open source Big Data technologies, as well as organizations looking to replace their expensive DW appliances or limited relational databases.

For this example, one uses the Adventure Works database, a relational OLTP database from a fictitious company that manufactures and sells bicycles, included as part of the Microsoft SQL Server samples (Microsoft, 2018). This database has a relatively complex schema that covers a wide spectrum of business processes and entities (e.g., employees, vendors, customers, stores, departments, products, work/production orders, purchases, sales, and inventories). The complete representation of the Adventure Works database is available in (Dataedo, 2017).

After applying the data modeling method, the resulting BDW data model can be seen in Figure 5.1, containing 7 analytical objects *("employee history", "sales line", "product review", "product vendor history", "purchase line", "product inventory", and "work order")*, 3 complementary analytical objects *("product", "vendor", and "special offer"),* 1 date object, 1 time object, and 2 spatial objects *("city" and "territory").* Descriptive attributes are divided into descriptive families, while analytical attributes are divided into analytical families, when applicable. Analytical objects can also contain outsourced descriptive families that are linked to a complementary analytical object through a unique identifier (granularity key) of that object, identifying a specific record. Several of these constructs and design guidelines, already discussed in section 4.3, are detailed and exemplified here, not only for this specific example, but also for the other BDW examples in the following sections.

The data model presented in Figure 5.1 sometimes omits certain attributes of the original Adventure Works database, in order to simplify its presentation in this work, such as the omission of the attributes in the *"header"* analytical family of the *"sales line"* analytical object due to its similarity with the *"purchase line*" analytical object or, for example, the omission of the attributes from the *"customer"* and *"sales person"* descriptive families of the *"sales line"* analytical object, due to the wide spectrum of available attributes (different practitioners may choose to incorporate different attributes). Therefore, the main idea is to exemplify the modeling approach and not to extensively enumerate the attributes.

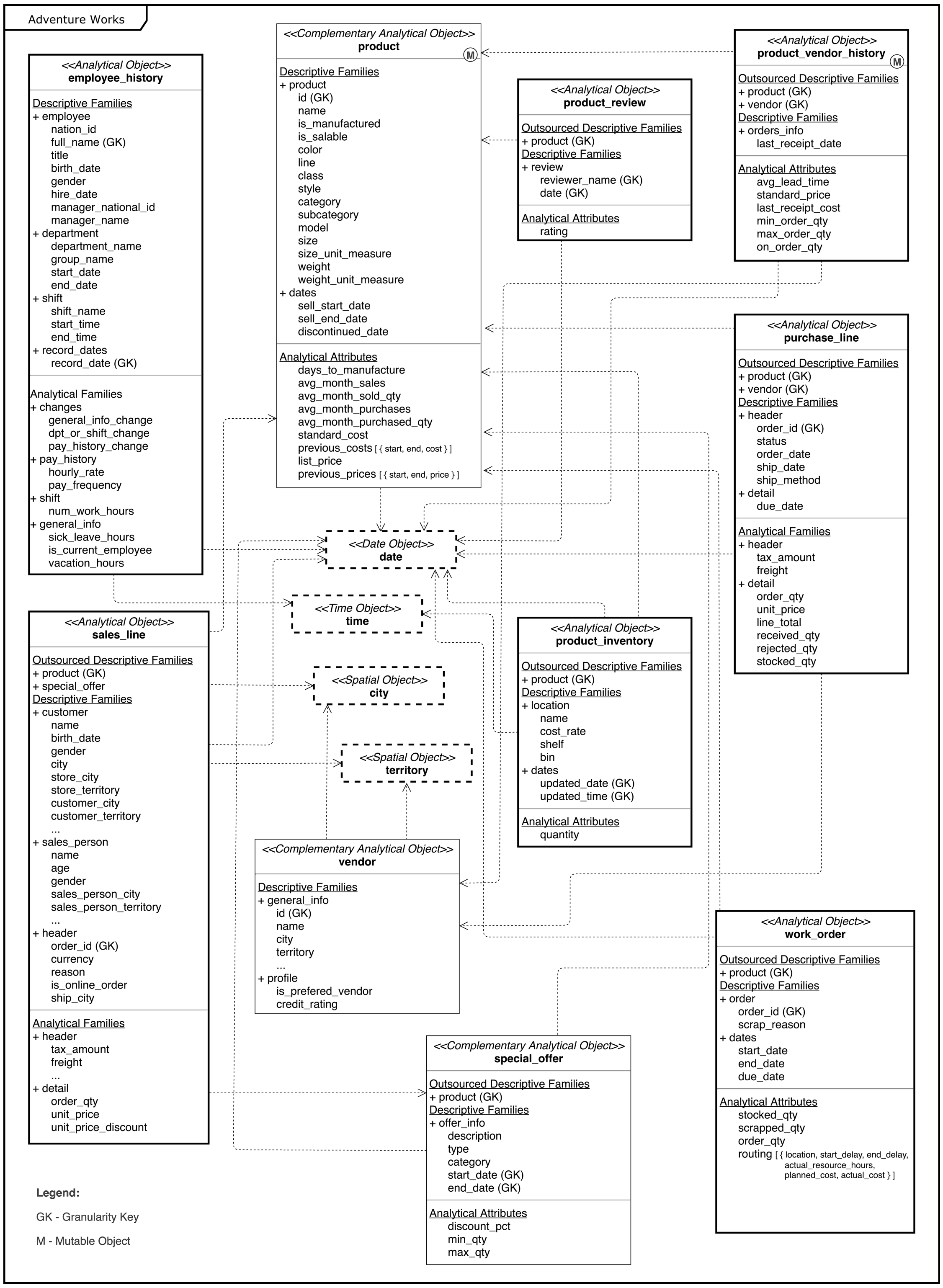


Figure 5.1. Adventure Works BDW data model.

5.1.1 Fully Flat or Fully Dimensional Data Models

The example in Figure 5.1 demonstrates the use of outsourced descriptive families and complementary analytical objects (subsection 4.3.3), using them to overcome extreme redundancy and storage size increase. By revisiting the arguments for the use of these concepts in subsection 4.3.3, one can highlight the following:

1. *"Product"* is an adequate candidate for a complementary analytical object because its attributes would otherwise appear repeated in several analytical objects, as a product is a core business entity in this context. A *"product"* object allows for the standardization of the products information across the BDW, and since new products are not added rapidly in this context, this is an adequate design choice, because it will not severely affect join performance, as broadcast/map joins will still be efficient as time goes by. The *"product"* object by itself holds a significant analytical value, which distinguishes itself from a traditional dimension just to avoid redundancy, as one can be interested in analyzing several metrics regarding products, without needing any additional analytical objects. This is therefore a valuable construct in the approach, and it resembles the concept of "aggregated facts for dimensions" from Kimball and Ross (2013). In subsection 5.1.3, one will detail how this concept can be implemented;
2. For the same reasons, *"vendor"* is also an adequate complementary analytical object that serves two outsourced descriptive families from the *"product vendor history"* and the *"purchase line"* analytical objects. However, in contrast to *"product"*, *"vendor"* does not have any evident analytical attributes, although *"is preferred vendor"* and *"credit rating"* could be considered analytical attributes as well, as the proposed approach offers this flexibility due to the denormalization process, allowing the execution of aggregate functions over any attribute present in the analytical object without involving any kind of join operation. Moreover, as explained above, other analytical attributes can be created (e.g., average monthly purchases). Another relevant consideration is the fact that *"vendor"* is also related to the spatial objects, so one can conclude that, as there is no need to define foreign keys in BDWs created using

the proposed approach, objects in the data model can be flexibly joined, as long as there are common unique identifiers among them (simple or composed);

3. *"Special offer"* is considered a complementary analytical object, although it is only related to the *"sales line"* analytical object and, therefore, it does not necessarily serve the purpose of avoiding extreme redundancy. However, theoretically, it represents a standard analytical object that happens to be joinable with the sales information by a unique identifier. Consequently, as seen in subsection 4.3.2, two analytical objects can be joined together, being the designation of complementary analytical object assigned to the object whose granularity key (or part of it) is included in other objects, which in this case makes *"special offer"* a complementary analytical object of *"sales line";*
4. Other potential candidates for complementary analytical objects could be the *"employee"* and *"customer"* objects. Regarding a possible *"employee"* complementary analytical object, there is employee information in the *"employee history"* and *"sales line"* objects but, in this model, one can consider that only a subset of the employee attributes are relevant for each analytical object, thus denormalization and redundancy is appropriate and, therefore, there is no need for a complementary analytical object integrating the employee information. In the case of the *"customer"* analytical object, since customer information only appears in the *"sales line"* analytical object, there is no apparent need for a complementary analytical object that can be shared by other analytical objects, being the level of denormalization presented in Figure 5.1 appropriate for this context. However, the creation of a *"customer"* analytical object is possible and sometimes encouraged, as can be seen in the data model depicted in section 5.3.

5.1.2 Nested Attributes

Nested attributes are a valuable construct in the proposed modeling method, as they provide a considerable amount of flexibility and a new set of analytical possibilities. As can be seen in Figure 5.1, considering the *"work order"* analytical object, one can observe that although this object stores information at the work order level, the routing attribute stores more granular information at the work order route level, detailing the several production steps of a specific order. This allows for a broader range of ad hoc queries to inspect routing information, without the need for heavy drill across operations. As mentioned in subsection 7.2.4, lambda or explode functions can be used to explore nested data. Nested attributes are also used in the *"product"* complementary analytical object to store the history of prices and costs of the products. These attributes are arrays of structs/rows (or similar data structures), and can serve to analyze price/cost history of a specific product, again, without the need to join tables. These constructs are powerful for ad hoc exploration of data, but require some attention when performing heavy aggregations or filtering operations based on nested values, as seen in subsection 7.2.4. Another relevant aspect to consider is the size of the collections, as they are not meant to grow rapidly, due to the fact that some Big Data technologies may present limitations when performing insert, read, or update operations on large nested attributes. Consequently, they are preferred in scenarios wherein practitioners can estimate their initial size and potential growth.

5.1.3 Streaming and Random Access on Mutable Analytical Objects

As stated in Chapter 4, one promotes the storage of immutable events, not only due to the fact that some of the core concepts of the approach take inspiration from the Lambda Architecture, but also due to some current limitations of Big Data storage technologies when performing update operations (e.g., HDFS/Hive). However, this guideline does not prevent practitioners from modeling and implementing mutable (complementary) analytical objects. In this subsection, one will discuss how mutable objects can be incorporated in a BDW, considering *"product"* and *"product vendor history"* as examples.

As stated previously, some of the analytical attributes of the *"product"* complementary analytical object resemble the concept of aggregated facts for dimensions (Kimball & Ross, 2013) (e.g., *"avg month sales" and "avg month sold qty" attributes*). However, without proper support for update operations, each month, this analytical object would have to be completely reconstructed to store the new monthly values. In contrast, if needed, as discussed in section 4.2, practitioners may opt for storage systems that are suitable for random reads and writes. When choosing a NoSQL database, for example, one does not need to recompute the *"product"* object, just to update the average monthly metrics for each product.

The proposed approach assumes that this type of design choice follows the streaming data flow in Figure 4.1, because this work only suggests NoSQL databases for the streaming storage component, not the batch storage component. However, it is evident that, in this case, the updates happen in relatively large batch intervals, which may or may not be supported by streaming technologies depending on the CPE workload execution frequency (e.g., every time a customer purchases something, each day, or each month). Such assumption forces these analytical objects to be stored in the streaming storage component, regardless of the CPE workload being based on batch or stream processing. This is a design choice of the proposed approach, as the batch data flows still remain considerably similar to constantly inserting/updating values on a streaming analytical object stored in a NoSQL database.

Nevertheless, with the rapidly evolving Big Data technological landscape, support for update operations and ACID transactions is a concern of several storage technologies, and Hive is no exception. Therefore, if practitioners choose a Hive transactional table to store products data, this scenario can be adequately supported by the batch storage component, without the need to store the *"product"* analytical object in a NoSQL database (streaming storage). Transactional tables are significantly optimized in Hive version 3 (Apache Hive, 2018), thus being a relevant feature to explore in future prototypes and production systems. Consequently, nowadays, practitioners do not necessarily have to choose NoSQL databases to adequately perform random insert/update operations with moderate frequency.

The context for the *"product vendor history"* is almost identical to the previous one. In contrast to these two examples, *"employee history"* is an example of how a potentially mutable object can be transformed into an immutable object, as each time some employee data changes (e.g., personal information, department, shift, or salary), a new record is created, which allows for analyzing employee history in significantly flexible ways.

5.2 Brokerage Firm

The financial sector has been increasingly considering the adoption of Big Data techniques and technologies as part of the Fintech phenomena (Gai, Qiu, & Sun, 2018). A brokerage firm, facilitating the trading of financial securities, can represent an appealing application context for a BDW, as it stores and processes vast amounts of daily market and news data, as well as trading and watching data of several securities related to multiple brokers and customer accounts. Consequently, in this section, one models a BDW for a fictional brokerage firm depicted in the TPC Benchmark E (TPC-E) (TPC, 2018), which thoroughly details a concurrent transactional database system for financial brokerage contexts.

In this work, one transforms the TPC-E data model into a BDW data model using the proposed approach (Figure 5.2). The brokerage firm BDW data model is presented in a simplified manner, in order to avoid repeated constructs already detailed in this chapter and, therefore, some (complementary) analytical objects are not detailed at the family or attribute level.

5.2.1 Unnecessary Complementary Analytical Objects and Update Problems

In the BDW data model depicted in Figure 5.2, there are 3 complementary analytical objects: "*customer account*", "*broker*", and "*security*". In this example, *"customer"* and *"company"* could theoretically be included as complementary analytical objects, but due to their lack of isolated analytical value for this specific context, as well as the frequency in which they appear related to other objects, both were not considered as complementary analytical objects, preferring some denormalization steps: *"customer"* data appears denormalized in the *"customer account"* object; *"company"* data appears denormalized in the *"news"* and *"security"* objects.

However, this design decision also means that the *"watch list"* analytical object, which in the original TPC-E model is related to the *"customer"* table and not to the *"customer account"* table, needs to be indirectly joined with the *"customer account"*. In this case, in order to retrieve customer information associated with specific watch list data, one needs to, for example, perform a left outer join retrieving

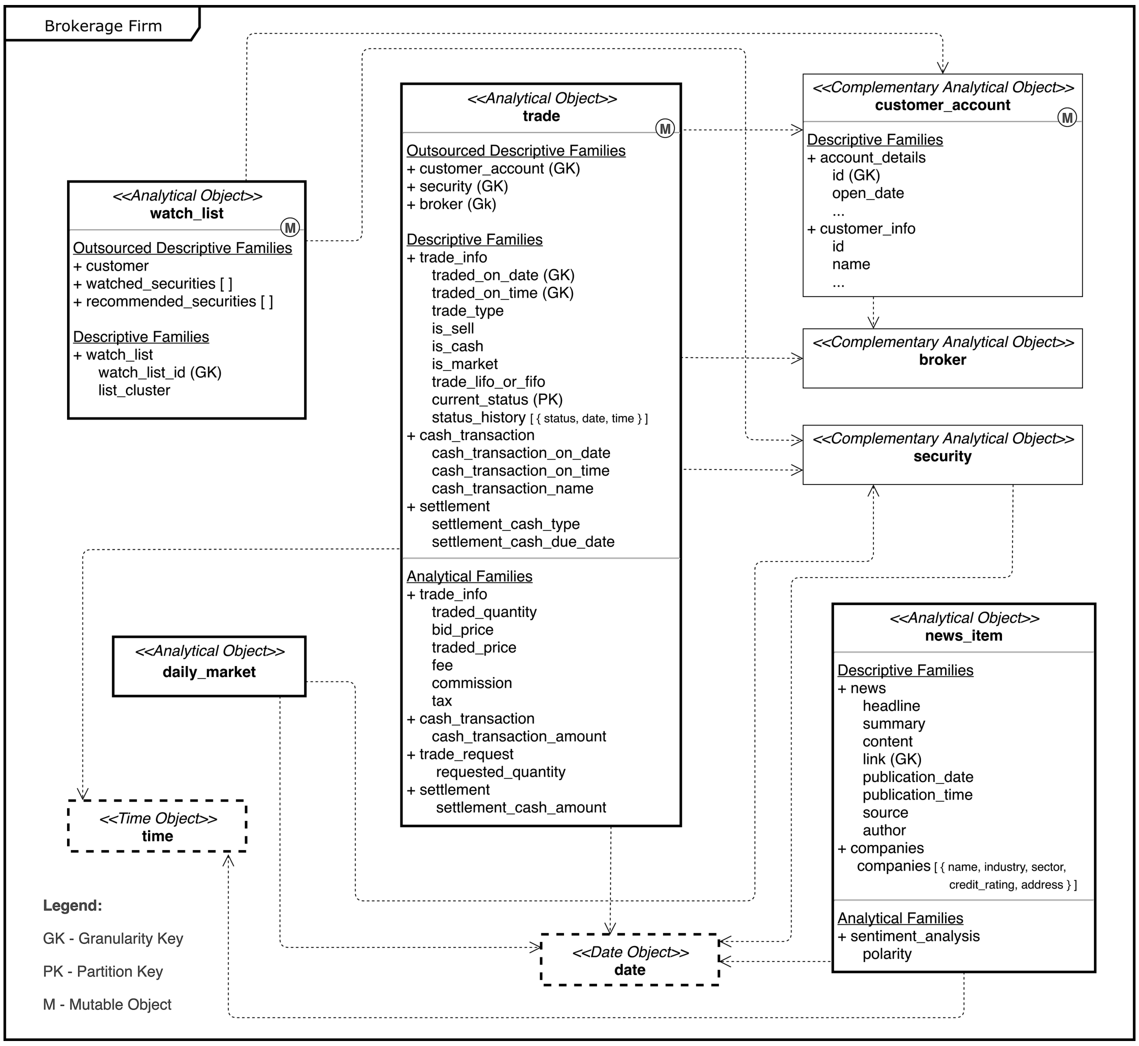


Figure 5.2. Brokerage firm BDW data model.

the customer information from its last customer account. Moreover, if there is a change in some attribute related to the customer, not the customer account, one needs to choose an update strategy:

1. Replace the values in all the related customer accounts by scanning the entire analytical object or several partitions (similarly to SCD type 1);
2. Update only the last customer account;

3. Only update customer accounts when a new account is inserted, as the customer created the accounts before this update, and such information is somehow valuable for business analysis (immutable events strategy);
4. Insert a new record for each customer account with the updated values (similarly to SCD type 2).

If practitioners find this design approach suitable for their use cases, the same can be implemented to provide more simplicity in CPE workloads, otherwise a new complementary analytical object *"customer"* can also be created, as the approach provides this flexibility by delegating some design decisions to practitioners according to their implementation's specificities. Regarding update operations on complementary analytical objects, design choices are often influenced by the adoption of a specific technology (see 4.2 and 4.3.4.3), due to their random access or batch update capabilities. However, some of these choices and challenges are also somehow related to the concept of SCDs (Kimball & Ross, 2013), as some of the underlying challenges of updating denormalized dimensions resemble the challenges of updating complementary analytical objects, due to data redundancy (scanning vast amounts of data to update certain values) and history maintenance.

Several strategies from multiple SCD types (e.g., SCD type 1, 2, and 3) can also be applied to complementary analytical objects, but one needs to consider that the proposed approach does not have the concept of surrogate key and, therefore, practitioners should rely on the originally defined granularity key, as well as modification dates and flags to indicate the current/active records, when needed, in order to appropriately join analytical objects, which creates a slightly more complex granularity key (granularity key information on subsection 4.3.1). In this example, *"customer account"* would not have a simple *"customer account id"* as granularity key, but a complex granularity key like *"customer account id", "insert date", "expiration date",* and *"is current".*

5.2.2 Joining Complementary Analytical Objects

As already dissected throughout this work, the approach considers every table as an analytical object, which can be complementary, or not, depending if they contain descriptive attributes that are outsourced from other objects, or not. Frequently, as seen in this brokerage firm, complementary analytical objects may resemble traditional dimensions, despite the fact that one encourages practitioners to provide analytical attributes for these complementary objects. This is the case for the *"broker"* and *"security"* objects in this example. Considering the guidelines provided in subsection 4.3.3, for BDW data models with significantly large complementary analytical objects created with the purpose of supporting outsourced descriptive families, if interactive query execution is a priority, one should consider denormalizing data even further, by including attributes from the *"security"* object in the *"trade"* object for example, taking into consideration the data model of this brokerage firm. This may be the case for the *"security"* complementary analytical object, which can become significantly large depending on the securities being traded in this context.

5.2.3 Data Science Models and Insights as a Core Value

One of the main design concerns of the proposed approach is to close the gap between data science models/results and the BDW data structures that store the data for later use. Throughout this work, one already discussed this topic several times (see subsection 4.1.2 and section 6.5). For this brokerage firm, one can apply the concept of predictive attributes to make data science results available to other analytical applications (e.g., dashboards, ad hoc querying, custom-made applications, and simulations). Such examples may include: the *"recommended securities"* and the *"list cluster"* in the *"watch list"* object, which can be derived from a recommendation engine and a clustering algorithm respectively; and the *"polarity"* attribute from the *"news item"* object, which may be the result of a sentiment analysis process that classifies a news item as being positive or negative, in order to enrich the decision-making processes that the BDW can support.

In contexts where custom-made applications may need to access the data stored in the BDW, such as a brokerage firm Web site that recommends securities to millions of customers based on the *"watch list"* recommendations, the *"watch list"* analytical object becomes an adequate candidate for a streaming analytical object that is stored in a NoSQL database to provide adequate random access to millions of concurrent users, a use case wherein NoSQL databases thrive (strategy already discussed in subsection 4.1.3.3.2, section 4.2, and subsection 5.1.3).

5.2.4 Partition Keys for Streaming and Batch Analytical Objects

Considering this financial brokerage context, the *"trade"* object is noticeably the analytical object in which most of the decision-making process will be centered in. Analyzing a stream of trading data can provide significant business value, accelerating the decision-making process in several forms. However, a trade follows different stages (e.g., request, cash transaction, and settlement), and as modelled in Figure 5.2, it may have different attributes filled in depending on its type (e.g., cash or margin trade).

One of constructs that can be used in this context is the partition key. By using this construct, practitioners can easily use the same analytical object to store both batch and streaming records, in this case, trading data. For example, if one partitions the *"trade"* object using the *"status"* attribute (or any other attribute available in the transactional system indicating different states of the trade), both batch and streaming data can be stored in the same analytical object and in the same storage technology (e.g., Hive), wherein the trade can be constantly updated until it reaches a state of completion. By using different Hive partitions to divide batch and streaming records of the same table, one can have different schemas for each partition, which means that some attributes of the *"trade"* analytical object may only be included in specific partitions, depending on the state of the trade (e.g., requested or settled). This is possible for storage technologies that can have schemas defined at the partition level, which is the case when using Hive.

This capability also means that the frequent use of update operations (e.g., Hive transactions) can be restricted to streaming partitions, as once the trade reaches completion, the chances of it being updated are rather reduced. This demonstrates the flexibility of the proposed approach, which allows for a seamless integration between batch and streaming data, and efficient ways of conducting update operations, despite the fact that it encourages the modeling of immutable objects whenever possible. However, in this case, in order to provide a timely and interactive analysis, the *"trade"* analytical object can be made mutable without significantly sacrificing efficiency, due to technological evolutions like Hive transactions (Apache Hive, 2018).

5.3 Retail

In this section, one provides an example of a BDW that supports a retail organization derived from the TPC-DS benchmark (TPC, 2017a), with store, catalog, and Web sales. This section provides some specific details regarding retail contexts that may be useful for practitioners, and that were possibly overlooked in the Adventure Works BDW (section 5.1), since it represents a broader organizational context. The retail BDW data model presented in Figure 5.3 presents several analytical objects (including complementary) in a highly dimensional model, focusing on sales, returns, promotions, customers, items, and warehouses.

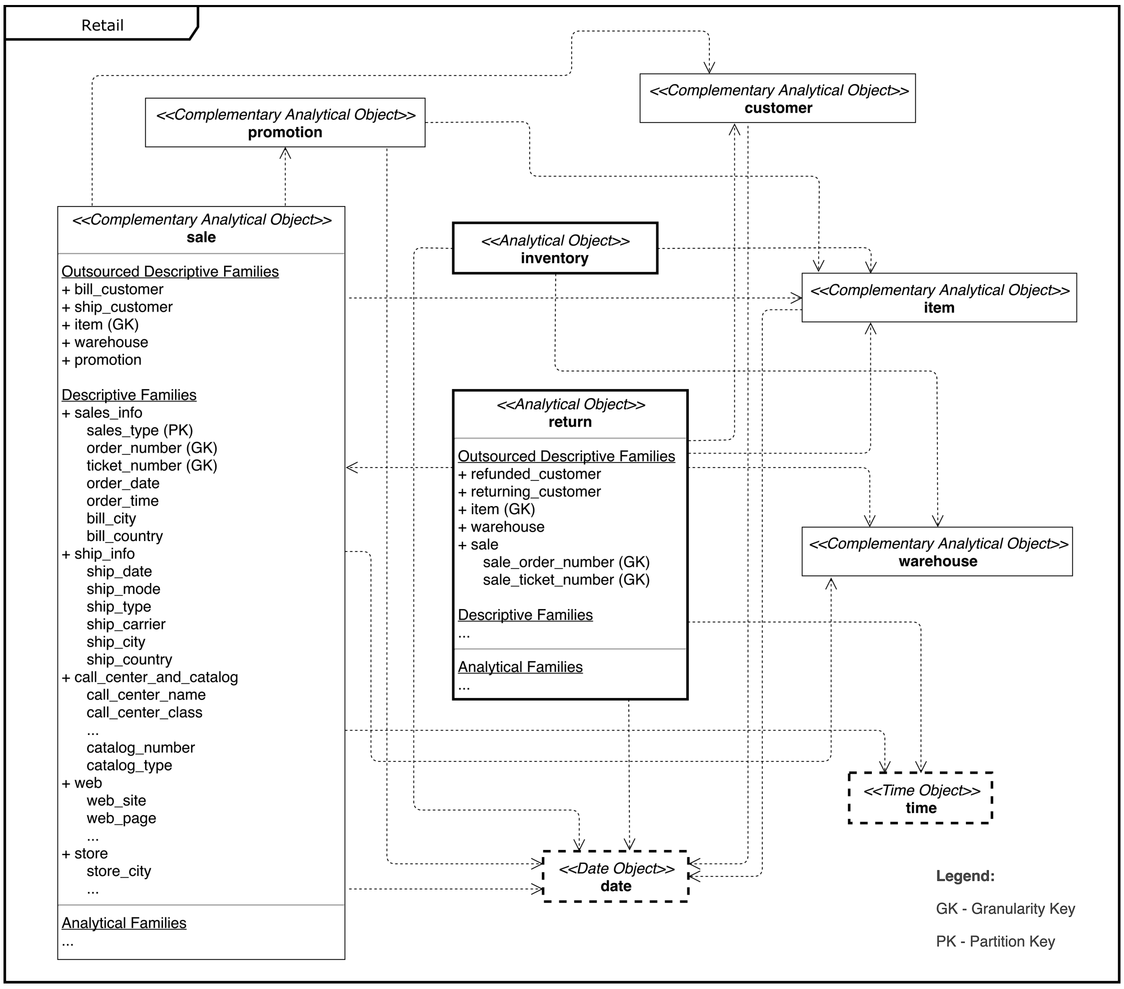


Figure 5.3. Retail BDW data model.

5.3.1 Simpler Data Models: Dynamic Partitioning Schemas

Similarly to the concepts demonstrated in subsection 5.2.4, the retail BDW data model presented in Figure 5.3 also makes use of the partition key to provide simplicity and agility when collecting, preparing, and enriching the data that flows to the BDW. However, considering this example, one does not use the partition key and dynamic partition schemas to simplify batch and streaming analytics in the same analytical object, but rather to provide simpler data models. By making use of different schemas for different partitions, using Hive for example, one can efficiently store what would possibly be three separate analytical objects into just one, i.e., store, catalog, and Web sales into the *"sale"* analytical object partitioned by *"sales type"*. Each partition can have different attributes, which provides a centralized and efficient way of storing each type of sales. This phenomenon also happens for the *"return" object,* as it is almost identical in structure when compared to the *"sale"* object, according to this specific retail context. Furthermore, in this example, *"sale"* is considered as a complementary analytical object, since the *"return"* object includes the granularity key of the "*sale"* object in its descriptive families, due to the fact that a return is related to a *"sale order/ticket number"* and an *"item"*. Such relationship may resemble scenarios in which practitioners use degenerate dimensions for drilling across fact tables, first aggregating the two result sets, as much as possible, and then combining the results, as also discussed in subsection 4.3.2.

5.3.2 Considerations for Spatial Objects

According to the proposed approach, *a priori* designed spatial objects are not mandatory. However, as seen in the previous data models, they are encouraged in predictable scenarios. Considering this retail context, despite the fact that customers have specific addresses, it frequently happens that sales are not billed nor shipped to the default customer address and, therefore, they end up being also attached to the sale itself, not only to the customer. It is possible, and perfectly plausible to include a spatial object (e.g., city) in the data model depicted in Figure 5.3, but, for this example, one shows that it is not mandatory to have one, as one may choose to perform the analysis at the city and country level only, i.e., without other standardized spatial attributes across the BDW (e.g., county, region, and continent), which makes the effort of having to join the *"sale"* or *"customer"* analytical objects with a *"city"* spatial object with more attributes almost useless.

Choosing the adequate attributes that are suitable for the analyses should always be a relevant consideration (Figure 5.3), and it will influence the use of wide spatial objects with several attributes or a few denormalized attributes in the analytical objects. Both possibilities are suitable for this context, but this example only serves the purpose of highlighting that, for specific contexts, spatial objects may not be particularly useful. Furthermore, one aspect that practitioners should take into consideration is to avoid significantly large spatial objects (e.g., denormalized hierarchies ranging from building numbers to country names). In this case, some of the more granular geospatial information can be contained within a descriptive family of the analytical object (e.g., building number and building type), and the less granular information can be stored in the spatial object (e.g., city and country).

5.3.3 Analyzing Non-Existing Events

Considering a traditional DW, if one uses a *"customer"* transactional table to directly load a *"customer"* dimension, the DW will be able to answer queries like the following: *"which customers have not returned a single item?".* However, considering a BDW with a fully denormalized analytical object *"return",* such analysis would not be possible, reason why practitioners have the option of using complementary analytical objects like *"customer".* The same consideration holds true for spatial objects, as one may want to analyze the cities in which the organization did not sell any item. Consequently, for such analytical use cases, practitioners should definitely consider complementary analytical objects, as well as date, time, and spatial objects, since fully denormalized analytical objects only store the events (records) that actually occur.

5.3.4 Wide Descriptive Families

Previously, in subsection 5.3.2, one has highlighted the relevance of adequately choosing the attributes that are relevant for the expected analyses. Such statement does not imply that there is the need to know each query that will be submitted to the system. Nevertheless, frequently, there are certain attributes that are considered as irrelevant for the analytical use cases of the BDW being implemented. In these cases, adequately choosing the attributes allows for smaller descriptive families, which is a relevant aspect when using fully denormalized structures, since, with larger descriptive families, more redundant data would be stored throughout several records, instead of just one or few attributes that allow for join operations with complementary analytical objects.

Taking into consideration the retail context illustrated in this section, the *"store"* descriptive family from the *"sale"* object can theoretically hold a considerable number of attributes. However, certain attributes may be considered as irrelevant depending on the analytical use cases, such as the store's *"GMT offset"* or *"tax percentage",* if the decision-making process of the organization does not consider such information. Consequently, narrow descriptive families should be preferred whenever possible, without sacrificing analytical value. Despite this guideline, if wide descriptive families are mandatory for a specific case, columnar file formats (e.g., ORC and PARQUET) with compression techniques can provide an efficient way of storing analytical objects with hundreds or thousands of columns.

Furthermore, if needed, one can create a *"store performance"* complementary analytical object related to sales, outsourcing the *"store"* descriptive family, as such object would provide significant analytical value at the store level, including several ratios between number of workers, floor space, and sales numbers, for example. The flexibility of the approach regarding dimensional data allows the delegation of some design decisions to practitioners, depending on the intended analysis and data characteristics.

5.3.5 The Need for Joins in Data CPE Workloads

Considering the TPC-DS data model (TPC, 2017a), information like customer demographics, customer household demographics, customer income, and customer address appears related using foreign key relationships between the several dimensions that contain this information and the *"customer"* dimension. In the BDW presented in Figure 5.3, all this information is denormalized into the *"customer"* complementary analytical object. Again, if one needs to answer queries like *"is there any customer demographic class in which the organization does not have any customer?",* this design choice is not appropriate, and the *"customer demographics"* descriptive family inside the *"customer"* object will need to be outsourced to a complementary analytical object. However, one assumes that this is not the case in this retail context.

Considering this denormalization process, with a *"customer"* complementary analytical object that includes demographics, household, address, and income information, at first glance, one may find the data CPE process to be somehow simpler than maintaining several separate dimensions, which, in fact, can be partially true. However, the degree of simplicity depends on the transactional source that fuels the *"customer"* object:

- If the transactional source is a relational database in which this information comes from several tables, then the data CPE workload corresponding to the loading and refreshment of the *"customer"* object will need to perform several joins to provide a fully denormalized structure;
- In contrast, considering the large-scale retail scenarios using NoSQL databases to support the vast amount of transactions being generated, this data may arrive at the BDW already denormalized (e.g., column-oriented and document-oriented NoSQL databases), representing the opposite situation and providing a high degree of simplicity without the need to perform join operations, which considerably simplifies the data CPE workload.

5.4 Code Version Control System

The software industry is under constant evolution, and open source or subscription-based remote version control systems like GitHub have been a core pillar of current software management and dissemination. GitHub is one of the main platforms for collaboration in software projects, whose activity has the potential to generate vast amounts of data. In this section, one explores the GitHub public dataset available on Google BigQuery (Google, 2018) regarding 2.9 million public software repositories, in order to model a BDW that supports the decision-making process regarding the activity and metrics of these repositories' commits and content in large-scale environments. The BDW data model illustrated in Figure 5.4 includes the *"commit"* and *"repository"* analytical objects, being the latter a complement to the first, and it also includes the date and time objects.

The *"commit"* analytical object stores data regarding the commits that have been made to the several repositories, including information regarding the author and the committer. This analytical object does not contain any relevant analytical attribute and, therefore, count operations will be the primary focus of analysis. The *"repository"* complementary analytical object stores information regarding the current state of the 2.9 million public repositories, including the license, an array containing the information of several files for each branch, an array containing the code (in bytes) of each programming language in the repository, and the number of issues classified by type (possibly extracted by scrapping and mining the text from the issues page of each repository, for example).

Both the *"commit"* and the *"repository"* objects can be implemented as streaming analytical objects, in which they are updated as soon as each commit or any other file activity takes place. However, due to the chosen data model, the streaming implementation may differ, as the *"commit"* object is

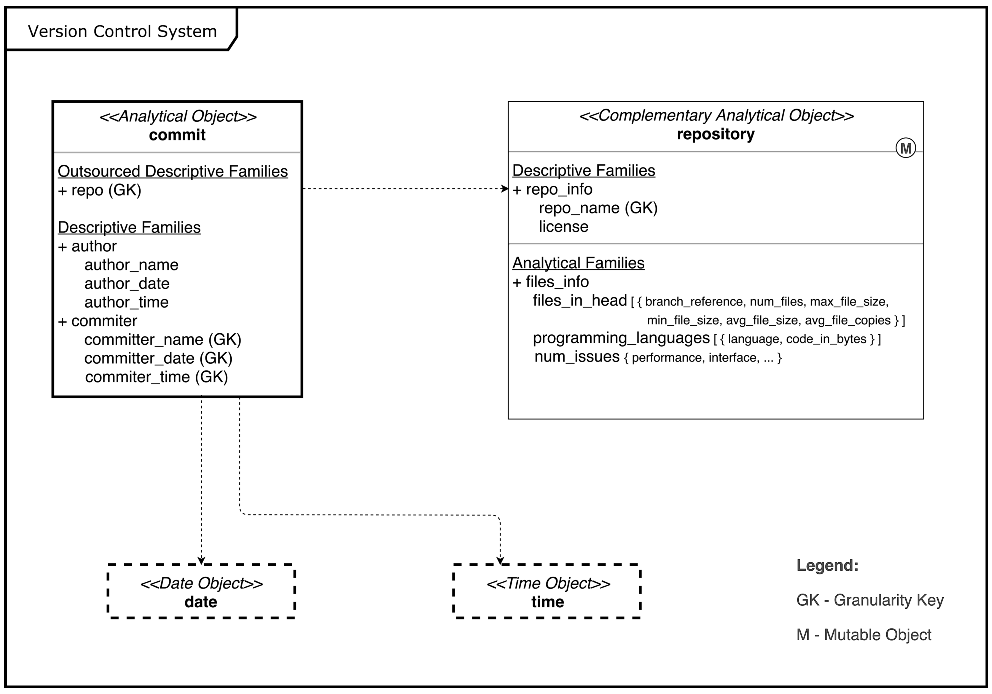


Figure 5.4. BDW data model for code version control systems.

an immutable append-only object, in which each commit originates a new record, while the *"repository"* object is a mutable object, because the nested analytical attributes should be updated (e.g., code in bytes and number of files) instead of originating a new record. Consequently, the *"repository"* object can be implemented using a NoSQL database with adequate support for fast random-access to nested objects or, depending on the specific implementation details (e.g., update frequency, latency requirements, and update throughput), as already seen, Hive transaction tables can also be an option.

5.5 A Global Database of Society – The GDELT Project

The GDELT project makes available an open database that monitors worldwide broadcast, print, and Web news, identifying the people, locations, organizations, topics, sources, emotions, among many other information regarding news (GDELT, 2018). The data model presented in Figure 5.5 represents a BDW to support decision-making processes using worldwide event data from the GDELT project, which is composed by date and time objects, a *"city"* spatial object (including denormalized data regarding the countries corresponding to the cities), and an *"event"* analytical object. This analytical object is responsible for storing news/events, with data regarding the event and the actors involved

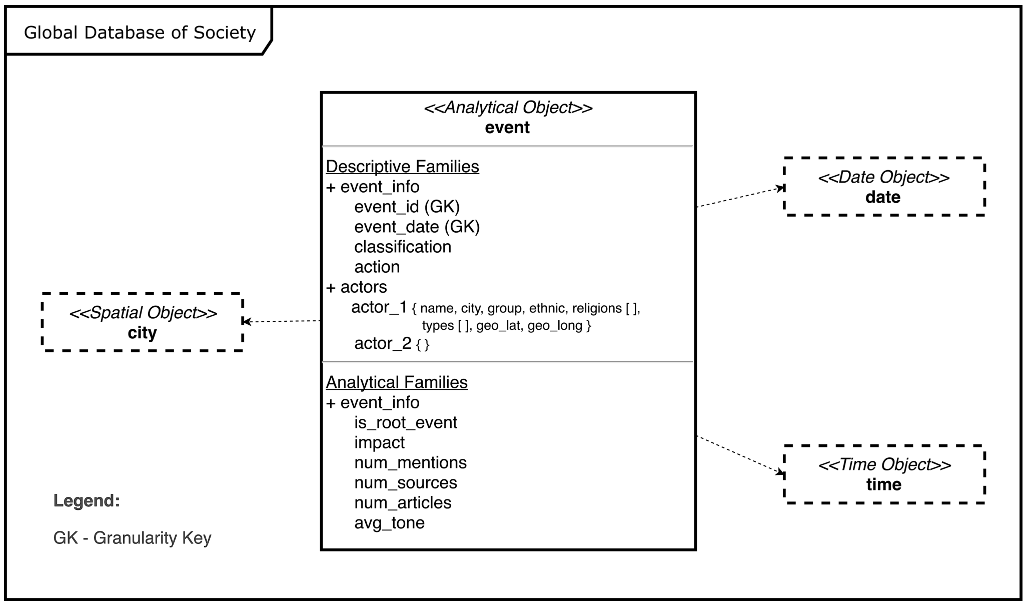


Figure 5.5. BDW data model for the GDELT project.

in it. The actors' data regarding name, city (attribute related to the spatial object *"city")*, group (e.g., United Nations or World Bank), ethnic and religion information, geocoordinates, among others, is stored in a complex data type (e.g., Row or Struct) for organization purposes, which can also contain other complex data types (e.g., *"religions" and "types" arrays*). Consequently, the *"event"* object allows for several analytical applications to process and analyze worldwide news/events.

5.6 Air Quality

The final BDW example of this chapter is focused on air quality analysis through sensors spread across different locations. The example presented in this section is based on the open air quality platform (OpenAQ, 2018). The BDW data model depicted in Figure 5.6 integrates a spatial object *"city"*, date and time objects, and a *"measurement"* analytical object corresponding to the measured value of a specific parameter from a specific location, date, and time. The *"measurement"* analytical object has geospatial coordinates which are not present in the spatial object. This is a design choice that is always encouraged, due to the high cardinality of geospatial coordinates. Consequently, space is broken down into levels of detail, and the lower levels are typically stored in spatial objects, while the higher levels of detail are stored in the analytical object, as already explored in subsection 5.3.2.

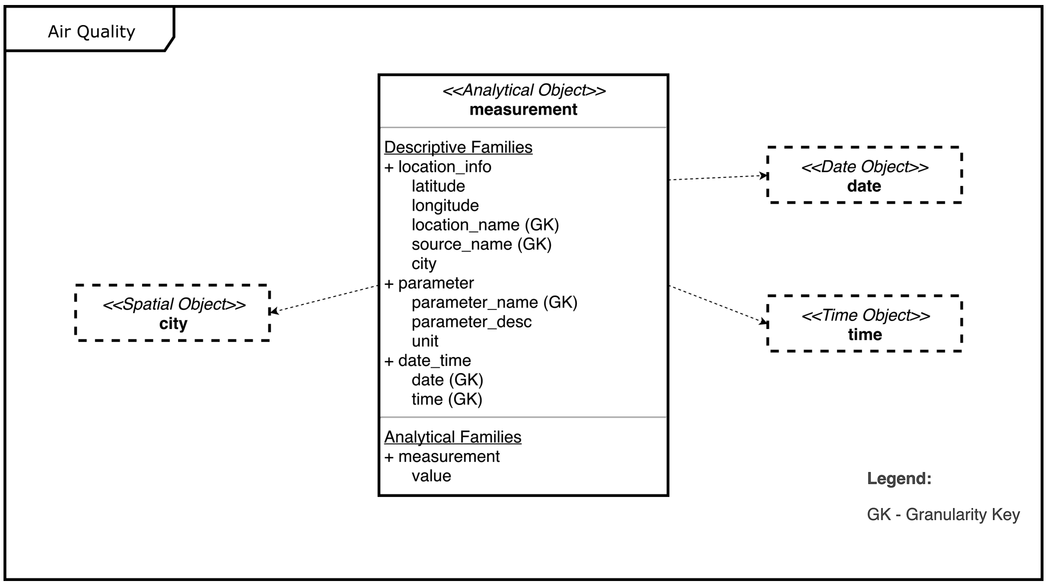


Figure 5.6. BDW data model for air quality analysis.

Real-time aggregations on sensor data are a really adequate use case for specific technologies like Druid (Correia et al., 2018), a columnar storage that provides aggregations and indexing at ingestion time. Such design and implementation choice can fuel a *“measurement”* analytical object modelled at a higher level of detail, as, for example, the *“value”* attribute can be an average of each minute, instead of the raw sensor readings produced each second. Besides the use of Druid, this scenario can also be supported by a Spark Streaming CPE workload using window operations or micro batch aggregations, for example, storing the resulting data in the streaming storage system of the BDW.

Nevertheless, when using Druid (or similar technologies), one should pay attention to the specificities of the data models that these technologies require, because, for example, Druid currently handles descriptive and analytical attributes in fully denormalized structures, which does not completely correspond to the data model presented in Figure 5.6. However, as seen in this work, the proposed approach for BDWing is relatively flexible and, if that is the case, practitioners can adopt a fully denormalized *“measurement”* analytical object without spatial, date, or time objects. After these considerations, this section can be seen as a collection of insights that practitioners can use to design streaming analytics on sensor data, specifically for air quality analysis in this case, but with further applications for other sensor-based analytical workloads.

This page was intentionally left blank

Chapter 6. Fueling Analytical Objects in Big Data Warehouses

One of the most laborious stages in the implementation of DWs, whether they are traditional or oriented for Big Data environments, is the development of ETL processes. As seen in the previous chapter, one does not use this terminology, in order to avoid confusion between ETL and ELT processes, which can cause several unnecessary discussions. Thus, this approach prefers the friendlier NIST terminology (collection and preparation), extending it with the term "enrichment", due to the relevance of derived attributes (feature engineering) for more impactful and actionable insights. As previously discussed, in the proposed approach, these processes are known as CPE processes/workloads. This chapter presents several examples of relevant CPE workloads that practitioners may find useful when implementing BDWs. In these examples, Spark and Talend Open Studio for Big Data are used for demonstration purposes. Designing and developing CPE workloads for BDWs can be considered one of the most time-consuming and difficult tasks in BDWing. For that reason, structuring several examples that demonstrate typical tasks in these environments is seen as a relevant contribution, mainly to the practitioners' community. Such examples are part of the demonstration activity in the DSRM for IS methodology used in this research process.

6.1 From Traditional Data Warehouses

Migrations from traditional DWs to BDWs will typically become more common (Russom, 2016). In BDWing implementations, one of the potential workloads will be the migration of the organization's current relational DW to a BDW. This section presents how this task can be achieved using Sqoop, HDFS, Spark, and Hive, four technologies depicted in Figure 4.3. In fact, the guidelines here provided are also useful for CPE workloads that read data from relational OLTP databases to fuel the BDW. Sqoop is used to transfer data from relational databases to HDFS, and Spark is used to prepare and enrich the data before storing it in the batch storage component (Hive in this example).

Figure 6.1 illustrates the data flow between the components of a possible technological infrastructure. The first step of this process consists in transferring the data from the RDBMS that currently supports the DW to HDFS. This task can be done using Sqoop's import functionality:

```
sqoop import --connect <db_connection_string> {authentication_details} --table
<table_name> --target-dir <path_to_data_folder>
```

In this example, the data from a traditional sales DW modelled according to the SSB benchmark is used, containing one fact table *("lineorder")* and four dimensions: "*customer", "supplier", "part",* and *"date dim"* (O'Neil, O'Neil, & Chen, 2009). After transferring the data and storing it in the distributed file system (HDFS), one can start the preparation and enrichment of this data according to the desired analytical object. In this case, the analytical object is a fully denormalized structure containing all the resulting attributes from the join between the fact table and each dimension, despite the fact that, as seen in section 4.3, analytical objects represent flexible and efficient structures that can be more than just a full denormalization of fact tables. The following Spark 2 Python code illustrates a typical script to perform this task:

1. Import Spark packages and classes.

```
from pyspark.sql import SparkSession, Row
from pyspark.sql.types import *
```

2. Define two variables: "*hdfsPath"* and "*hiveDbName".*

```
hdfsPath = "hdfs://<servername>:8020/<path_to_data_folder>/"
hiveDbName = "ssb"
```

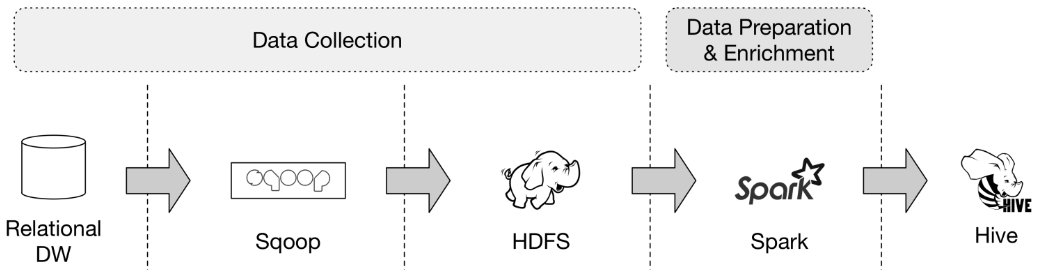


Figure 6.1. CPE workload for traditional DW migration.

3. Create Spark session.

```
spark = SparkSession \
        .builder \
        .appName("Create SSB Analytical Object") \
        .config("spark.sql.warehouse.dir", "/apps/hive/warehouse/") \
        .enableHiveSupport() \
        .getOrCreate()
```

4. Create the Hive database for the BDW.

```
spark.sql("DROP DATABASE IF EXISTS " + hiveDbName + " CASCADE")
spark.sql("CREATE DATABASE " + hiveDbName)
```

5. Create a Spark DataFrame and a Spark Temporary View for each table imported from Sqoop. This will allow the execution of SQL-based instructions on top of the data that has been stored on HDFS.

```
...
dfSchema = StructType([
            StructField("custkey", IntegerType(), True),
            StructField("name", StringType(), True),
            StructField("address", StringType(), True),
            StructField("city", StringType(), True),
            StructField("nation", StringType(), True),
            StructField("region", StringType(), True),
            StructField("phone", StringType(), True),
            StructField("mktsegment", StringType(), True)])
customerDF = spark.read \
             .csv(hdfsPath + "customer", header=False, schema=dfSchema)
customerDF.createGlobalTempView("customer")
...
```

6. Create the Hive table to store the analytical object. In this example, the table uses the ORC file format, which is an optimized columnar format that considerably improves Hive's performance (Huai et al., 2014). The Parquet file format can also be used for Hive tables, in order to achieve adequate performance (Parquet, 2018).

```
spark.sql("CREATE TABLE ssb.analytical_obj (c_custkey int, c_name varchar(25), ...)
STORED AS ORC")
```

7. Join the five tables (one fact table and four dimensions) and store the result in the previously created Hive table. If the Hive table is partitioned, the insert statement should reflect the partition scheme, and the adequate HiveQL constructs should be used. This example illustrates a table without partitions.

```
spark.sql("INSERT INTO ssb.analytical_obj SELECT ... FROM global_temp.lineorder LEFT
OUTER JOIN global_temp.customer ON ...")
```

8. Depending on the total size of the resulting table and the number of partitions in the DataFrame, Spark can generate several small ORC files, which can interfere with the performance and adequate operation of Hive and Hadoop. Consequently, the following Hive Data Definition Language (DDL) statement may be useful, in order to concatenate these small ORC files into larger ones. Practitioners may consider this statement in their CPE workloads. Note: there are other ways of manipulating the number of output files, including some Spark configurations and functions (e.g., coalesce and repartition).

```
ALTER TABLE ssb.analytical_obj CONCATENATE
```

9. Finally, it is relevant to highlight the need to assure that after every CPE workload, the table and column statistics in Hive are adequately computed and refreshed, taking the maximum advantage of this query optimization mechanism. Therefore, the following Hive DDL is also significantly relevant in these scenarios.

```
ANALYZE TABLE ssb.analytical_obj COMPUTE STATISTICS
ANALYZE TABLE ssb.analytical_obj COMPUTE STATISTICS FOR COLUMNS
```

6.2 From OLTP NoSQL Databases

In Big Data environments, NoSQL databases are typically the main driver for OLTP workloads, assuring adequate scalability in intensive random access scenarios (Cattell, 2011). Organizations are currently using NoSQL databases for several applications, for example: massive online sales services (e.g., Amazon); IoT applications; search engines (e.g., Google); and mobile applications.

This section presents a workload to collect, prepare and enrich data from Cassandra, which is used to store millions of records from sensors. The sensors send a record to Cassandra every 15 minutes, including the following attributes: "*sensor id"; "date" - a timestamp containing the date and time of the record; "building id"* – the building in which the sensor is located; "*kwh"* – the energy consumption recorded at that moment. The goal of this workload is to collect Cassandra's data for a specific month and store it in the BDW's batch storage (Hive). The Hive analytical object used for this purpose will be partitioned by year and month. Throughout the workload, the data will be aggregated to match an hourly aggregation level, instead of the original "quarter of an hour" aggregation level. This workload can be coded using the following Spark Java code:

1. Import Spark packages and classes. In this example, one will use the DataStax open source Spark Cassandra connector.

```
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.SaveMode;
import org.apache.spark.sql.SparkSession;
import static org.apache.spark.sql.functions.col;
...
```

2. Create the main Java class and method that will be used to include the several tasks for this workload. As already known, the first task is the definition of the Spark Session.

```
SparkSession spark = SparkSession
    .builder()
    .appName("Read sensor records from Cassandra")
    .config("spark.cassandra.connection.host", <hostname(s)>)
    .config("spark.cassandra.auth.username", <username>)
    .config("spark.cassandra.auth.password", "<password>")
    .config("spark.sql.warehouse.dir", "/apps/hive/warehouse/")
    .config("hive.exec.dynamic.partition.mode", "nonstrict")
    .enableHiveSupport()
    .getOrCreate();
```

3. Create a Spark Dataset that reads data from the Cassandra table. Spark Datasets are an abstraction introduced in Spark 1.6, which combines the benefits of Spark DataFrames (Spark SQL's optimized execution engine) with the benefits of Resilient Distributed Datasets (RDDs), namely strong typing and the ability to use powerful lambda functions (Spark, 2017).

```
Dataset<Row> ds = spark.read().format("org.apache.spark.sql.cassandra")
   .option("keyspace", <keyspace_name>)
   .option("table", <table_name>)
   .load();
```

4. Filter the Dataset to select a specific month (January), and aggregate the Dataset to match an hourly aggregation level.

```
Dataset<Row> dsFiltered = ds.filter(month(col("date")).equalTo(1));

Dataset<Row> dsGrouped = dsFiltered
   .groupBy(
      col("sensor_id"),
      date_format(col("date"), "YYYY-MM-DD HH:00:00").as("moment"),
      col("building_id"))
   .sum("kwh");
```

5. Store the Dataset into the corresponding Hive table and partition. Since dynamic partitioning is enabled in the Spark Session configurations, Spark will figure out the partitioning scheme automatically. One needs to be aware that using the method presented below, the columns of the Dataset must be ordered according to the columns of the Hive table, being the partitioning columns the last ones. Similarly to the previous section, after this task, one can concatenate small files and recompute table and column statistics.

```
dsGrouped.select(
      col("sensor_id"),
      col("moment"),
      col("building_id"),
      col("sum(kwh)"),
      year(col("moment")),
      month(col("moment")))
   .write().mode(SaveMode.Overwrite).insertInto(<hive_database.table>);
```

6.3 From Semi-Structured Data Sources

The variety of data is one of the major characteristics for defining Big Data. As already highlighted, data may be more or less structured depending on the underlying source. The previous CPE workloads focused on relatively structured data, namely relational and column-oriented schemas. In this section, the focus is on semi-structured data sources, which can typically produce data in formats that are not completely detached from a schema, but are significantly flexible or nested, such as server logs, JSON, or XML files, for example.

Take as an example the following GeoJSON file, which is basically a JSON file that, among other attributes, holds geospatial information about buildings in Lisbon:

```
"features": [
   {"type": "Feature",
    "properties": {
       "Shape_Leng": 68.663877,
       "Shape_Area": 276.535056,
       "L_HtRf": 21,
       "Building_Occupation": 3, ...
    },
    "geometry": {
       "type": "MultiPolygon",
       "coordinates": [ [ [
          [ -9.095283006673773, 38.75460513863176, 0.0 ],
          [ -9.095298222128497, 38.754405797462653, 0.0 ], ...] ] ]
    }
   }, ...
```

In the proposed approach, one highlights the use of analytical objects that can contain nested structures. Extracting useful attributes for analysis and implementing an analytical object that adequately deals with semi-structured data is the key in this specific scenario. In order to handle semi-structured data, one needs to assure two relevant aspects: the technology used to implement the CPE workload must be able to process these data structures; and the technology used to store the results of the workload should also be able to handle semi-structured data. Regarding the first aspect, in this section, Talend Open Studio for Big Data is used to build the CPE workload. However, there are many other technologies that are suitable for this purpose (Spark inclusively). Regarding the second aspect, Hive is used again as the batch storage of the BDW, since it can adequately handle flexible and nested data structures like arrays and maps, not only providing ways of storing them, but also providing ways of querying and performing analytics on these structures.

Figure 6.2 presents a Talend job used to collect the aforementioned GeoJSON file from HDFS, preparing and enriching it with supplementary GeoJSON files. This job is responsible for fueling a previously created analytical object storing several buildings indicators in Lisbon, including not only their geospatial information, but also their associated services (e.g., gyms and restaurants), occupation, and construction characteristics, for example.

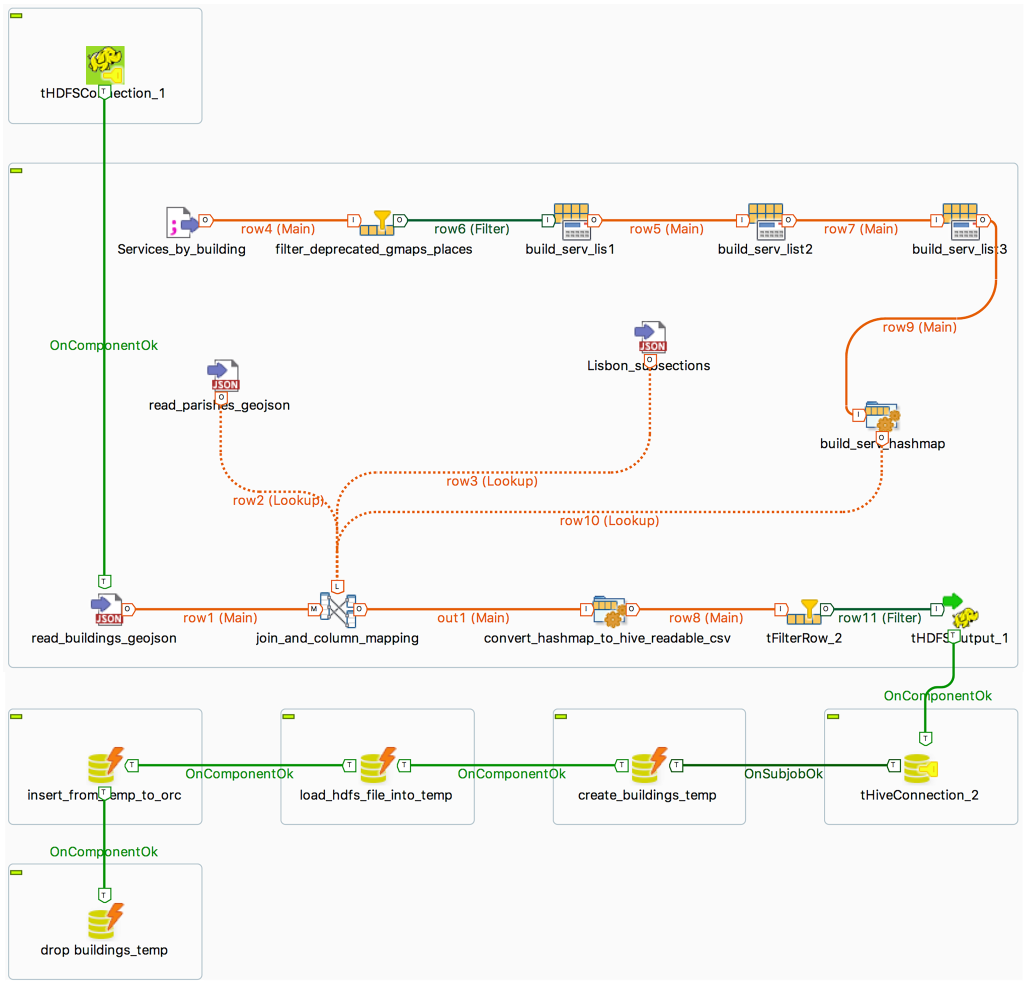


Figure 6.2. CPE workload for semi-structured data.

This job starts by reading the content of the GeoJSON file, and Talend Open Studio for Big Data is able to automatically deduce its schema by inspecting a sample of the records within the file. Then, one is able to join the buildings file with other supplementary files, like parishes, neighborhoods (subsections), and services inside the building or near it. The service list can be extracted from the Google Maps Application Programming Interface (API), and afterwards, one can use several Talend components (e.g., list aggregations and custom java code) to create a list of services associated with each building and make it available in the appropriate format.

Finally, after all the previous tasks are completed, the data is sent to HDFS and a temporary Hive table is created to store that data in text format. As previously highlighted, Hive tables in ORC or Parquet format are more suitable for analytical purposes, so one needs to move the data from this temporary table to the table using the ORC format. This procedure of using a temporary table is common in Hive-based DWs. However, practitioners may find other ways to directly move the data to ORC tables without the need for a temporary table, for example, using the ORC API. The final result is an analytical object stored as a Hive table, which is able to handle a variety of structures, including arrays and maps.

6.4 From Streaming Data Sources

Until now, only the use of the BDW's batch storage component was demonstrated. When a source generates data through streaming mechanisms, one needs to rely on the streaming storage of the BDW. Data velocity is another relevant characteristic in Big Data environments, and in this section, one will be discussing the development of a streaming CPE workload to fuel the BDW. Kafka is used for data collection, Spark Streaming is used for data preparation and enrichment, and Cassandra is used as the NoSQL database responsible for the BDW's streaming storage. Figure 6.3 summarizes this CPE workload.

1. The first step is the development of a Kafka producer. In this example, this producer generates a record each five seconds, corresponding to a random product sale in a simulated e-commerce environment. Each record contains a key (*"sales id"*) and a value (*"URL"* of the

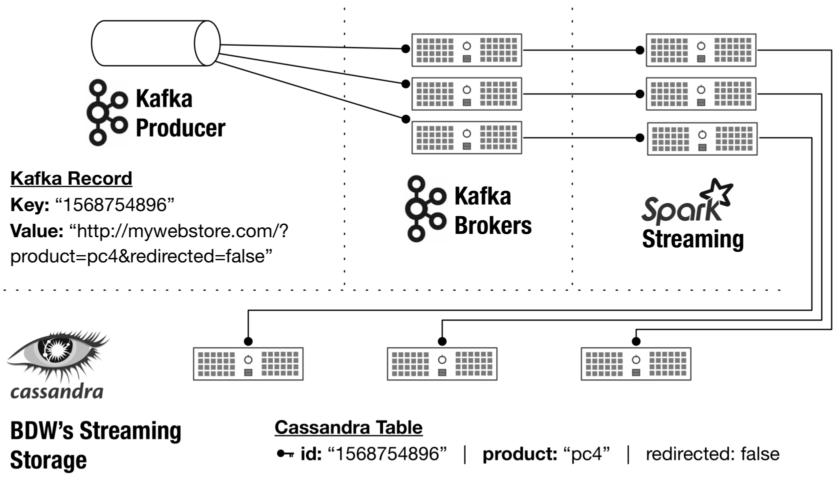


Figure 6.3. Streaming CPE workload using Kafka, Spark Streaming, and Cassandra.

Web page wherein the product was purchased). The following Java code snippets demonstrate this scenario, and can be used as a guide for other Kafka producers.

a. Import the Java packages and classes. For this producer, the Apache Kafka API is used.

```
...
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.clients.CommonClientConfigs;
import org.apache.kafka.clients.producer.Callback;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.RecordMetadata;
```

b. Create the Java class and its variables. Generally, this class needs the topic in which the producer will publish the records, the producer object, and several properties that reflect the infrastructure in use and the application requirements (e.g., secure/unsecure cluster and the number of acknowledgments to consider a request as being completed). In this example, there is also one variable containing random products used to generate random online sales URLs.

```
public class DummyProducer extends Thread {
    private final String topic;
    private final KafkaProducer<String, String> producer;
    private final Properties props;
    private final String[] products;
...
```

c. Create the constructor that initializes the variables enumerated above.

```
public DummyProducer(String topic, String kafkaServerUrl, int kafkaServerPort)
{
   this.products = new String[] {
      “smartphonex7”, “pc4”, “keyboardy”, “monitorpro”
   };

   this.props = new Properties();
   this.props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
      kafkaServerUrl + “:” + kafkaServerPort);
   this.props.put(ProducerConfig.CLIENT_ID_CONFIG, “DummyProducer”);
   this.props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG,
      “SASL_PLAINTEXT”);
   this.props.put(ProducerConfig.ACKS_CONFIG, “all”);
   this.props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
      “org.apache.kafka.common.serialization.StringSerializer”);
   this.props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
      “org.apache.kafka.common.serialization.StringSerializer”);

   this.producer = new KafkaProducer<>(props);
   this.topic = topic;
}
```

d. Create the run method that is responsible for the execution of the main task, i.e., generating an URL of a random online sale each five seconds for an infinite period of time. In this example, a random product is selected between a set of four available products (see code snippet above). Each sale also contains a random flag indicating if the sale was the result of a recommendation based on a previous visualized product (*“redirected”* attribute).

```
@Override
public void run() {
    Random random = new Random();
    String message;

    while(true) {
       String salesID = “” + random.nextInt() + System.currentTimeMillis();
       message = String.format(
          “\”http://mywebstore.com/?product=%s&redirected=%s\””,
          this.products[random.nextInt(4)], random.nextBoolean()
       );

       ProducerRecord<String, String> data = new ProducerRecord<>(
          this.topic, salesID, message
       );

       this.producer.send(data);
       try {
          Thread.sleep(5000);
       } catch (InterruptedException ex) {
          System.err.println(ex.getMessage());
       }
    }
}
```

e. Finally, create the main method of the *“DummyProducer”* class, which will simply run the Kafka producer given a Kafka topic, broker, and port.

```
public static void main(String args[]) {
   DummyProducer producer = new DummyProducer(<topic>, <kafka_broker>, <port>);
   producer.start();
}
```

2. Having a streaming producer is just part of the CPE workload, namely it represents the collection step of the workload. Consequently, in BDWing environments, one typically needs to prepare and enrich the data before making it available for analytical purposes. One way of achieving this goal is to use the powerful and stable Spark Streaming API, which allows the relatively easy use of multiple functions (e.g., filter, join, count, and map) on streaming sources like Kafka, assuring adequate scalability and fault-tolerance. The following Java code snippets demonstrate a Spark Streaming application that uses regular expressions to extract information from the Kafka messages and to store the results in the BDW’s streaming storage

component. In this example, Cassandra is used to store a streaming analytical object containing the *"sales id"*, the *"product",* and the *"redirected"* attributes.

a. Import the required packages for this Spark Streaming application. The crucial APIs are the Spark Core API, the Spark Streaming API, the Spark Streaming Kafka API, and the DataStax Cassandra connector.

```
...
import static com.datastax.spark.connector.japi.CassandraJavaUtil.
javaFunctions;
import static com.datastax.spark.connector.japi.CassandraJavaUtil.mapToRow;
import org.apache.spark.SparkConf;
import org.apache.spark.streaming.api.java.*;
import org.apache.spark.streaming.kafka010.*;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.common.serialization.StringDeserializer;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.streaming.Durations;
```

b. After creating the main class and the main method of the Spark Streaming application, configure the Kafka connector appropriately, including the consumer configurations and the list of topics to consume. The following code snippet illustrates the configuration for a Kerberized cluster, in which the Spark consumer informs Kafka when it has finished consuming a certain offset, that is why one disables Kafka auto commits and uses the *"offsetRanges"* variable. This assures that the Spark consumer only acknowledges the processing of certain offsets when the records were already stored in Cassandra. One will clarify this functionality later in this subsection.

```
Map<String, Object> kafkaParams = new HashMap<>();
   kafkaParams.put("bootstrap.servers", <kafka_broker>:<port>);
   kafkaParams.put("key.deserializer", StringDeserializer.class);
   kafkaParams.put("value.deserializer", StringDeserializer.class);
   kafkaParams.put("group.id", "spark.events");
   kafkaParams.put("auto.offset.reset", "latest");
   kafkaParams.put("enable.auto.commit", false);
   kafkaParams.put(“security.protocol”, “SASL_PLAINTEXT”);

   Collection<String> topics = Arrays.asList(<topic(s)>);
   final AtomicReference<OffsetRange[]> offsetRanges = new AtomicReference<>();
```

c. Create the Spark configuration and the Spark Streaming object. Since one is storing the results in Cassandra, the Cassandra connection properties are also needed, similarly to the previous OLTP NoSQL-based CPE workload. In this example, the streaming application processes data arriving from Kafka in 10 seconds micro batch intervals. As highlighted in subsection 4.1.3.2, micro batches are configurable, and they are often a trade-off between latency, throughput, and flexibility.

```
SparkConf conf = new SparkConf()
   .setAppName(“StreamingCPEWorkload”)
   .set(“spark.cassandra.connection.host”, <host(s)>)
   .set("spark.cassandra.auth.username", <username>)
   .set("spark.cassandra.auth.password", <password>);

JavaStreamingContext jssc = new JavaStreamingContext(
   conf, Durations.seconds(10)
);

JavaInputDStream<ConsumerRecord<Integer, String>> stream =
   KafkaUtils.createDirectStream(
      jssc,
      LocationStrategies.PreferConsistent(),
      ConsumerStrategies.Subscribe(topics, kafkaParams)
);
```

d. As previously stated, this application is using manual Kafka commits and, therefore, one needs to inform Kafka when the data has been processed. Consequently, the first step after creating the stream is to store the Kafka offset range in the Spark application, in order to commit the offsets already processed after the data has been successfully stored in Cassandra. The following function needs to be the first function called after the creation of the stream, since it does not work after the application of transformations to the *“stream”* object. This assures that the application has "exactly-once" semantics instead of "at-least-once" semantics, which assures that data arriving from Kafka does not get processed and stored twice.

```
stream.foreachRDD((JavaRDD<ConsumerRecord<Integer, String>> rdd) -> {
   OffsetRange[] offsets = ((HasOffsetRanges) rdd.rdd()).offsetRanges();
   offsetRanges.set(offsets);
});
```

e. To extract the *"product"* and *"redirected"* attributes arriving from Kafka's messages, one can use regular expressions applied to the URL. The "map" transformation can be used to extract these attributes. Spark Streaming offers several transformations, window, join, and output functions that can be used for streaming contexts. For example, joining a stream with an historical dataset can be significantly useful for BDWing purposes, in order to prepare and enrich data for certain analytical objects.

```
JavaDStream<DummySale> transformedStream = stream
   .map((ConsumerRecord<Integer, String> event) -> {
      String[] fields = event.value().split("\";\"");
      Matcher m = Pattern
         .compile("product=(.*)&redirected=(.*)").matcher(fields[1]);
      m.find();
      return new DummySale(
         fields[0], m.group(1), Boolean.parseBoolean(m.group(2))
      );
});
```

f. Since all the processing tasks for this example are already completed, the results can be stored in Cassandra, and the Spark Streaming application can then commit the offsets to Kafka, acknowledging that it already processed and stored that specific records. It should be noted that, in this example, *"DummySale"* is a typical Java Bean containing the same attributes as the analytical object stored in Cassandra. It is used to apply a schema to each row in the Spark RDD.

```
transformedStream.foreachRDD((JavaRDD<DummySale> rdd) -> {
   javaFunctions(rdd).writerBuilder(
      <topic>, <cassandra_database>, mapToRow(DummySale.class)
```

```
    ).saveToCassandra();
    ((CanCommitOffsets) stream.inputDStream()).commitAsync(offsetRanges.get());
});
```

g. Finally, the last task consists in simply starting the application and waiting for its termination.

```
try {
    jssc.start();
    jssc.awaitTermination();
} catch (InterruptedException ex) {
    System.err.printf("The application '%s' has stopped! ", conf.getAppId());
}
```

6.5 Using Data Science Models

One of the main aspects in the proposed approach (previously described in Chapter 4) is the inclusion of data science models in CPE workloads fueling the BDW. This work considers data science as an umbrella for several related and more specific subareas, including: data mining/machine learning; text mining; image mining; and video mining. Regarding data mining, traditional DWs are frequently considered a relevant data source for the algorithms used in this area, since they typically contain an extensive record of historical data regarding the organization. Since these algorithms need a vast training set to extract patterns, traditional DWs are natural sources of data for feeding these algorithms, and can be considered "clients of the DW" (Kimball & Ross, 2013). This is also true for a BDW (Figure 4.1), wherein it can be queried by data scientists that are "playing" with the data in the data science sandbox. However, this work extends this ideology by inviting practitioners to include data mining/machine learning algorithms in CPE workloads, in order to create new predictive attributes and include them in the analytical objects stored in the BDW (Figure 4.2).

The same strategy stays valid for unstructured data science techniques like text mining, image mining, or video mining. These techniques are not frequently seen in traditional DWing environments. One can argue that raw unstructured data holds almost no value for analytical purposes. Patterns should be extracted using data science techniques and then, since these results are already relatively structured, they can follow their path to an OLAP-oriented system like the BDW. Another argument that can be made is that the rigid structure of relational DWs can be seen as a significant barrier in these scenarios, since most of the time it becomes unnatural, time-consuming, and inefficient to model dimensions, fact tables, and relationships for this type of analytical workloads.

For example, when an organization is collecting images in real-time and instantaneously using an algorithm to predict an occurrence of a certain pattern in that image (e.g., template matching for manufacturing quality control), it becomes really inefficient to fuel a relational DW via streaming mechanisms. For each image being analyzed, a typical ETL process has to scan the several dimensions for any changes since the last DW refresh (e.g., new rows to add/update in dimension tables), or to retrieve each dimension's surrogate key for matching the foreign key of each new row in the fact table, for example. In these contexts, relational DWs are not the most adequate solution, and fully denormalized structures (analytical objects), are arguably more efficient and simpler to implement, since their corresponding CPE workloads are considerably easier to develop and maintain when compared to traditional ETL processes.

In Big Data environments, there is the need to integrate both structured and unstructured sources (Kimball & Ross, 2013). As previously discussed, predictive analytics is also a relevant use case that BDWs must consider among their set of mixed and complex analytical workloads. This is the reason why including structured and unstructured data science models in the CPE workloads can be seen as a way of extracting the value hidden in Big Data, which can then be used to make predictions of future events and to fuel the analytical objects stored in the BDW. This section discusses two types of CPE workloads including data science models, using data mining/machine learning models for structured data and using text mining, image mining, and video mining models for unstructured data.

6.5.1 Data Mining/Machine Learning Models for Structured Data

Predictive attributes are the key for predictive analytics inside BDWs. One highlights the use of data science models to create these attributes. The data stored either in the file system or in the indexed storage of the BDW can be used to train these predictive models, which can then be used to enrich data arriving at the system with new predictive attributes. In these contexts, data mining/machine learning models can be significantly useful for CPE workloads dealing with structured data. This subsection uses the Spark MLlib API to demonstrate one of many data mining techniques that can be used in BDWing systems, namely clustering.

Clustering can be used when the training dataset is not previously labeled with the attribute one wants to predict, which can also be mentioned as unsupervised learning. There are many other techniques available in Spark MLlib, either unsupervised (e.g., association rules) or supervised (e.g., classification and regression), assuring a scalable way of training, testing, and applying data mining/machine learning models. The following Java code snippets demonstrate the use of clustering algorithms in Spark, namely the very broadly used K-means algorithm. Figure 6.4 presents an overview of the CPE workload being implemented in this subsection.

1. Import the java packages needed for the application.

```
...
import org.apache.spark.sql.SparkSession;
import org.apache.spark.ml.clustering.KMeansModel;
import org.apache.spark.ml.clustering.KMeans;
import org.apache.spark.ml.feature.MinMaxScaler;
import org.apache.spark.ml.feature.MinMaxScalerModel;
import org.apache.spark.ml.feature.VectorAssembler;
import org.apache.spark.ml.linalg.Vector;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Encoders;
import org.apache.spark.sql.Row;
```

2. Create the main class and the main method, which starts by initiating the Spark Session. In this example, one will be segmenting customers according to their buying behavior (following the example context of section 6.1): how many orders do they place? How much revenue do they generate to the company? Are they regular monthly customers?

```
SparkSession spark = SparkSession
    .builder()
    .appName(“Segmenting Customers using K-means”)
    .config(“spark.sql.warehouse.dir”, “/apps/hive/warehouse/”)
    .config(“hive.exec.dynamic.partition.mode”, “nonstrict”)
```

```
.enableHiveSupport()
.getOrCreate();
```

3. To accomplish this goal, the analytical object created in subsection 6.1 can be used, which, as already seen, is based on the SSB dataset (O'Neil et al., 2009). Since the analytical object corresponds to data that is already stored in the BDW, this workload does not include a data collection stage from an external data provider. This represents a workload that uses the models, insights, and results derived from the data science sandbox component of the BDW and, therefore, the data science sandbox can be considered the data provider. In this example, using Spark SQL, one can submit a query to the Hive batch storage component, in order to retrieve the training set needed to segment customers, as mentioned above.

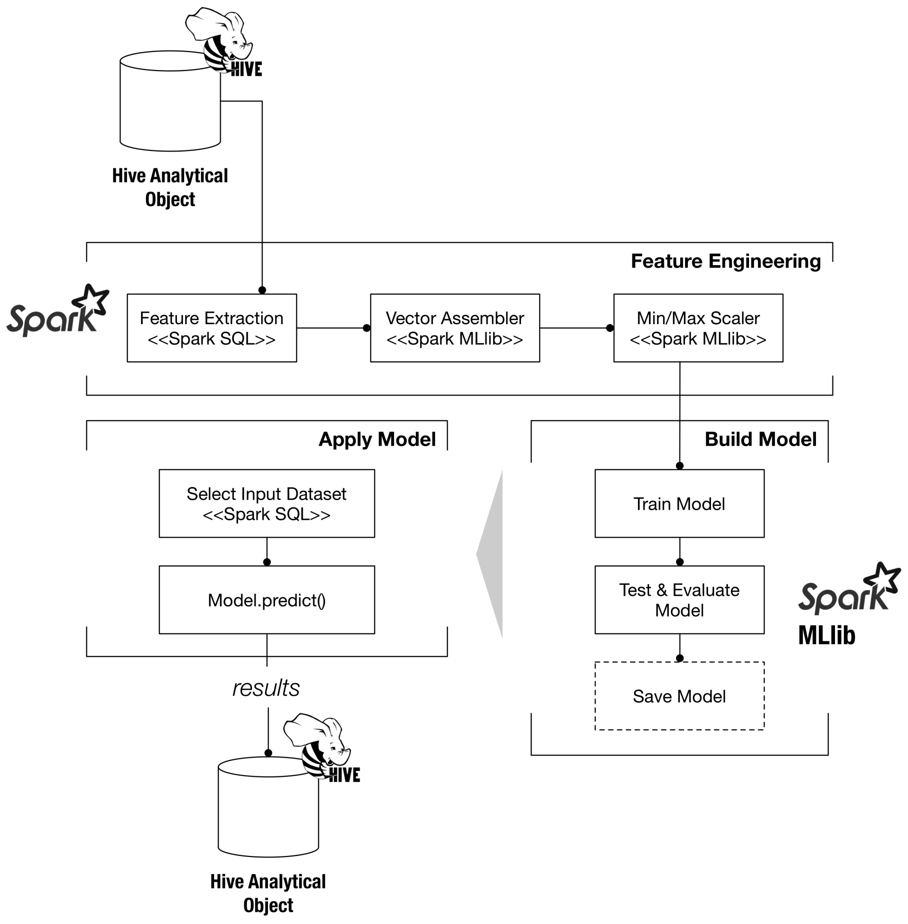


Figure 6.4. Example of using data mining/machine learning algorithms in CPE workloads.

```
Dataset<Row> customerSales = spark.sql("
WITH

customerSales AS (
SELECT c_custkey, c_name, c_city, c_nation, c_region, c_mktsegment, od_monthnuminyear,
sum(quantity) as monthly_quantity, sum(revenue) as monthly_revenue, count(1) as
monthly_orders
FROM ssb_analytical_objects.analytical_obj10
GROUP BY c_custkey, c_name, c_city, c_nation, c_region, c_mktsegment, od_monthnuminyear
),

minmax AS (
SELECT c_custkey, MIN(monthly_revenue) min_monthly_revenue, MAX(monthly_revenue)
max_monthly_revenue
FROM customerSales
GROUP BY c_custkey)

SELECT customerSales.c_custkey, c_name, c_city, c_nation, c_region, c_mktsegment,
stddev((monthly_revenue - min_monthly_revenue)/(max_monthly_revenue -
min_monthly_revenue)) revenue_monthly_stddev, sum(monthly_revenue) revenue,
sum(monthly_orders) total_orders, sum(monthly_quantity) quantity
FROM customerSales
LEFT OUTER JOIN minmax ON customerSales.c_custkey = minmax.c_custkey
GROUP BY customerSales.c_custkey, c_name, c_city, c_nation, c_region, c_mktsegment
");
```

4. One of the main tasks in data mining/machine learning processes is feature engineering. In this example, the previous query already did part of the job by creating a training set with customers and their respective orders, revenue, and standard deviation regarding monthly revenue. However, one way to improve the efficiency of learning algorithms is feature scaling, i.e., providing a standard scale for all features. Moreover, in Spark MLlib 2, all features must be contained in a Vector object and, therefore, one can use the VectorAssembler to transform the original data from Hive, while at the same time replacing null values with zeros, so that the VectorAssembler can be properly used. The following code snippet illustrates these simple feature engineering tasks used in this workload. Spark offers several other functions for these purposes.

```
VectorAssembler assembler = new VectorAssembler()
   .setInputCols(new String[]{“revenue_monthly_stddev”, “revenue”, “total_orders”})
   .setOutputCol(“vectors”);
Dataset<Row> vectorizedData = assembler.transform(customerSales.na().fill(0));
MinMaxScaler scaler = new MinMaxScaler()
   .setInputCol(“vectors”)
   .setOutputCol(“features”);
MinMaxScalerModel scalerModel = scaler.fit(vectorizedData);
```

5. The next step consists in training and testing the K-means model with the previously prepared features. After training the model, data scientists can evaluate its performance, by changing the number of clusters to be created and analyzing the behavior of the within cluster sum of squared errors, and by manually inspecting the clusters’ centers, for example. This evaluation allows the understanding of what each cluster means. For example: one cluster may represent customers with less orders, but generating more revenue for the company and at constant monthly rates; in contrast, another cluster may represent irregular customers with several orders, but generating low income for the company. In this workload, for demonstration purposes, the selected number of clusters is 2.

```
KMeans kmeans = new KMeans().setK(2).setSeed(1L);
KMeansModel model = kmeans.fit(trainingSet);

double wssse = model.computeCost(trainingSet);
System.out.println(“Within Set Sum of Squared Errors = ” + wssse);

Vector[] centers = model.clusterCenters();
System.out.println(“Clusters Centers: ”);
for (Vector center : centers) {
   System.out.println(center);
}
```

6. Despite the fact that in this workload one trains, tests, and applies the model in the same Spark application, both the feature engineering models and the K-means model can be permanently saved and used later in future executions of this or other workloads. This means that the models do not need to be trained each time that they are applied.

```
try {
   model.write().save("<path_in_hdfs>");
} catch (IOException ex) {
   System.err.println("Error saving the model!");
}
```

7. The last step is storing the results into the new Hive analytical object. The inclusion of data science models in CPE workloads does not always need to generate new analytical objects. Sometimes, the workload simply refreshes the analytical object with new data. Other times, existing analytical objects can be updated with new attributes (e.g., Hive supports different schemas for different partitions in a table). In this workload, since one is training, testing, and applying the learning model using a single application, each time the workload is executed the analytical object is created/overwritten. This analytical object contains attributes similar to those of the *"customerSales"* Spark Dataset used to train the K-means model, but with the addition of the cluster to which the customer belongs, along with a user-friendly description of the cluster according to its centroid. This is possible by applying the predict method of the K-means model. Having an analytical object containing the customers' buying behavior allows for interesting analyses, even allowing the join between this analytical object and the original one containing all sales transactions. It should be noted that, in this example, *"customerSale"* is a typical Java Bean containing the same attributes as the analytical object stored in Hive.

```
Dataset<CustomerSale> analyticalObject = trainingSet.map((Row r) -> {
   int cluster = model.predict((Vector) r.get(11));
   String levConstantIncome;
   String levRevenueGenerated;
   String levTotalOrders;
   switch (cluster) {
      case 0:
         levConstantIncome = "Buys more frequently";
         levRevenueGenerated = "Low";
         levTotalOrders = "Low";
         break;
      case 1:
         levConstantIncome = "Buys less frequently";
```

```
            levRevenueGenerated = “Average-Higher”;
            levTotalOrders = “Average-Higher”;
            break;
         default:
            levConstantIncome = “NA”;
            levRevenueGenerated = “NA”;
            levTotalOrders = “NA”;
            break;
      }
      CustomerSale c = new CustomerSale();
      c.setC_custkey(r.getInt(0));
      c.setC_name(r.getString(1));
      c.setC_city(r.getString(2));
      c.setC_nation(r.getString(3));
      c.setC_region(r.getString(4));
      c.setC_mktsegment(r.getString(5));
      c.setRevenue_monthly_stddev((int) r.getDouble(6));
      c.setRevenue((int) r.getDouble(7));
      c.setTotal_orders(r.getLong(8));
      c.setCluster(“cluster” + cluster);
      c.setLev_constant_income(levConstantIncome);
      c.setLev_revenue_generated(levRevenueGenerated);
      c.setLev_total_orders(levTotalOrders);
      return c;
   }, Encoders.bean(CustomerSale.class));

analyticalObject.write().mode(SaveMode.Overwrite).insertInto(“<hive table>”);
```

6.5.2 Text Mining, Image Mining, and Video Mining Models

Although unstructured data mining is relatively different from structured data mining, the general steps presented in the previous subsection can still be applied. For that reason, as Figure 4.2 demonstrates, the proposed method for CPE is fairly similar both for structured and unstructured data. Obviously, while one can use classification, regression, clustering, association rules, or time series forecasting for extracting patterns and making predictions in structured environments (Pujari, 2001), regarding unstructured contexts, the techniques may be severely different (although sometimes they overlap). Regarding the technologies to be used in these contexts, they depend on the specific use case. For example, Spark MLlib does not have an extensive set of text mining algorithms, but it offers some text-based feature extraction and clustering algorithms (e.g., TF-IDF, Word2Vec, and LDA). However, currently, Spark does not offer adequate support for image or video mining algorithms. In these contexts, choosing complementary technologies for the data science sandbox is appropriate, such as Python, for example, which offers some interesting libraries oriented towards image mining.

In BDWing environments, the inclusion of unstructured data science models in CPE workloads has the goal of extracting structured predictive attributes, which are structured findings extracted from unstructured sources. These attributes can be considered the structured value that can be extracted from unstructured data, which by itself in its raw state would not be significantly relevant for BDWing purposes. The data has to be prepared and enriched using adequate techniques and technologies capable of mining the value from these sources. Only then, the results of these tasks provide analytical value.

Figure 6.5 presents a workflow based on Figure 4.2, including several techniques useful in these scenarios. As can be seen, despite the challenges and complexity of unstructured data mining, the general tasks remain similar to a CPE workload that includes data mining/machine learning algorithms for structured data. First, the data is collected using batch or streaming mechanisms. For a specific source and a specific technique, a previously trained model is used to extract structured patterns from text, images, or video, depending on the use case. Complementary datasets can also be used for data enrichment, if applicable. After all the attributes of the analytical object are created (descriptive, factual, and predictive), the analytical object is ready to be used. So far, there is no difference compared to the CPE workload of the previous subsection.

That being said, the difference solely relies on the use of new and challenging techniques: for text mining, techniques such as information extraction and sentiment analysis can be significantly useful for extracting entities (e.g., people and dates), relationships (e.g., events), and sentiments from raw text (Gandomi & Haider, 2015). This will make possible the fueling of analytical objects which can be significantly useful for several organizations; for image mining purposes, techniques like object recognition (e.g., template matching) and image classification can also be significantly useful (Zhang,

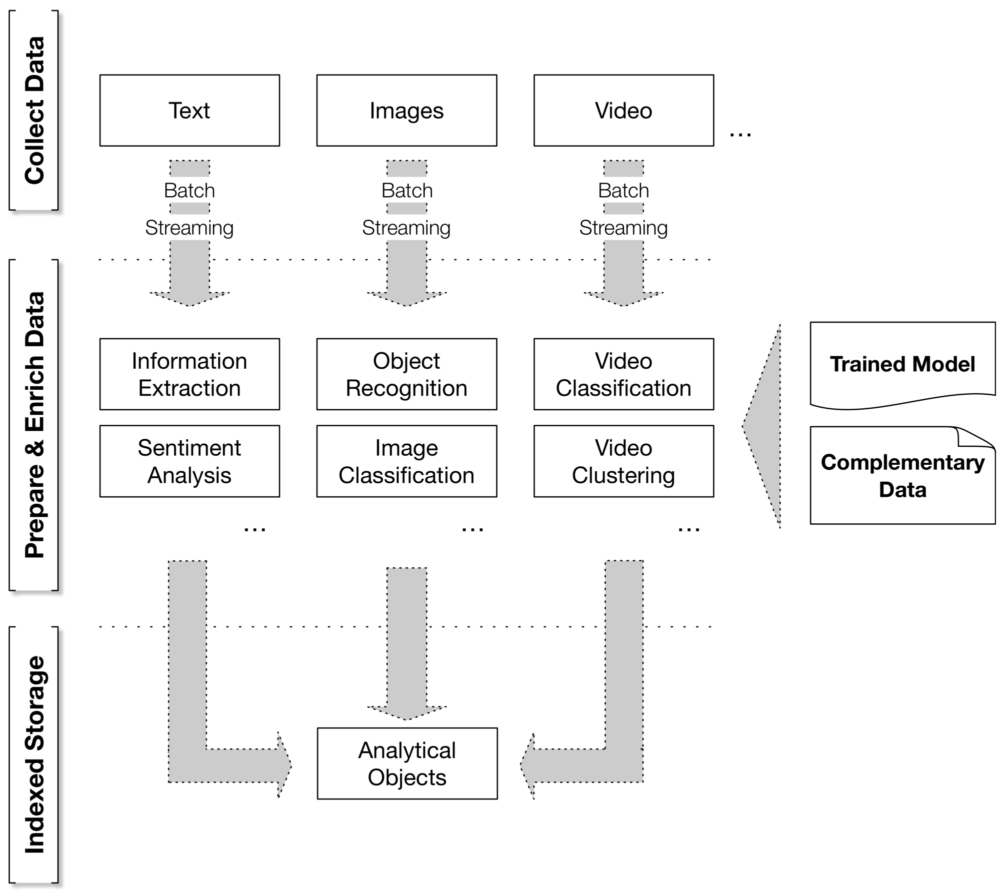


Figure 6.5. Including unstructured data science models in CPE workloads.

Hsu, & Lee, 2001); finally, for video mining, video classification and video clustering can be used (Vijayakumar & Nedunchezhian, 2012), which have similar goals as their structured data mining counterparts, although, of course, with different specifications. These are just some examples of possible techniques, since the list can be considerably extended. However, for demonstrating their role in BDWing environments, these techniques provide adequate examples of the capabilities of unstructured analytics in BDWs.

This page was intentionally left blank

Chapter 7. Evaluating the Performance of Big Data Warehouses

This chapter discusses the evaluation of BDWs built using the proposed approach. In order to evaluate the performance of a BDW, several related benchmarks can be used, such as the TPC-DS benchmark (TPC, 2017a) or the SSB benchmark (O'Neil et al., 2009), for example. In this work, an extension of the SSB benchmark, named SSB+ (C. Costa & Santos, 2018), was specifically created for BDWing contexts, combining batch and streaming data. An extension of the original SSB benchmark was needed due to the lack of workloads that combine volume, variety, and velocity of data, with adequate customization capabilities and integration with current versions of different Big Data technologies. Moreover, one needs to evaluate different modeling strategies (e.g., flat structures, nested structures, and star schemas) and different workload considerations (e.g., partitioned analytical objects and dimensions' size in star schema-based BDWs) and, therefore, an adaptation of the SSB benchmark is required. This chapter presents the SSB+ Benchmark, discussing the performance, advantages, and disadvantages of several design and implementation choices in the proposed approach, extending and integrating previously published scientific works (C. Costa & Santos, 2018; E. Costa, Costa, & Santos, 2017).

7.1 The SSB+ Benchmark

This section details the SSB+ Benchmark, namely the data model, queries, system architecture, and infrastructure. Besides serving as a proof-of-concept validation, presenting several insights related to relevant design decisions for BDWs, the SSB+ Benchmark is useful for practitioners to evaluate the performance of their own implementations.

7.1.1 Data Model and Queries

The SSB+ Benchmark data model (C. Costa & Santos, 2018), presented in Figure 7.1, is based on the original SSB benchmark (O'Neil et al., 2009), so all the original tables remain the same (*"lineorder", "part", "supplier",* and *"customer"*), with the exception of the *"date"* dimension, which has been streamlined to remove the several temporal attributes that are not used in the 13 original

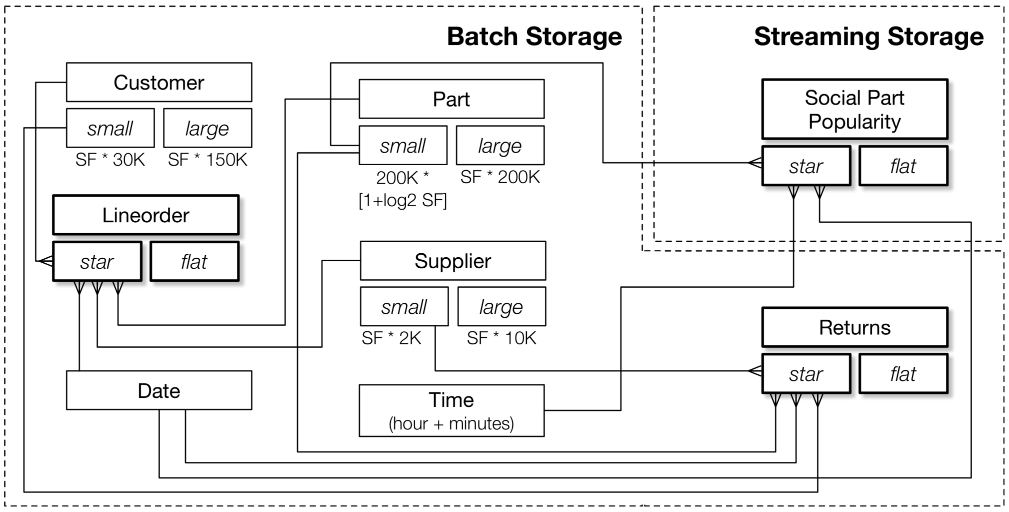


Figure 7.1. SSB+ data model. Adapted from (O'Neil et al., 2009; C. Costa & Santos, 2018) with extended content.

SSB queries. These 13 queries do not suffer any modification besides the replacement of "where clause" joins for ANSI SQL joins with an explicit join operator. This measure is taken to assure an optimal execution plan in the optimizers of the query engines.

Since the original SSB benchmark only takes into consideration a star schema-based DW, the SSB+ also includes jobs for transforming the *"lineorder"* star into a flat *"lineorder"* analytical object. Obviously, the original 13 SSB queries are also modified to match the new flat analytical object. These changes allow us to compare the advantages and disadvantages of star schemas and flat structures for BDWs. Moreover, the SSB+ also considers two different dimensions' sizes: the original TPC-H sizes (TPC, 2017b) (benchmark in which the original SSB is based), which includes larger *"part", "customer",* and *"supplier"* tables; and the original SSB sizes, in which these tables are smaller to represent more traditional dimensions in the retail context. This SSB+ feature allow us to understand the impact of the dimensions' size in star schema-based BDWs. Furthermore, the SSB+ also includes a *"returns"* table (flat analytical object and star schema fact table) and 4 new queries to evaluate the performance of drill across operations and window and analytics functions.

Regarding the streaming workloads of the SSB+ Benchmark, a new *"time"* dimension table is included, as the data stream has a "minute" granularity. This new dimension can then be joined with the new *"social part popularity"* fact table, as well as other existing dimensions like *"part"* and *"date"*. A flat version of this fact table is also available for performance comparison purposes. The *"social part popularity"* table contains data from a simulated social network, where users express their sentiments regarding the parts sold by the organization represented in the SSB and SSB+ Benchmark. Along with these new tables, 3 new streaming queries were developed for both the star schema-based BDW and the flat-based BDW, performing several aggregation, filtering, union, and join operations on streaming data. All the applications, scripts, and queries for the SSB+ Benchmark can be found in (C. Costa, 2017).

7.1.2 System Architecture and Infrastructure

The SSB+ Benchmark takes into consideration several technologies to accomplish different goals, from data CPE workloads to querying and OLAP tasks. These technologies are presented in Figure 7.2. Starting with the CPE workloads, for batch data, the SSB+ considers a Hive script with several beeline commands that load the data from HDFS to the Hive tables stored in the ORC format, an efficient columnar file format for data analytics. Several SFs can be generated using the original SSB generator. This work considers the SF=30, SF=100, and SF=300 for the batch performance evaluation. Regarding streaming data, a Kafka producer generates simulated data at configurable rates, and this data is processed by a Spark Streaming application that finally stores it in Hive and Cassandra. Streaming data is stored both in Hive and Cassandra for benchmarking purposes (see section 7.3).

For querying and OLAP, this work considers both Hive on Tez and Presto, which are two robust and efficient SQL-on-Hadoop engines (Santos et al., 2017). Obviously, practitioners can run the SSB+ Benchmark with any SQL-on-Hadoop engine of their choice, as long as they develop the adequate scripts to run the workloads. Currently, the repository pointed in the previous subsection contains only applications and scripts supporting the technologies mentioned in Figure 7.2. However, all the content of the repository is open to the public, in order to facilitate any change or extension. Hive and Presto are used to provide insights from different SQL-on-Hadoop engines, in order to see if the conclusions hold true for more than one engine, since one of them may perform better with certain data modeling

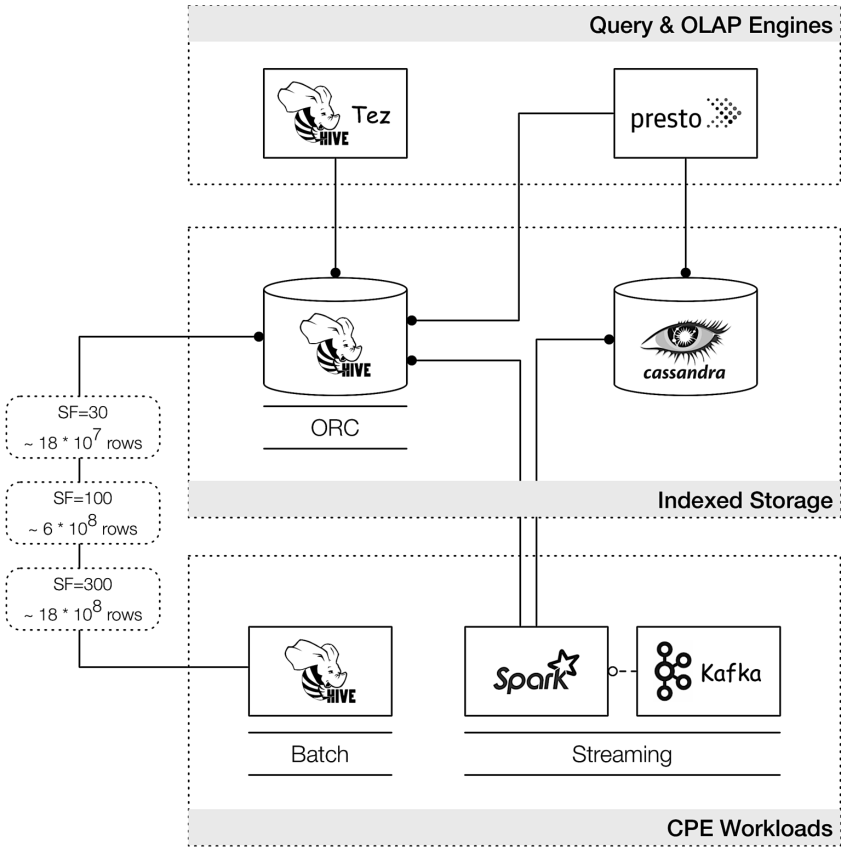


Figure 7.2. SSB+ architecture. Adapted from (C. Costa & Santos, 2018).

strategies, for example. However, in the streaming workloads, only Presto is used, since it targets interactive SQL queries over different data sources, including NoSQL databases, which is not a very proclaimed feature in Hive, although it can also be used for this purpose. Moreover, despite Tez' tremendous improvements to Hive's performance, Hive on Tez may not be considered a low-latency engine, as the results presented in this chapter may suggest.

The infrastructure used in this work is a 5-node Hadoop cluster with 1 HDFS NameNode (YARN ResourceManager) and 4 HDFS DataNodes (YARN NodeManagers). The hardware used in each node includes:

- 1 Intel core i5, quad core, with a clock speed ranging between 3.1GHz and 3.3 GHz;
- 32GB of 1333MHz DDR3 RAM, with 24GB available for query processing;

- 1 Samsung 850 EVO 500GB Solid State Drive (SSD) with up to 540 MB/s read speed and up to 520 MB/s write speed;
- 1 gigabit Ethernet card connected through Cat5e Ethernet cables and a gigabit Ethernet switch.

The operative system in use is CentOS 7 with an XFS file system, and the Hadoop distribution is the Hortonworks Data Platform (HDP) 2.6. Besides Hadoop, a Presto coordinator is also installed on the NameNode, as well as 4 Presto workers on the 4 remaining DataNodes. All configurations are left unchanged, apart from the HDFS replication factor, which is set to 2, as well as Presto's memory configuration, which is set to use 24GB of the 32GB available in each worker (identical to the memory available for YARN applications in each NodeManager).

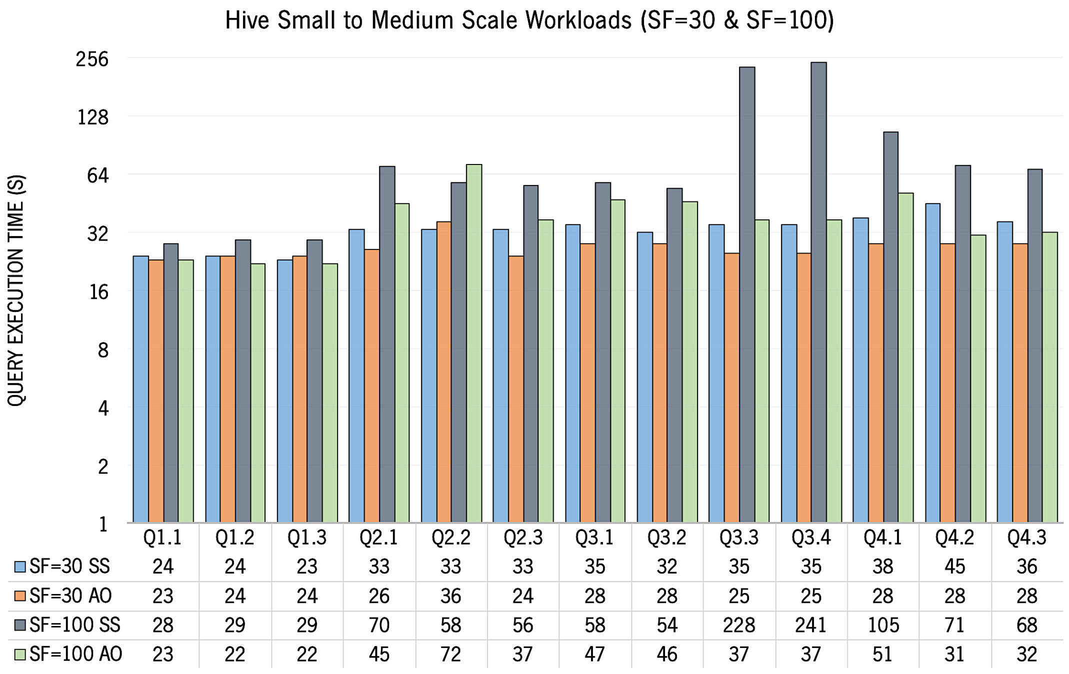
7.2 Batch OLAP

Batch OLAP queries are seen as queries that take as input vast amounts of data stored in the batch storage component of the BDW. This section discusses the performance of batch OLAP queries for BDWs using two modeling approaches: star schemas and flat analytical objects. Moreover, this section also addresses the impact of the dimensions' size in star schemas, the use of nested structures in analytical objects, the improvement of the BDW's performance by using adequate data partitioning, and the performance of drill across queries and window and analytics functions.

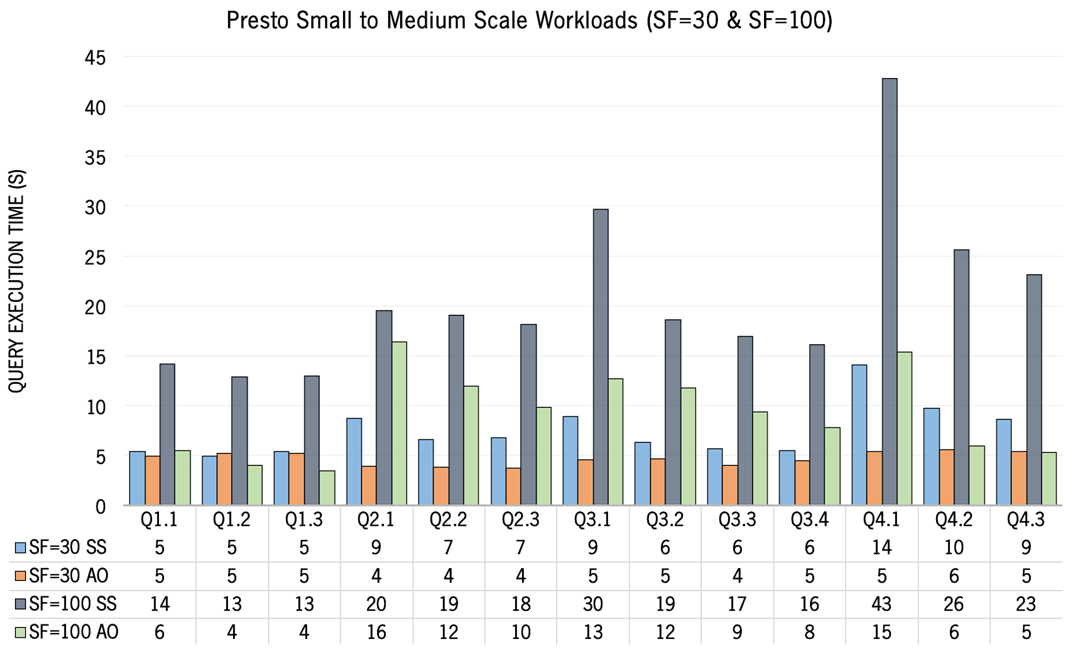
7.2.1 Comparing Flat Analytical Objects with Star Schemas

This first evaluation consists in analyzing the performance, storage size, CPU usage, and memory requirements of flat analytical objects and star schemas, using the 13 SSB+ batch queries. Regarding the star schema, all the workloads depicted in this subsection use the larger dimensions instead of the smaller ones, which will only be discussed in subsection 7.2.3.

Analyzing the small to medium SFs, illustrated in Figure 7.3, it can be concluded that the performance advantage of having flat analytical objects is quite noticeable. For the majority of the queries, the flat



	Q1.1	Q1.2	Q1.3	Q2.1	Q2.2	Q2.3	Q3.1	Q3.2	Q3.3	Q3.4	Q4.1	Q4.2	Q4.3
SF=30 SS	24	24	23	33	33	33	35	32	35	35	38	45	36
SF=30 AO	23	24	24	26	36	24	28	28	25	25	28	28	28
SF=100 SS	28	29	29	70	58	56	58	54	228	241	105	71	68
SF=100 AO	23	22	22	45	72	37	47	46	37	37	51	31	32



	Q1.1	Q1.2	Q1.3	Q2.1	Q2.2	Q2.3	Q3.1	Q3.2	Q3.3	Q3.4	Q4.1	Q4.2	Q4.3
SF=30 SS	5	5	5	9	7	7	9	6	6	6	14	10	9
SF=30 AO	5	5	5	4	4	4	5	5	4	5	5	6	5
SF=100 SS	14	13	13	20	19	18	30	19	17	16	43	26	23
SF=100 AO	6	4	4	16	12	10	13	12	9	8	15	6	5

Figure 7.3. Small to medium batch SSB+ workloads.
Star schema (SS); analytical object (AO). Hive's results are based on (E. Costa et al., 2017).

object is able to considerably outperform the star schema, especially in the SF=100 workload, wherein the performance of a star schema with a high number of rows starts to degrade. Interestingly, such phenomenon does not hold true for Hive's Q2.2, and the star schema presents better performance in this scenario, possibly due to some performance problems in Hive's ability to process string range comparisons *("p_brand1 between 'MFGR#2221' and 'MFGR#2228')* in significantly larger amounts of data (see Figure 7.4 to understand the storage size impact of the flat analytical object). The same phenomenon does not occur in the Presto SQL-on-Hadoop engine.

Moreover, looking into Hive's Q1.1, Q1.2, and Q1.3, the flat analytical object and the star schema performance is fairly similar, which is comprehensible, since for these queries, the star schema only needs to join the fact table *"lineorder"* with the dimension *"date".* As the flat analytical object is around 2.5 times bigger than the corresponding dimensional DW stored in the ORC format (see Figure 7.4), it balances out the cost of the join operation. However, in Presto's SF=100 workload, despite this fact, the flat analytical object still outperforms the star schema. At this point, Presto started to present very satisfactory performance when using completely flat structures.

Regarding the large-scale batch workload (SF=300), depicted in Figure 7.5, the trend continues, namely the overall performance advantage of using flat structures. The performance of a Hive star schema for most of the queries is not satisfactory for interactive scenarios, often being more than 3 or 4 times slower than a flat structure. There are some exceptions (Hive/Presto's Q1.1, Q1.2, and Q1.3; and Hive's Q4.3), mainly due to the aforementioned reasons, i.e., the storage size of the flat structure causes a significant overhead in the I/O tasks of the queries, which mainly makes them I/O

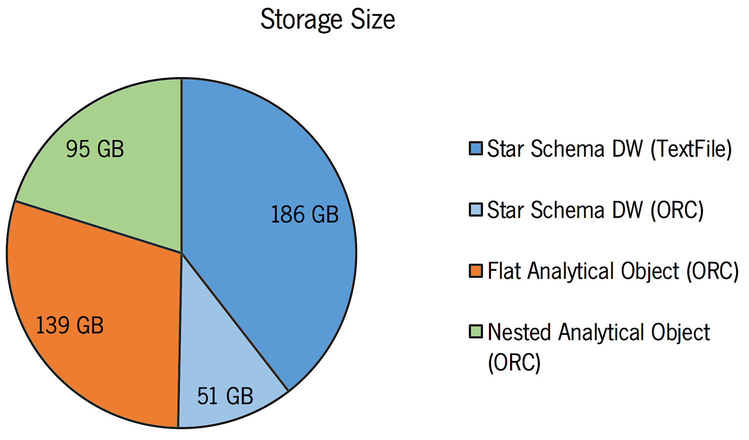
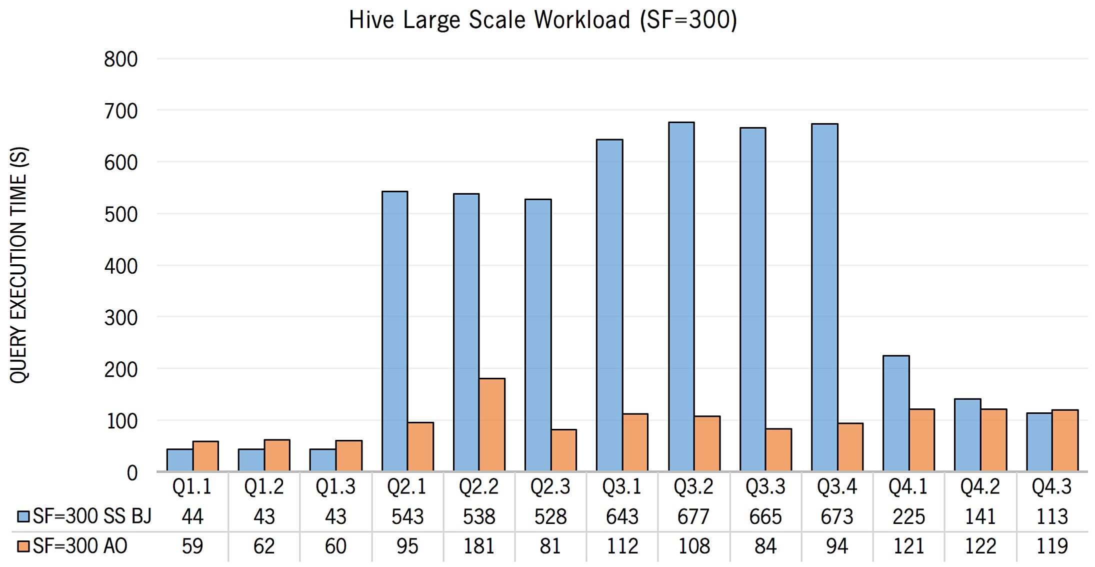


Figure 7.4. Storage size for the SF=300 using different modeling approaches.

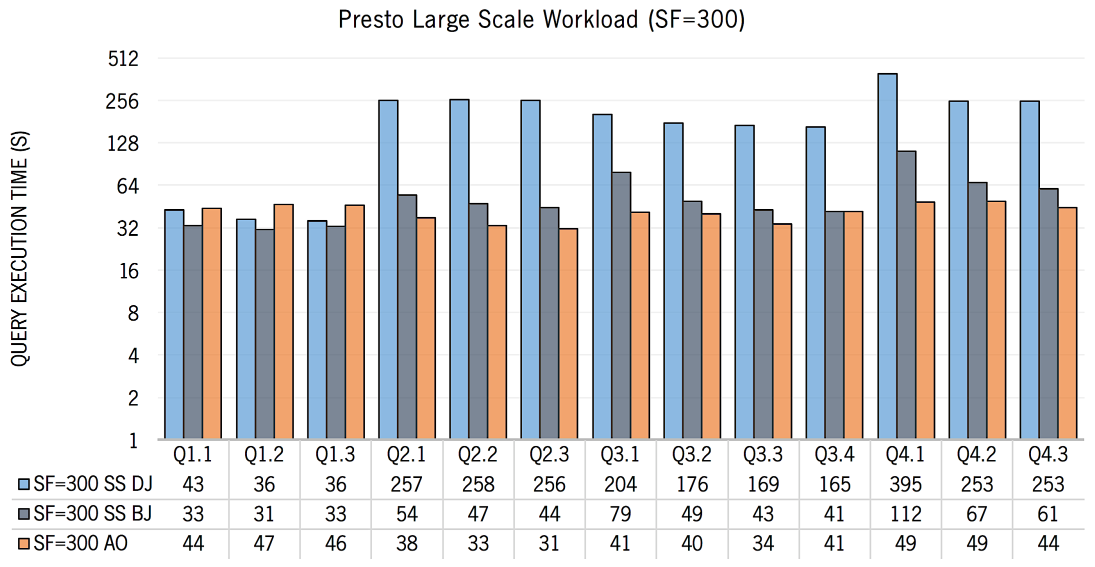
bound queries, and causes the flat analytical object to perform worse than the star schema. Consequently, despite the fact that flat structures tend to perform significantly better than star schemas in these environments, there are certain queries wherein joining a fact table with a small dimension (e.g., *"date"* dimension) is faster than executing the same queries on flat structures. However, one also needs to consider the storage size of these two data sources. Looking at Figure 7.4, the entire star schema DW using the ORC file format has around 51GB, while its flat counterpart has around 139GB. Considering the infrastructure used in this work and previously described in subsection 7.1.2, the entire star fits into memory, while the flat analytical object far surpasses the total amount of memory available for querying.

Smaller dimensions allow for a very efficient type of join, known as broadcast join (or map join in Hive) (Floratou et al., 2014). When using broadcast joins, the smaller tables involved in the join operation are broadcasted to the memory of the nodes involved in the computation, which means that the large table (traditionally a fact table) is joined with all these structures in memory, while it is being processed throughout the nodes. The effects of using broadcast joins can be seen in Figure 7.5, in which Presto reveals a significant decrease in query execution times, comparing to the more conventional distributed join. However, despite this advantage, Presto is even faster when using flat structures that do not need any join at all. Such results do not favor the dimensional approach for DWs in Big Data environments.

Moreover, doing broadcast joins is not always possible, since this technique requires that the dimensions fit into a fraction of the memory available for query processing, which is not always the case if the dimensions are naturally large or become larger through the application of type 2 SCD techniques (Jukic et al., 2017; Kimball & Ross, 2013). Certain query optimizers do not automatically select the most appropriate join technique according to the size of the tables, which is the case of Presto's optimizer in version 0.180. In this work, the two join techniques (distributed and broadcast) were manually selected. When enforcing broadcast joins, one must be aware that if the broadcasted input is too large, "out of memory" errors can occur, due to the lack of memory to process all inputs. Hive 1.2.1, included in the Hortonworks Data Platform used in this work, automatically selects the most appropriate type of join. Nevertheless, since all configurations are left to their default values, Hive does not trigger a map join in the SF=300 workload (and for certain SF=100 queries as well), since the threshold regarding the fraction of memory dedicated for map join is probably surpassed. This leads to a severe performance degradation for the star schema implemented in Hive. Such



	Q1.1	Q1.2	Q1.3	Q2.1	Q2.2	Q2.3	Q3.1	Q3.2	Q3.3	Q3.4	Q4.1	Q4.2	Q4.3
🟦 SF=300 SS BJ	44	43	43	543	538	528	643	677	665	673	225	141	113
🟧 SF=300 AO	59	62	60	95	181	81	112	108	84	94	121	122	119



	Q1.1	Q1.2	Q1.3	Q2.1	Q2.2	Q2.3	Q3.1	Q3.2	Q3.3	Q3.4	Q4.1	Q4.2	Q4.3
🟦 SF=300 SS DJ	43	36	36	257	258	256	204	176	169	165	395	253	253
⬛ SF=300 SS BJ	33	31	33	54	47	44	79	49	43	41	112	67	61
🟧 SF=300 AO	44	47	46	38	33	31	41	40	34	41	49	49	44

Figure 7.5. Large-scale batch SSB+ workload.
Star schema with broadcast joins (SS BJ); analytical object (AO); star schema with distributed joins (SS DJ). Hive's results are based on (E. Costa et al., 2017).

phenomenon raises a relevant discussion regarding the effect of the dimensions' size in the star schema modeling approach, which will be further discussed in subsection 7.2.3.

Besides these memory requirements, during the benchmark, one analyzed the cumulative and peak memory for each query running in Presto, and it was observed that the star schema tends to achieve a higher peak memory when processing queries. The total amount of memory used for star schema-based queries is also substantially higher than flat-based queries in Presto's workloads. Regarding CPU usage, Figure 7.6 shows that despite being slower, the star schema tends to have a significantly higher CPU usage than a flat analytical object. On average, in Presto's workloads, the star schema uses considerably more CPU time. Consequently, significantly higher CPU usage can also be seen as a drawback of star schema-based BDWs.

7.2.2 Improving Performance with Adequate Data Partitioning

Data partitioning can significantly impact the performance of storage systems. DWs are typically partitioned by date, or parts of a date (e.g., year, month, and day). However, there are other relevant attributes that can be typically used for partitioning, which are related to specific implementation

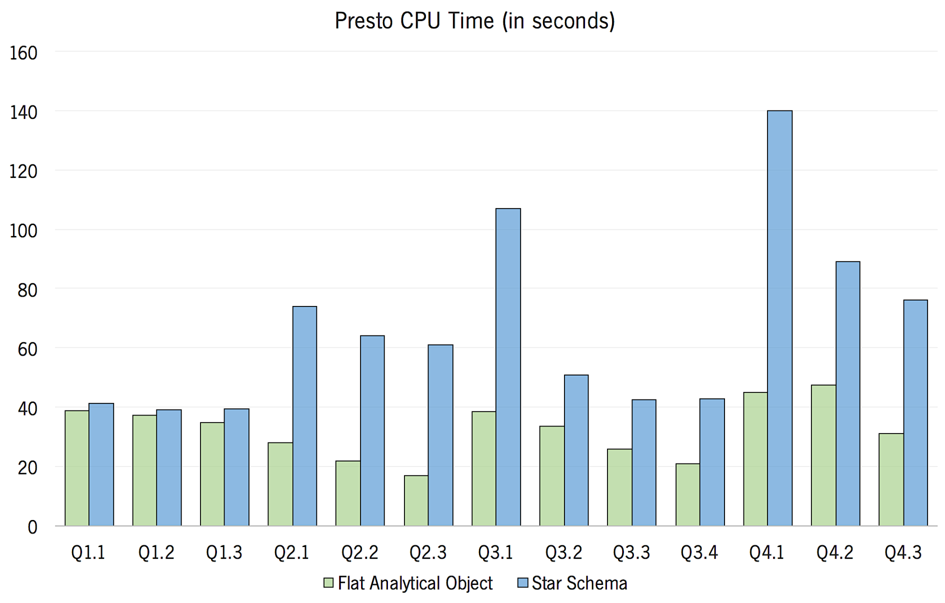


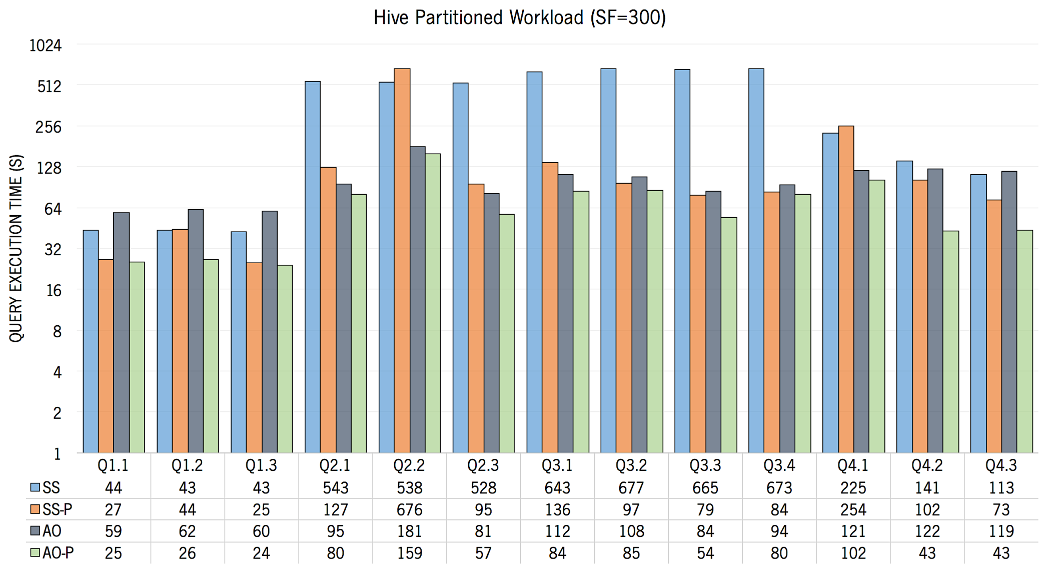
Figure 7.6. Presto CPU time for the star schema and the flat analytical object.

contexts. Depending on the attributes frequently used in the where clause of the queries, data partitioning can considerably reduce query execution times, since the amount of data that needs to be processed will be much smaller. Another benefit of this technique is the simplification of CPE workloads, due to the fact that one can make specific changes to previously loaded partitions, without affecting the entire dataset. For example, if CPE workloads for sales data are executed each day, and there was a mistake in the data that was loaded yesterday, today's workloads can correct these mistakes by completely overwriting yesterday's partition without affecting the entire dataset. Sometimes, especially in Big Data environments, completely overwriting partitions becomes more efficient than updating multiple records. Furthermore, frequently, Big Data storage systems do not provide adequate updating capabilities (e.g., HDFS/Hive without ACID transactions enabled).

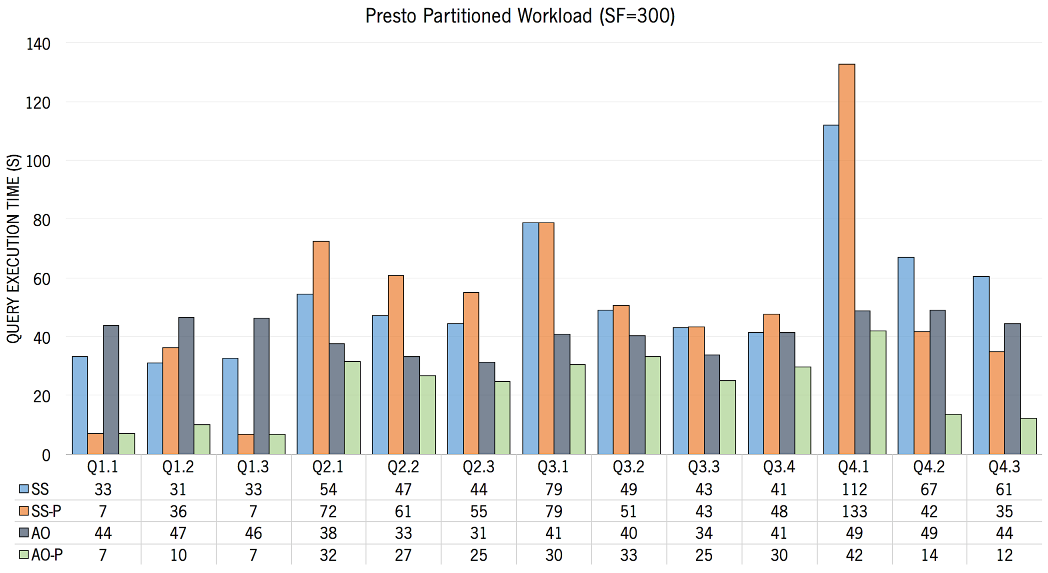
The workloads presented in the previous subsection do not make use of partitioning strategies, which is not very typical in real-world contexts. However, it allows the evaluation of queries over large amounts of data. In certain organizations and contexts, even daily batches of data are significantly large and, therefore, it becomes relevant to understand how well flat analytical objects and star schemas can handle a large volume of data for certain infrastructures. In contrast, the workloads presented in this subsection use the SSB+ dataset partitioned by *"order year"*, which is the attribute that appears more frequently in the where clause of the 13 SSB+ batch queries.

Figure 7.7 presents the results of the SF=300 workload using data partitioning, including a flat analytical object and a star schema, and comparing them with the results achieved in the SF=300 workload of the previous subsection. Obviously, the performance advantage of using partitions is noticeable when the queries include the *"order year"* attribute as a filter. This is the main reason why the dataset was partitioned in the first place. This is true both for Presto and Hive. However, while Presto typically presents the expected behavior when the query does not benefit from the partition scheme, i.e., there is an increase in query execution or any difference is negligible, Hive presents an odd and unexpected behavior at first glance. The Q2 and Q3 variants are not supposed to benefit from this partitioned scheme, since they do not take advantage of any relevant *"order year"* filtering operations in the where clause. The Q3 variants tend to filter *"order year"* using a range of values, but the range is so wide that it is almost equivalent as scanning the entire dataset. Despite this, Hive's execution times for Q2 and Q3 variants (except Q2.2) drop drastically for the star schema using partitions, which is not expected at all.

After inspecting the execution of the queries more closely, one observes that, using data partitioning, generally, some of the query plans for the Q2 and Q3 variants changed, and more mappers and



	Q1.1	Q1.2	Q1.3	Q2.1	Q2.2	Q2.3	Q3.1	Q3.2	Q3.3	Q3.4	Q4.1	Q4.2	Q4.3
🟦 SS	44	43	43	543	538	528	643	677	665	673	225	141	113
🟧 SS-P	27	44	25	127	676	95	136	97	79	84	254	102	73
⬛ AO	59	62	60	95	181	81	112	108	84	94	121	122	119
🟩 AO-P	25	26	24	80	159	57	84	85	54	80	102	43	43



	Q1.1	Q1.2	Q1.3	Q2.1	Q2.2	Q2.3	Q3.1	Q3.2	Q3.3	Q3.4	Q4.1	Q4.2	Q4.3
🟦 SS	33	31	33	54	47	44	79	49	43	41	112	67	61
🟧 SS-P	7	36	7	72	61	55	79	51	43	48	133	42	35
⬛ AO	44	47	46	38	33	31	41	40	34	41	49	49	44
🟩 AO-P	7	10	7	32	27	25	30	33	25	30	42	14	12

Figure 7.7. Large-scale batch SSB+ SF=300 workload with data partitioning.
Star schema (SS); star schema partitioned (SS-P); analytical object (AO); analytical object partitioned (AO-P). Hive's results are based on (E. Costa et al., 2017).

reducers were produced. This number is affected by the organization of the ORC files in the system, as the partitioned Hive table may contain a different number of files with different sizes. Since the number of mappers and reducers is automatically derived in this work, this new number seems to drastically affect the query performance, resulting in a massive drop in query execution times for the star schema. These benefits are also present in the flat analytical object, but with much less predominance, since the query execution times for the Q2 and Q3 variants did not drop as significantly as in the star schema workload.

Presto does not behave like Hive for the star schema Q2 and Q3 variants, presenting expected results, i.e., results similar to the workload without partitions, even demonstrating small increases in execution times, which is to be expected, since there is the overhead of scanning multiple partitions when the query does not take advantage of the data partitioning scheme. In contrast, there is a significant performance advantage when running the Q2 and Q3 variants over the flat analytical object with partitions, which again is an unexpected behavior.

Investigating more in depth on this issue, one could argue that there are certain scenarios where natural hierarchies between attributes can cause ORC files to distribute data in such a way that it unintentionally improves query performance. This phenomenon happens because of a feature known as predicate pushdown at the ORC file/stripe level, together with file/stripe level statistics. For example, if one partitions a table by *"supplier region"*, the queries that filter the data by *"supplier nation"* will also significantly benefit from this partitioning scheme. The attributes *"supplier region"* and *"supplier nation"* form a natural hierarchy, and a specific partition will only contain countries that belong to the corresponding region. Consequently, the ORC files/stripes within this partition will provide statistics regarding the countries contained in them, and the query execution engine (e.g., Presto or Hive) can completely ignore files/stripes that do not contain the countries being filtered in the query, which makes query processing much faster, since it scans less data. However, this does not happen in the partitioning scheme used in this benchmark, as the Q2 variants do not filter the data by any attribute hierarchically related to *"order year"*, and Q3.1, Q3.2, and Q3.3 only discard 1 in 8 years of data. Therefore, as previously explained, one can only conclude that the different organization of ORC files when using partitions may also affect stages, tasks, and drivers that are planned in Presto's queries, resulting in a performance boost, similarly to the one caused by having different numbers of mappers and reducers in Hive, but with less predominance.

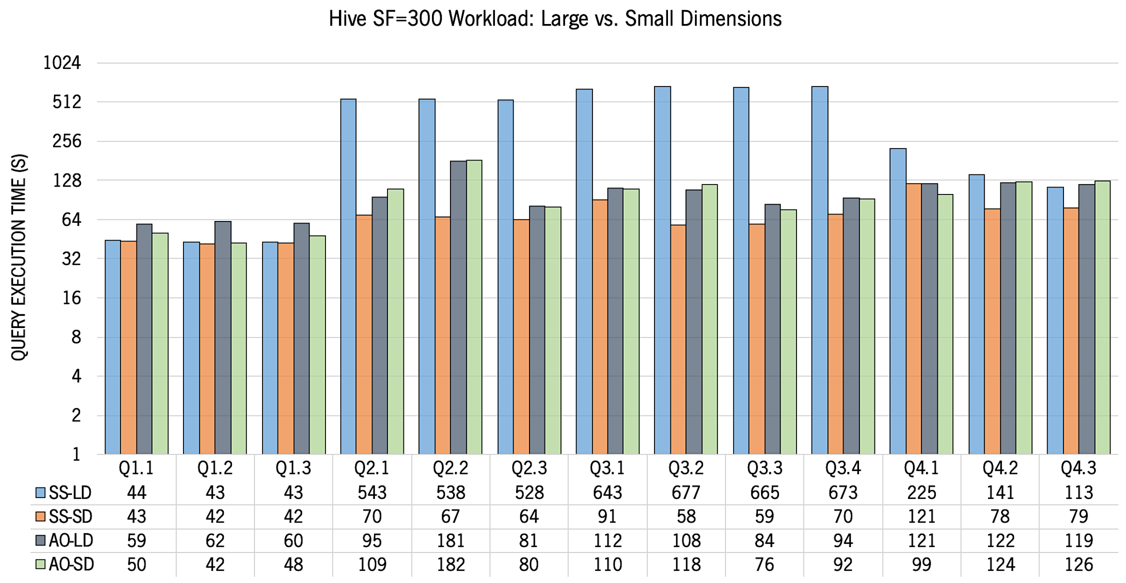
Overall, data partitioning is a mechanism that BDWing practitioners need to seriously take into consideration, as the performance advantage it brings is significantly noticeable. One needs to understand recurrent query patterns, namely the attributes that appear more frequently in where clauses, as well as specific needs for CPE workloads, in which data partitioning can be helpful, as previously explained.

7.2.3 The Impact of Dimensions' Size in Star Schemas

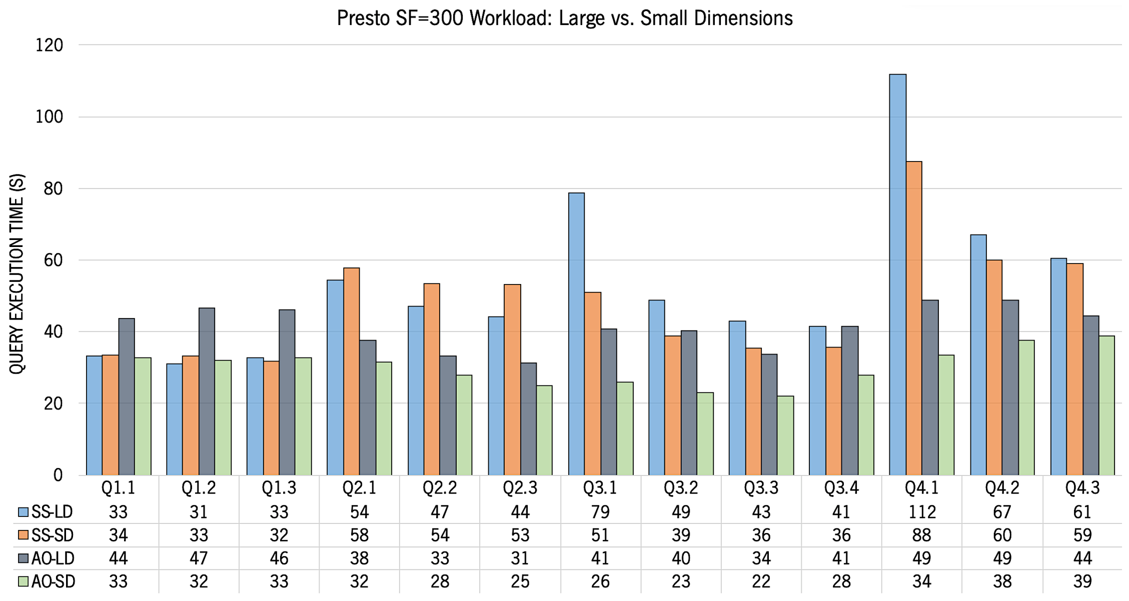
Large dimensions can have a considerable impact in star schema-based DWs, as they require more time to compute the join operations between the fact tables and the dimension tables. In previous workloads, one used larger dimensions' sizes. Although this may not be the usual scenario for many traditional contexts, such as store sales analysis, for example, larger dimensions are typically found in several Big Data contexts. Let us take into consideration a very large Web sales company like Amazon, which has hundreds of millions of customers and products. In these contexts, dimensions' size may be very similar to the ones evaluated in subsection 7.2.1. In Big Data environments, there may be many other use cases that rely on very large dimensions, such as the set of Facebook users, which easily surpasses the 1 billion mark nowadays.

Nevertheless, there are also several contexts wherein dimensions can have a small size, because many organizations can generate several sales transactions only based on a small set of products, customers, and suppliers, for example. For this reason, it becomes interesting to analyze the performance impact caused by dimensions with different sizes. Figure 7.8 illustrates the results of the SF=300 workload for large and small dimensions.

The workloads for the flat analytical object were executed again, since smaller dimensions in the star schema also imply less cardinality in the descriptive attributes of a fully denormalized structure, e.g., if there are less rows in the customer dimension, there are also less distinct values in the *"customer*



	Q1.1	Q1.2	Q1.3	Q2.1	Q2.2	Q2.3	Q3.1	Q3.2	Q3.3	Q3.4	Q4.1	Q4.2	Q4.3
🟦 SS-LD	44	43	43	543	538	528	643	677	665	673	225	141	113
🟧 SS-SD	43	42	42	70	67	64	91	58	59	70	121	78	79
⬛ AO-LD	59	62	60	95	181	81	112	108	84	94	121	122	119
🟩 AO-SD	50	42	48	109	182	80	110	118	76	92	99	124	126



	Q1.1	Q1.2	Q1.3	Q2.1	Q2.2	Q2.3	Q3.1	Q3.2	Q3.3	Q3.4	Q4.1	Q4.2	Q4.3
🟦 SS-LD	33	31	33	54	47	44	79	49	43	41	112	67	61
🟧 SS-SD	34	33	32	58	54	53	51	39	36	36	88	60	59
⬛ AO-LD	44	47	46	38	33	31	41	40	34	41	49	49	44
🟩 AO-SD	33	32	33	32	28	25	26	23	22	28	34	38	39

Figure 7.8. Large-scale batch SSB+ SF=300 workload with small dimensions.
Star schema with large dimensions (SS-LD); star schema with small dimensions (SS-SD); analytical object with large dimensions (AO-LD); analytical object with small dimensions (AO-SD). Adapted from (C. Costa & Santos, 2018).

name" attribute. The cardinality of the attributes can also affect the performance of "group by" and "order by" operators and, therefore, the flat analytical object was reconstructed and evaluated again.

At first glance, looking at Hive's workloads, the result was relatively unexpected. The flat analytical object, which until now was the modeling approach with better performance in almost every query and workload, was surpassed by the star schema with small dimensions. This shows that when using Hive as the SQL-on-Hadoop engine, practitioners may sometimes benefit from modeling the BDW using dimensional structures, which not only saves a considerably amount of storage size, but as Figure 7.8 demonstrates, it can also bring considerable performance advantages. In this scenario, one concludes that if Hive is able to perform a map join, having a larger denormalized structure may not be appropriate for highly dimensional data, such as sales data. The overhead caused by a storage size that is around 2.5 times bigger (see Figure 7.4) leads to a performance drop, and may become a bottleneck for the Hive query engine. Consequently, in these cases, practitioners may consider the strategy presented in subsection 4.3.3, discussing the modeling of traditional dimensions as complementary analytical objects for dimensional BDWing contexts.

Considering only Hive's results in a small dimensions scenario, they would benefit the dimensional approach for modeling BDWs, namely: in the context of traditional DWing, structuring data as fact tables and dimension tables (Kimball & Ross, 2013); and in the context of the proposed approach, structuring data as analytical objects and complementary analytical objects. Consequently, one saw that considering Hive's results, it sometimes makes sense to model parts of the BDW's data that way. However, frequently, Big Data does not adequately fit into the strictures of dimensional and relational approaches (e.g., high volume/velocity sensor data or social media data). Moreover, taking a closer look at Presto's workloads, which are typically much faster than Hive's workloads, it can be observed that, generally, the star schema with smaller dimensions is significantly slower than the corresponding flat table. Furthermore, the star schema with smaller dimensions is frequently slower than the flat table with the higher attributes' cardinality (corresponding to larger dimensions). Overall, the star schema with smaller dimensions takes 61% more time to complete the workload when compared to the equivalent flat table. The discussion in this subsection is an adequate example why one uses two SQL-on-Hadoop systems in each workload, as the insights retrieved from these specific tests sometimes differ according to the system.

Summarizing the conclusions, there is no hard rule. In certain BDWing contexts, practitioners need to consider their limitations regarding storage size and the characteristics of a particular dataset: is data highly dimensional? Do the dimensions have a high number of rows or a large storage footprint and, therefore, not enabling map/broadcast joins? Are these dimensions frequently reused by other analytical contexts? In this work, these concerns are discussed in subsection 4.3.3, which is the result of relevant insights provided by this evaluation. Furthermore, practitioners may need to perform some preliminary benchmarks with sample data before fully committing to either the extensive use of complementary analytical objects or the use of flat analytical objects without any complementary joins.

7.2.4 The Impact of Nested Structures in Analytical Objects

Nested structures like maps, arrays, and JSON objects can be significantly helpful in certain contexts. For example, Chapter 8 discusses one of these contexts, describing the implementation of a BDW for smart cities, wherein geospatial analytics is a priority, including several geometry attributes that are typically complex and nested. There are many contexts where the use of nested structures can be adequately integrated in the data modeling approach. As exemplified in subsection 4.3.1, sales analysis is another context where practitioners may find appealing the application of nested structures, namely using a less granular analytical object *"orders"* with the granularity key *"order key",* and using a nested structure to store the data about the products sold in a particular order (e.g., *"product name", "quantity",* and *"revenue").* The proposed approach fosters the use of nested structures when feasible, but is it really the most efficient solution every time? Do the processing of less rows and the smaller storage footprint always create tangible advantages?

To answer these questions, let us consider Figure 7.4, which compares the storage size (SF=300) of the different modeling approaches used in this work. Considering the nested analytical object created for this evaluation, one can conclude that it represents 68% of the equivalent flat analytical object's storage size, and roughly 186% of the equivalent star schema's size. Moreover, this new modeling approach is able to reduce the number of rows from 1.8 billion to just 450 million, since the granularity of this analytical object is *"order",* instead of *"line order".* The data regarding the lines of the order is stored in a nested structure, namely an array of Struct values named *"lines"* (similarly to the Row datatype in certain SQL-on-Hadoop systems).

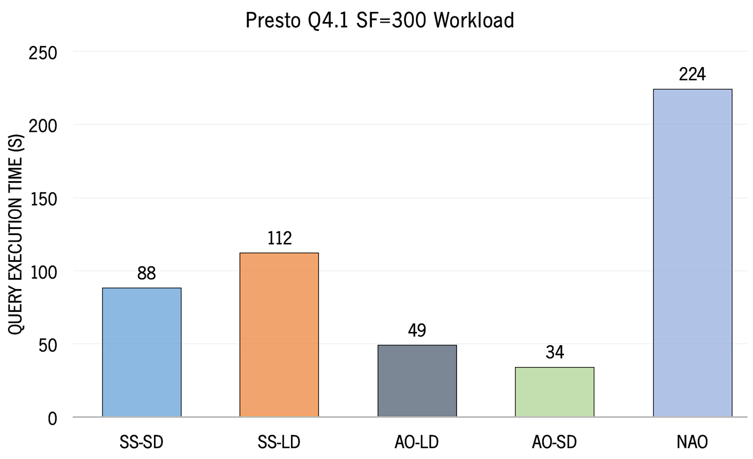


Figure 7.9. Performance of a nested analytical object in the SSB+ context.
Star schema with small dimensions (SS-SD); star schema with large dimensions (SS-LD); analytical object with large dimensions (AO-LD); analytical object with small dimensions (AO-SD); nested analytical object (NAO). Adapted from (C. Costa & Santos, 2018).

At first glance, these numbers look promising, but Figure 7.9 shows a different perspective with the results of executing the SF=300 Q4.1 in all modeling approaches. Q4.1 was chosen because it involves all the dimensions, representing a scenario wherein practitioners will need to aggregate and filter data that is stored in the nested attribute *"lines"*. This allows the evaluation of applying different operators to nested attributes, like the array of Structs used in this context. To achieve the same results as the other modeling approaches, since one is dealing with nested structures, other SQL operators must be used, such as lambda expressions, as the following modification of the Q4.1 SQL query demonstrates:

```
SELECT od_year, c_nation, SUM(profit) AS profit
FROM (SELECT od_date, c_custkey,
      REDUCE(lines,
             CAST(0.0 AS real),
             (s, x) -> IF(x.s_region = 'AMERICA' AND (x.p_mfgr = 'MFGR#1' OR x.p_mfgr =
             'MFGR#2'), s + (x.revenue - x.supplycost), s),
             s -> s) AS profit
      FROM <db_name>.<table_name>)
WHERE c_region = 'AMERICA' GROUP BY od_year, c_nation ORDER BY od_year, c_nation;
```

Despite saving storage space and having much less rows, the nested analytical object is the modeling approach with the lowest performance. It can be concluded that storing a large number of attributes in a complex structure may result in a large overhead regarding query processing times. After all, one is storing all the attributes of the *"part"* and *"supplier"* dimension in this array of Structs, along with the several facts regarding the lines of the order. Such data modeling choice requires the use of lambda expressions or lateral views to answer Q4.1. In this particular test, looking into the query execution, Presto uses the majority of the time computing the lambda expression, which results in a significant increase in query execution time.

These results do not mean that processing nested structures is always detrimental for performance. It depends on the complexity of the structure and what kind of operators will be applied. As shown in this evaluation, highly complex nested structures that will be accessed sequentially to answer most of the queries may not be an adequate design pattern. However, as will be shown in Chapter 8, nested structures offer great flexibility, can be really efficient for certain access patterns, and allow the introduction of new analytical workloads in the BDW, such as intensive geospatial simulations and visualizations.

7.2.5 Drill Across Queries and Window and Analytics Functions

In a traditional DWing context, submitting queries to combine data from multiple fact tables is a frequent phenomenon, which can also be described as drilling across fact tables. On the other hand, window and analytics functions (e.g., over clause, partition by, and rank) also play a relevant role in the ad hoc exploration of the data. Consequently, this subsection explores the performance of BDWs when using drill across and window and analytics functions, following the same strategies already presented above, i.e., using a flat analytical object and a star schema with Hive and Presto (with broadcast joins) as SQL-on-Hadoop engines. Figure 7.10 summarizes the results for the 4 queries in this SF=300 workload, which are available in (C. Costa, 2017). The first 3 queries focus on drill across operations and the last one focuses on window and analytics functions, using the following questions:

- (Q5) Sum of the quantities ordered from the top 20 suppliers that had complaints from American customers in the last 4 years;

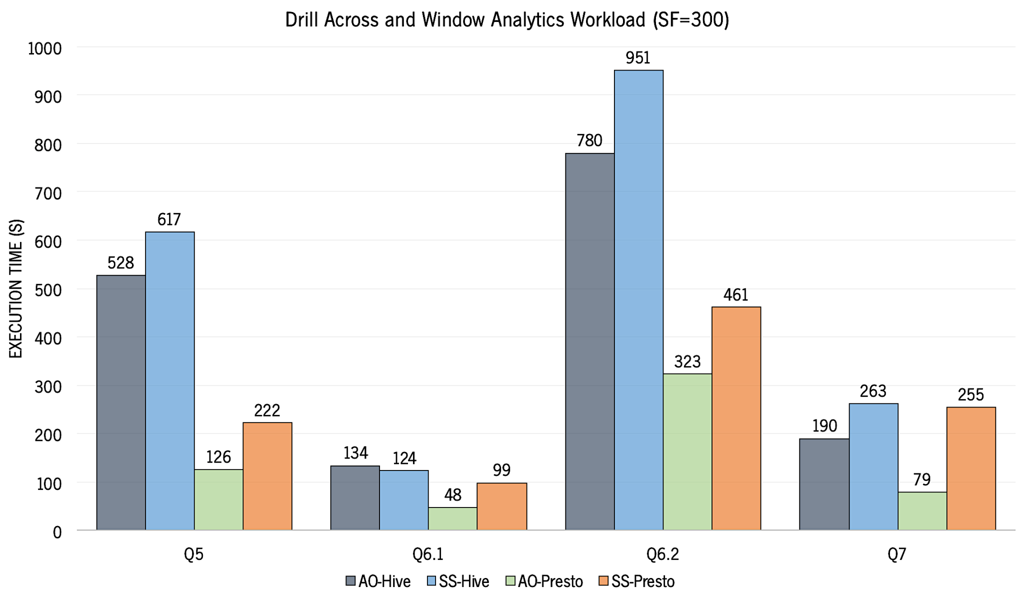


Figure 7.10. Performance of an analytical object (AO) and a star schema (SS) in a workload based on drill across queries and window and analytics functions.

- (Q6.1) Number of times the company have sold parts from the manufacturer 'MFGR#3', provided by Asian suppliers, with an average selling price over 1000$ in America, which have been returned more than one time;
- (Q6.2) Top 10 parts with an average selling price over 1000$ that were returned more than one time;
- (Q7) Top 5 parts of every market segment according to the generated revenue.

The results of this workload revealed that, overall, the performance of a completely flat analytical object is more satisfactory than a star schema, although Q6.1-Hive is an exception to this trend. Considering Presto's results, which was the SQL-on-Hadoop engine that revealed greater differences, one can observe that, frequently, the flat analytical object completes the query in approximately half the time needed for the star schema. Considering Hive's results, the star schema is faster in one of the four queries, although with less significance than Presto's results (only 10 seconds). In contrast, considering Q6.2-Hive, one of the queries in which the star schema is slower than the flat analytical object, the difference in performance is considerable, which shows that certain queries using subqueries with large (less filtered) intermediate results may significantly impact the performance when drilling across fact tables of a star schema. Consequently, it can be concluded that, based on overall performance, flat analytical objects are able to provide significantly lower execution times than star schemas in scenarios with drill across queries and window and analytics functions.

7.3 Streaming OLAP

Streaming scenarios are common in BDWing contexts. The BDW must be able to adequately deal with the high velocity and frequency of the CPE workloads. Daily or hourly batch CPE workloads may not always be the most effective or efficient solution to solve specific problems, and streaming CPE workloads can be significantly useful in these cases. This section evaluates the performance of BDWs created using the proposed approach in streaming scenarios, while discussing several concerns that practitioners must take into consideration.

Using the SSB+ Benchmark, one can observe the performance of the streaming storage of the BDW. As illustrated in Figure 4.3, there are several technologies that can be used to implement this storage component, being responsible for storing data that flows continuously to the BDW with low-latency requirements. In section 4.2, the trade-offs between these different technologies were also discussed. For example, Hive adequately deals with sequential access workloads, typically found in OLAP queries, but it is not adequate for random access, which is often suitable for storing streaming data. In contrast, NoSQL databases like Cassandra are efficient in random access scenarios, but typically fall short in sequential access workloads required for analytical contexts. Consequently, this subsection evaluates the performance of these two technologies using the two main data modeling strategies previously explored: a flat analytical object and a traditional star schema approach, as detailed in section 7.1. The data flow is as follows:

1. A Kafka producer generates 10 000 records each 5 seconds;
2. A Spark Streaming application with a 10 seconds micro batch interval consumes the data for that interval and stores it in Hive and Cassandra;
3. Presto is used to query both streaming storage systems, every hour, over a period of ten hours.

7.3.1 The Impact of Data Volume in the Streaming Storage Component

The performance of the streaming storage system of a BDW typically starts to degrade as the amount of stored data increases. This is the main reason why the proposed approach includes an inter-storage pipeline to move the data from the streaming storage system to the batch storage system. Consequently, in this subsection, one is interested in analyzing how the data volume affects the performance of the streaming storage component of the BDW.

Figure 7.11 illustrates the total execution time for all streaming queries (Q8, Q9, and Q10) during a ten-hour workload with roughly constantly increasing data volume. Each hour, all the queries are executed using Presto, both for the flat analytical object and the star schema, and both for Hive and Cassandra. The queries Q8, Q9, and Q10 are focused on the following analytical questions:

- (Q8) The 2 countries that have the most positive average sentiment polarity united with the 2 countries that have the most negative average sentiment polarity;

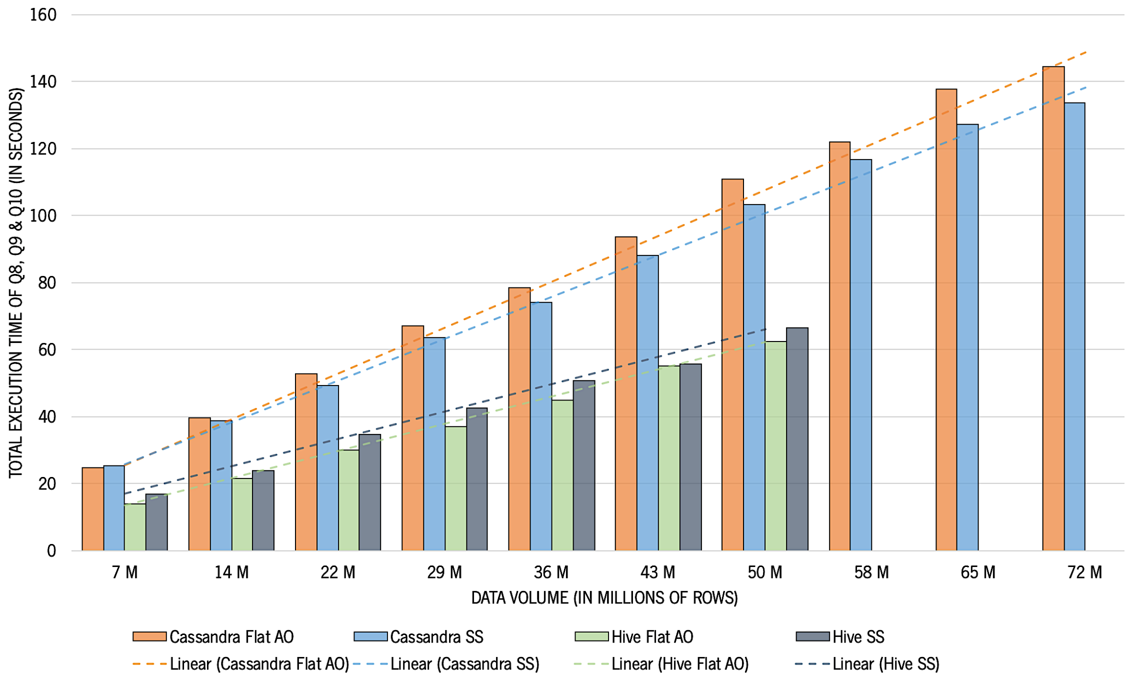


Figure 7.11. Cassandra and Hive SSB+ streaming results.
Star schema (SS); analytical object (AO). Adapted from (C. Costa & Santos, 2018).

- (Q9) The count of sentiments that were expressed by females in Portugal or Spain, grouped by product (part) category and period of the day (e.g., dawn, morning, afternoon, and night);
- (Q10) The groups of product (part) categories and genders having an average sentiment polarity greater than the total average sentiment polarity.

There are several relevant insights that emerge from this evaluation. The first one is regarding the overall effect of data volume on both systems. Looking into the trend present in this ten-hour workload, one can conclude that both Hive and Cassandra are affected in a linear fashion, i.e., as hours pass by (as well as data volume), the increase in the execution time of the workload can be modelled as a linear function. A significant drop in performance as the data volume increases is to be expected in Cassandra, since as previously argued, sequential access over large amounts of data is not one of its strong points. However, this is not expected in Hive, since as the batch workloads demonstrate, when using Presto to query Hive, one is able to achieve much faster execution times than the ones obtained in this streaming workload, even with significantly higher SFs (e.g., SF=30 with 180 000 000 rows).

Despite this observation, detailed afterwards, it can also be concluded that Hive is always much faster than Cassandra, until the mark of 50 million rows is reached. After this moment, it becomes clear that the Spark Streaming micro batch interval is too short for the demand, and the application is also generating thousands of small files in HDFS (storage backend for Hive). Therefore, after the 50 million rows mark, hundreds of micro batches are being delayed, which makes the results for Hive inconclusive, as the number of stored rows does not match those of Cassandra. Overall, it can be concluded that having small micro batch intervals when using Hive may severely deteriorate the performance of the system, complementing the conclusion made before regarding the overhead of having many small files stored in HDFS.

Cassandra also shows some delay in write operations when being queried by Presto, which causes the Spark Streaming application to queue a few micro batch jobs. However, this phenomenon is significantly less concerning than the one in Hive's scenario, and the streaming system is able to control the load without too much delay. Moreover, this is not caused by an increase in data volume, but rather a concurrency issue and resource starvation while Presto queries are running. One can always sacrifice data timeliness by increasing the micro batch interval, but to compare the results between Cassandra and Hive, the write latency and throughput should be identical. In this case, Cassandra adequately handles 20 000 rows each 10 seconds without significant delays, despite being slower, while Hive fails to do so, despite being faster for all workloads under the 58 million rows mark. This efficiency problem is discussed in more detail in the next subsection, among other relevant considerations for streaming scenarios in BDWing systems.

In this analysis, it is also interesting to evaluate the performance of a flat analytical object and a star schema. In this streaming context, the performance is considerably similar in both cases. The star schema is typically faster when using Cassandra, while the flat analytical object is typically faster when using Hive. In the SSB+ Benchmark, the star schema for the streaming scenario is not very extensive or complex, which in this case favors this modeling approach, since queries do not have to join an extensive set of tables. Despite this, it can be concluded that both modeling strategies are feasible, without any significant performance drawback. In the star schema's case, as the dimension tables are stored in Hive, it can also be concluded that using a SQL-on-Hadoop system like Presto is also feasible to combine complementary analytical objects stored in Hive (e.g., *"part" and "time"*) with streaming analytical objects stored in Cassandra. It must be remembered that the proposed approach uses the concept of complementary analytical objects to model dimensions, when practitioners prefer the use of dimensional structures for certain contexts (see subsection 4.3.3).

7.3.2 Considerations for Effective and Efficient Streaming OLAP

A successful streaming application can be seen as an adequate balance between data timeliness and resource capacity. To explain these trade-offs, this subsection is divided into three main problems that emerged from the evaluation of the SSB+ Benchmark, presenting possible solutions to overcome these issues:

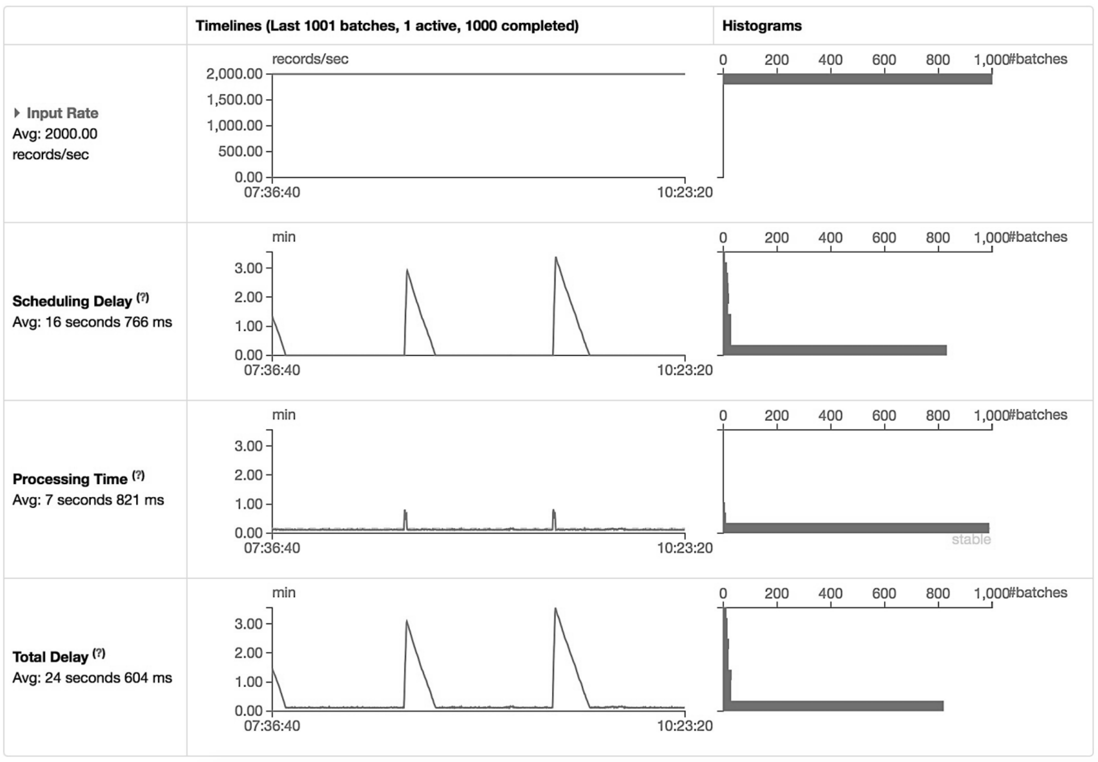
1. High concurrency in multi-tenant clusters (multiple users and multiple technologies) can cause severe resource starvation;

2. Storage systems oriented towards sequential access (e.g., Hive) may present some problems when using small micro batch intervals;
3. Inter-storage pipeline operations and CPE workloads should be properly planned, and the adequate amount of resources should be reserved.

Starting with the first problem, the proposed approach promotes a shared-nothing and scale-out infrastructure that is typically capable of multi-tenancy, i.e., it adequately handles the storage and processing needs of multiple BDWing technologies and users. Streaming applications, like the one discussed in the previous section, typically require a nearly constant amount of CPU and memory for long periods of time. Data arrives at the system continuously, thus it needs to assure that the workload has the adequate amount of resources.

A common setup, like the one evaluated in this chapter, would be a producer (e.g., Kafka), a consumer (e.g., Spark Streaming), a storage system (e.g., Cassandra and Hive), and a query and OLAP engine (e.g., Presto). At first glance, the first three components of this setup may seem to work perfectly fine. However, once one adds the query and OLAP engine, resource consumption can get significantly high, and the performance of the streaming application may suffer, because one did not choose the adequate trade-off between data timeliness and resource capacity. Take as an example Figure 7.12. If carefully observed, in certain periods of time coinciding with the time interval when Presto queries are running, there is a significant increase in the processing time of the micro batches, which consequently causes an increase in the scheduling time of further micro batches.

In this case, this happens because there is not enough resource capacity in the current infrastructure to handle the processing demands of Spark Streaming, Cassandra, and Presto running simultaneously. In these periods of time, these technologies are mainly racing for CPU usage, and the initial micro batch interval of 10 seconds is not enough to maintain the demands of the streaming application. Again, these insights bring us back to the previously discussed trade-off: either resource capacity is increased, in this case more CPU cores, or the micro batch time interval is raised, which inevitably affects data timeliness. In this benchmark, the queries are only executed each hour, thus



Timelines (Last 1001 batches, 1 active, 1000 completed)

Histograms

Input Rate
Avg: 2000.00 records/sec

Scheduling Delay (?)
Avg: 16 seconds 766 ms

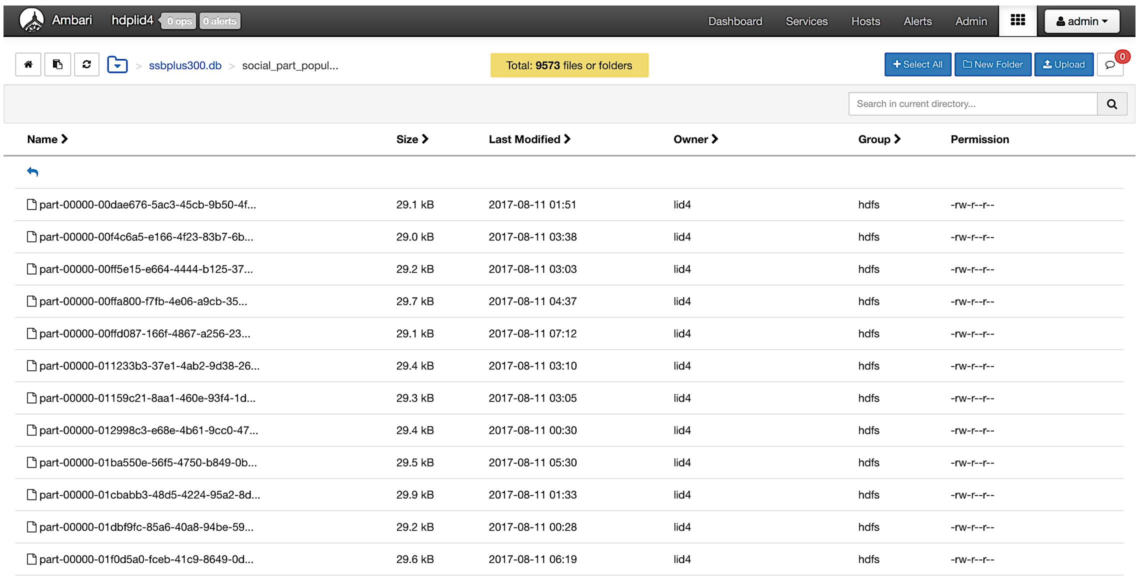
Processing Time (?)
Avg: 7 seconds 821 ms

Total Delay (?)
Avg: 24 seconds 604 ms

Figure 7.12. Spark Streaming monitoring GUI showing resource starvation when using Cassandra and Presto simultaneously. Adapted from (C. Costa & Santos, 2018).

the system is only affected during these periods. However, in real-world applications, users are constantly submitting queries, which makes this consideration hard to ignore.

Regarding the use of storage systems like Hive for streaming scenarios, as seen in the previous subsection, it has its advantages, namely reduced query execution times, since it can be considerably faster than Cassandra. Nevertheless, this performance advantage comes at a cost: as data volume increases, the number of small files stored in HDFS rises considerably, generating a significant load on the infrastructure. One small file is created for each RDD partition, in this case each 10 seconds, due to the chosen micro batch interval. In a matter of hours, the Hive table has stored thousands of small files (see Figure 7.13). The problem is that, as the number of files increases, HDFS metadata operations take more time, thus affecting the time it takes for Spark Streaming to save the data in the Hive table. A write operation in HDFS includes steps like searching for the existence of the file and checking user permissions (White, 2015), which with thousands of files can take longer than



Total: **9573** files or folders

Name	Size	Last Modified	Owner	Group	Permission
part-00000-00dae676-5ac3-45cb-9b50-4f...	29.1 kB	2017-08-11 01:51	lid4	hdfs	-rw-r--r--
part-00000-00f4c6a5-e166-4f23-83b7-6b...	29.0 kB	2017-08-11 03:38	lid4	hdfs	-rw-r--r--
part-00000-00ff5e15-e664-4444-b125-37...	29.2 kB	2017-08-11 03:03	lid4	hdfs	-rw-r--r--
part-00000-00ffa800-f7fb-4e06-a9cb-35...	29.7 kB	2017-08-11 04:37	lid4	hdfs	-rw-r--r--
part-00000-00ffd087-166f-4867-a256-23...	29.1 kB	2017-08-11 07:12	lid4	hdfs	-rw-r--r--
part-00000-011233b3-37e1-4ab2-9d38-26...	29.4 kB	2017-08-11 03:10	lid4	hdfs	-rw-r--r--
part-00000-01159c21-8aa1-460e-93f4-1d...	29.3 kB	2017-08-11 03:05	lid4	hdfs	-rw-r--r--
part-00000-012998c3-e68e-4b61-9cc0-47...	29.4 kB	2017-08-11 00:30	lid4	hdfs	-rw-r--r--
part-00000-01ba550e-56f5-4750-b849-0b...	29.5 kB	2017-08-11 05:30	lid4	hdfs	-rw-r--r--
part-00000-01cbabb3-48d5-4224-95a2-8d...	29.9 kB	2017-08-11 01:33	lid4	hdfs	-rw-r--r--
part-00000-01dbf9fc-85a6-40a8-94be-59...	29.2 kB	2017-08-11 00:28	lid4	hdfs	-rw-r--r--
part-00000-01f0d5a0-fceb-41c9-8649-0d...	29.6 kB	2017-08-11 06:19	lid4	hdfs	-rw-r--r--

Figure 7.13. Thousands of small files created in HDFS (Hive's storage backend) when using Spark Streaming.

usual. Nevertheless, one needs to highlight that this problem can be solved by applying an adequate partition scheme to streaming Hive tables, e.g., partitioning by "date" and "hour", which creates a folder structure containing fewer files in each folder and, therefore, reducing the time to execute metadata operations (Vale Lima, Costa, & Santos, 2019).

At this point, the system can be under intensive load and the Spark Streaming application queues hundreds of micro batches. Micro batches are queued when the Spark application cannot process them before the defined micro batch interval, in this case 10 seconds. Again, this predefined micro batch interval is not able to assure that the data gets processed before the next micro batch, and the performance of the streaming application is compromised. In Hive's case, this is much more severe than the concurrency issue shown by running Cassandra and Presto simultaneously. In Hive's case, even increasing resource capacity is not the best solution, and one should prefer higher micro batch intervals, which will consequently create bigger files. Moreover, the inter-storage pipeline is significantly relevant to periodically consolidate these small files into bigger files, or moving them into another analytical object which contains historical data. It must be remembered that Hadoop prefers large files, which are then partitioned, distributed, and replicated as blocks.

Finally, and taking into consideration the phenomena discussed above, the inter-storage pipeline and CPE workloads should also be carefully planned when streaming applications are using the cluster's resources. These operations can be really heavy on CPU and memory, and can unexpectedly cause resource starvation, as seen with Presto and Cassandra running simultaneously. Practitioners should not take this lightly, and Linux Cgroups, YARN queues, and YARN CPU isolation can be extremely useful to assure that the current infrastructure is able to properly assure a rich, complex, and multi-tenant environment such as a BDW. These techniques assure that resources are adequately shared by multiple applications, by assigning portions of the resources according to the expected workloads. Moreover, practitioners should evaluate their requirements regarding data timeliness, and avoid small micro batch intervals for streaming applications when they are not needed, as well as avoiding the execution of really complex inter-storage pipelines or CPE workloads when business users are intensively using the BDW. More resource capacity may not always be the most efficient solution to the problem, since even in commodity hardware environments, buying hardware always comes at a cost, while making some of these changes may increase efficiency without any relevant implication.

7.4 SQL-on-Hadoop Systems Under Multi-User Environments

In real-world environments, the BDW will not be queried by a single business user. The system may have to support several decision-makers at different organizational levels. Single-user benchmarks allow us to understand the raw performance of a certain system without considering concurrency. However, one should expect to design and implement a BDW with the goal of supporting several users simultaneously.

Since the proposed approach promotes the use of SQL-on-Hadoop engines as the frontend for querying and OLAP, the way these systems handle concurrency is a key factor for a BDW that adequately supports several users. Consequently, this section discusses the performance of Hive and Presto in a SF=30 concurrent workload, wherein four users execute the 13 SSB+ batch queries simultaneously. In this case, the smaller SF=30 was used to create an adequate balance between the concurrency requirements, the size of the dataset, and the available infrastructure (subsection 7.1.2). In this evaluation, the SQL-on-Hadoop systems were left to their default configurations.

Looking at Table 7.1, Presto emerges as the fastest engine. However, since one is looking into multi-user efficiency, execution time may not be the only metric to take into consideration. Obviously, if Presto is the fastest engine to retrieve the results to the concurrent users submitting the queries, it can perfectly be considered the most adequate system. The problem is that, in single-user workloads, Presto already tends to be significantly faster than Hive, which gives it a severe advantage in this multi-user test. Taking a closer look at Table 7.1, one of the most interesting insights is Hive's increase in execution time from single-user to multi-user queries. Despite its inferior performance in single-user workloads when compared to Presto, Hive is the system that gets less affected by having multiple users submitting queries simultaneously. An increase below the 3x mark means that, in a concurrent environment with four users, the system is able to execute the query faster than executing the same query four times in a single-user environment.

These results aim to provide an overview regarding the performance of SQL-on-Hadoop systems under concurrent environments. Generally, it can be concluded that SQL-on-Hadoop systems are able to handle concurrent queries on relatively modest hardware, such as the one used in this work.

Table 7.1. Multi-user SSB+ workload SF=30.
Multi-user execution (M - in seconds); single-user execution (S - in seconds).

Queries	Hive (S)	Hive (M)	Presto (S)	Presto (M)
Q1.1	23	42	4	20
Q1.2	24	45	5	20
Q1.3	24	43	4	18
Q2.1	26	66	3	18
Q2.2	36	99	3	13
Q2.3	24	80	4	13
Q3.1	28	63	4	17
Q3.2	28	64	3	16
Q3.3	25	57	4	14
Q3.4	25	63	5	17
Q4.1	28	85	5	20
Q4.2	28	69	5	20
Q4.3	28	64	5	19
Total	347	839 (Increase: 1.4x)	52	225 (Increase: 3,3x)

Obviously, all of the configurations were not changed, which does not always represent the best setup for these systems, especially since concurrency configurations are one of the aspects that may need some tuning to achieve optimal performance in production systems. However, performing a benchmark using the vanilla version of the systems also means that any kind of over-fitting does not occur and they are on the same level, without any misconfigurations. Depending on the SQL-on-Hadoop system practitioners end up choosing for their BDWs, it is advisable and necessary to read the documentation and adjust any relevant configuration. It must also be remembered that each version of the systems brings a number of improvements, and if concurrency performance is a critical factor to choose the SQL-on-Hadoop system, then an on-site benchmark may be needed before making any decision.

Chapter 8. Big Data Warehousing in Smart Cities

This chapter discusses the implementation of the SusCity BDW in the context of smart cities (C. Costa & Santos, 2017c; Monteiro, Costa, Pina, Santos, & Ferrão, 2018; SusCity, 2016), which is built upon the proposed models and methods as a demonstration case. In the context of smart cities, vast amounts of heterogeneous data are constantly being produced by an extensive network of interconnected things, including smartphones, smart meters, temperature sensors, noise sensors, smart appliances, location sensors, among many others. Moreover, there are also other data sources like the cities' transactional database systems, geospatial files, census data, and data provided by private companies responsible for certain city services. This phenomenon is typically associated with the concepts of IoT and Big Data (Jara, Bocchi, & Genoud, 2013). Consequently, smart cities are seen as rich BDWing contexts, given these extensive set of data sources and their relevance in the cities' decision-making process.

8.1 Logical Components, Data Flows, and Technological Infrastructure

In the context of smart cities, an adequate BDWing approach is crucial to support the decision-making process at scale, complying with the characteristics of a BDW. Figure 8.1 presents the SusCity BDWing architecture, following the proposed approach. The logical layer helps researchers and practitioners understanding the logical components of the system and how data flows throughout these components. It uses the taxonomy of the proposed approach, partially inherited from the NBDRA (NBD-PWG, 2015), since the lack of concepts standardization can be an issue in Big Data research, as discussed in Chapter 2 and Chapter 3. The technological infrastructure focuses on the technologies used for instantiating the logical components and on the infrastructure in which these technologies are deployed (detailed later in subsection 8.1.2).

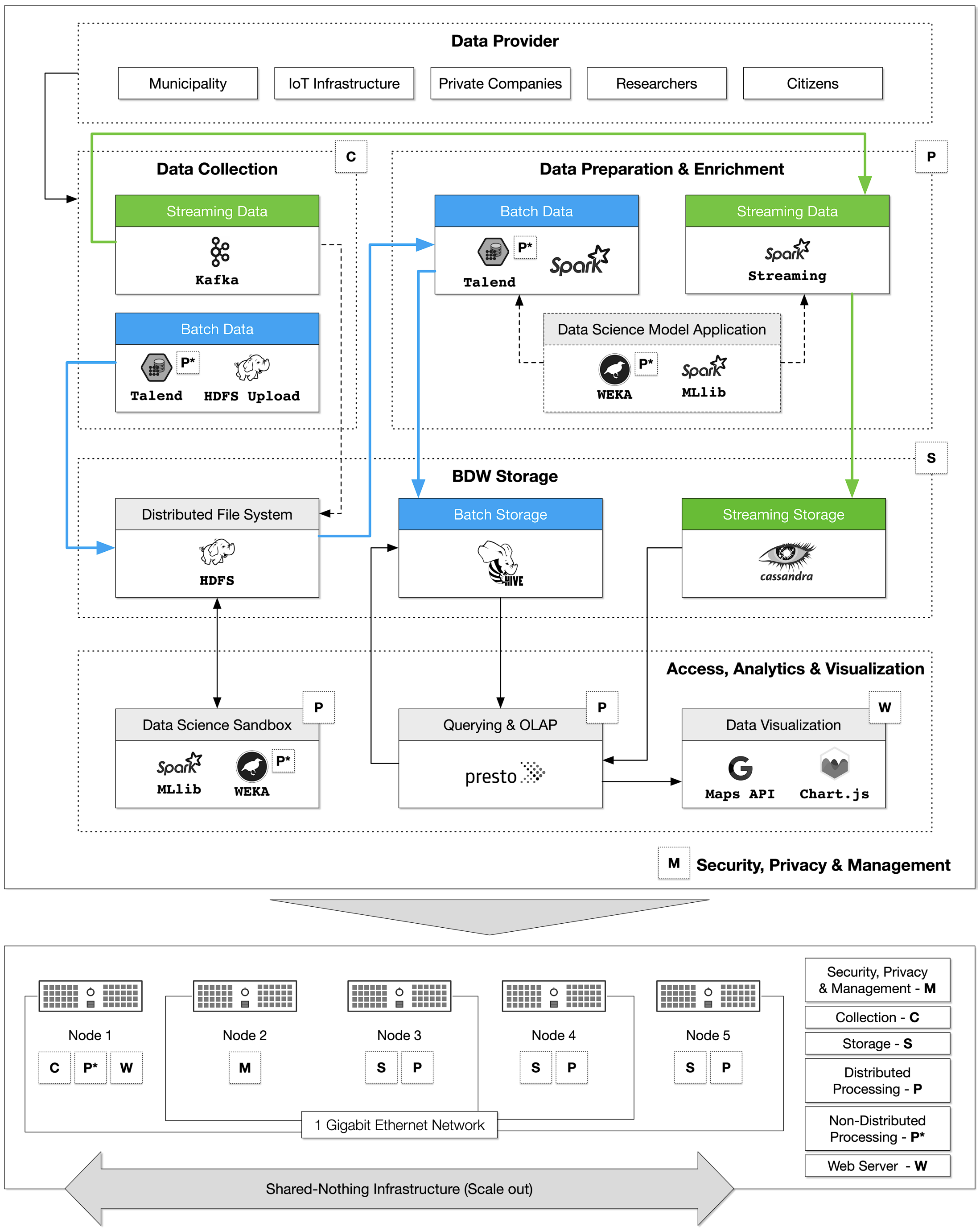


Figure 8.1. The SusCity BDWing architecture. Adapted from (C. Costa & Santos, 2017c).

8.1.1 SusCity Architecture

Regarding the logical layer, the first component is the data provider, making data available for further storage and processing. In a typical smart cities context, which is the case of the SusCity research project (SusCity, 2016), the data provider component can include several actors:

- Municipality – the municipality itself is able to make available several data sources relevant for analytical tasks, including buildings information or geospatial representations of the city's infrastructures, for example. The city's transactional systems are also valuable data sources;
- IoT infrastructure – includes different kinds of sensors reporting electricity consumption, temperature, noise, and mobility patterns, for example. This data is significantly relevant to understand events and real-time patterns in the city;
- Private companies – the city's infrastructures are not always public and, therefore, interactions with private companies are of major relevance in smart cities, in order to collect historical energy consumption, buildings certificates, water consumption, census data, among many other data sources;
- Researchers and citizens – research projects being conducted in the city are a relevant data source for the BDW, including simulation data regarding different phenomena in the city (e.g., buildings' energy efficiency and mobility patterns), or any other relevant insights corresponding to scientific studies impacting the city. Moreover, citizens engaged in the initiatives promoted by researchers or by the municipality can provide useful data for the decision-making process, such as personal energy consumptions, mobility patterns, and service consumption habits.

Taking into consideration the data sources presented in Figure 8.1 and discussed above, data may arrive at the BDW via batch or streaming mechanisms. For data arriving in batches, one uses Talend Open Studio for Big Data (Talend, 2017) and a HDFS client to upload it to the distributed file system. Before any preparation and enrichment process, data is first uploaded to HDFS, since raw data may serve further analytical purposes (e.g., training and testing data mining models) or may be useful for disaster recovery, in case some problems occur in the storage component. HDFS is capable of handling large amounts of structured, semi-structured, and unstructured data, distributing them across several nodes in the cluster for further preparation and enrichment. For data arriving in a streaming fashion, Kafka (Kafka, 2018) is used to assure highly scalable and robust data collection. Periodically, one can optionally move data from Kafka to HDFS, using systems such as LinkedIn's Gobblin (Qiao et al., 2015), in cases where streaming raw data is also useful for further purposes.

To prepare and enrich batch data, the SusCity BDWing architecture takes into consideration the volume of data. If the dataset being processed fits in the constraints of non-distributed technologies, Talend Open Studio for Big Data is used to prepare and enrich data, since it offers a wide set of processing components (e.g., filtering, aggregation, joins, and type parsing) in a user-friendly graphical interface. However, when the volume of the dataset requires distributed processing, Spark (Shanahan & Dai, 2015) is used to accomplish the preparation and enrichment tasks. In streaming scenarios, one can use Spark Streaming to process data as it arrives at the BDWing system. Previously trained data mining models can also be applied in this phase, using WEKA (Hall et al., 2009) for small-scale algorithms (e.g., classification/regression of previously aggregated data or time series forecasting problems) and Spark MLlib for large-scale algorithms, i.e., when the training set contains vast amounts of very detailed data, not previously aggregated. Relevant data mining use cases in smart cities contexts may include forecasting and segmenting energy consumption (C. Costa & Santos, 2015); predicting attendance at the city's events; or segmenting buildings according to their characteristics and energy efficiency. The same logic applies to unstructured data, since text mining algorithms, for example, can also be applied using WEKA, Spark or any other suitable technology, in order to extract structured patterns to be further stored in the BDW.

Once the data is prepared and enriched, it is stored in the storage component. The storage component consists of three subcomponents. The distributed file system (HDFS) acts as a staging area and sandbox, storing raw batch and streaming data, and temporary files needed in the data science sandbox component. It is a crucial component to assure a flexible storage capable of handling data variety from several sources. HDFS is also the underlying storage system for Hive, the technology used in the batch storage component of the proposed architecture. Hive is a DWing system on

Hadoop, frequently mentioned as the de-facto SQL-on-Hadoop solution. In the SusCity BDWing architecture, Hive tables stored as ORC files (Huai et al., 2014) are used to store large amounts of structured data, using the proposed data modeling method (the SusCity data model is presented in section 8.2). In this work, Hive only stores data arriving in batches, since it is mainly designed for fast sequential access to data. For fast random access, Cassandra is used as the NoSQL database supporting the streaming storage component, since one can assure hundreds or thousands of concurrent writes frequently required by typical streaming applications. Previous benchmarks reveal that Cassandra is a suitable distributed database for intensive random read and random write scenarios (C. Costa & Santos, 2016b). Moreover, Cassandra tables are also modelled according to the data modeling method proposed in this work (section 8.2).

Regarding the use of Hive as a streaming storage system, without several concerns (e.g., efficient compaction techniques), Hive tends to generate several small files in HDFS, which can become a bottleneck in the system. Despite the current advancements regarding Hive's transactions (Apache Hive, 2018), Hive's suitability for a large number of concurrent and continuous writes needs to be tested in prototype or production systems. As mentioned, Hive is a DWing system on Hadoop mainly used to scale OLAP applications. Since NoSQL databases are mainly designed to scale OLTP applications (Cattell, 2011), they are not as effective and efficient as Hive in sequential access scenarios, typically required in OLAP applications, as can be further seen in section 8.3. Therefore, the SusCity BDWing system considers these trade-offs and maintains two separate storage technologies for batch and streaming data. As techniques and technologies evolve and stabilize, the same technology may be able to adequately support both scenarios, as discussed in section 4.2.

The goal of the BDW is to support analytical tasks. Consequently, the access, analytics, and visualization component is crucial to deliver adequate insights for data-driven decision-making processes. The querying and OLAP component, using Presto, assures the communication between the batch storage, the streaming storage, and the data visualization component. Presto was open sourced by Facebook, and it is seen as a SQL-on-Hadoop system providing low-latency query execution over large amounts of data (Presto, 2016). In fact, it is more than a SQL-on-Hadoop system, since it can provide a SQL interface to a vast set of storage technologies besides Hive (Hadoop), including NoSQL databases like Cassandra and MongoDB. Therefore, in the proposed architecture, Presto is used to query Hive tables and Cassandra tables. As previously discussed, since Cassandra is less efficient than Hive for fast sequential access (section 8.3), one also uses Presto to transfer data between Cassandra and Hive, avoiding the accumulation of vast amounts of historical data in the Cassandra tables. For more complex queries that surpass the interactivity threshold defined for the SusCity data visualization component (10 seconds), Presto is also used to create materialized views stored in Hive tables.

Although several improvements have been made in Hive, such as the Tez execution engine (Floratou et al., 2014; Huai et al., 2014), Presto achieves significantly faster execution times when querying Hive tables (as can be seen in Chapter 7), reason why it is used as the querying and OLAP engine in the proposed architecture. Interactive query execution is one of the main requirements of the SusCity data visualization component, in order to engage users through a responsive interface. Therefore, the data visualization component (discussed in section 8.4) uses Presto to submit SQL queries to the batch and streaming storage systems. Presto is also able to combine data from these two components using a single query (e.g., joins and unions), which is of major relevance to combine historical and streaming data into a unified view of the data.

Although some benchmarks demonstrated that interactive SQL-on-Hadoop systems similar to Presto (e.g., Impala) may struggle with datasets that do not fit into memory (Floratou et al., 2014), one did not feel the need to use the Hive execution engine in the SusCity project, since Presto was able to execute all the workloads requested by the SusCity testbed, processing several Gigabytes of data from energy grid simulations, buildings information, geospatial files, historical energy consumption data, and more than one hundred smart meters. However, if certain scalability issues arise, the Hive execution engine is always available for more demanding workloads. Scalability will certainly not be an issue, since Presto is being used at Facebook to perform queries over its Petabyte-scale Hive DW, thus it is possible to scale the cluster to accommodate growing data in a smart cities context.

Still concerning analytical tasks, the application of data science models (e.g., data mining and text mining models) is only possible with an adequate sandbox where data scientists can explore the data, training and testing models to support their hypotheses (C. Costa & Santos, 2017b). Therefore, the proposed architecture includes a dedicated component, named data science sandbox, which interacts with HDFS. Since raw batch and streaming data can be stored in HDFS, data scientists can interact with this data to produce models capable of extracting patterns and making predictions when new data arrives at the preparation and enrichment component. WEKA and Spark are the driving forces for this purpose, as previously discussed.

Security, privacy, and management is a relevant component in the SusCity BDWing architecture. There are certain Hadoop-related technologies that can be used to assure a secure environment that is properly managed. In this work, Kerberos is used to provide a secure authentication protocol in Hadoop. To assure an extra-layer of security and privacy, Ranger can be used to deploy rigorous authorization policies, defining which users have access to certain files or tables (Hortonworks, 2016). Regarding Cassandra's security, TLS/SSL encryption can be used for client-to-node or node-to-node communications. Cassandra also makes available simple password authentication and an internal authorization model. In a smart cities context, data privacy is a main concern and, therefore, whenever possible, one encourages the anonymization of sensitive data before storing it in the BDW (C. Costa & Santos, 2016a). Finally, Ambari can be used to manage and monitor the Hadoop components deployed in the cluster (Apache Hadoop, 2018).

8.1.2 SusCity Infrastructure

All the components and technologies discussed in the previous subsection are deployed in 5 commodity hardware machines installed on-premises, which have been capable of supporting the workloads demanded by the SusCity testbed. In this demonstration case, each machine has 16GB of RAM, Intel i5 quad-core CPUs, and 500GB 7200rpm hard disks (except node 1, which has an Intel i7 quad-core CPU and a 256GB SSD drive). All the machines are connected using a 1 Gigabit Ethernet switch and 5 Ethernet CAT6 cables, as Hadoop clusters should be deployed using at least a 1 Gigabit Ethernet network (Shvachko et al., 2010). The use of Big Data technologies like Hadoop, Spark, and

Cassandra, which rely on commodity hardware and shared-nothing infrastructures, allows scaling the cluster as data volume increases and workloads become more demanding.

Due to resource limitations, Hadoop and Cassandra nodes are co-located (node 3, node 4, and node 5). Presto and Spark are also deployed in these nodes for co-located processing. Distributed processing technologies should be co-located with storage nodes, since in Big Data environments the processing should be brought closer to the storage, in order to avoid moving large amounts of data through the network (C. L. P. Chen & Zhang, 2014). The collection technologies (Kafka, Talend Open Studio, and HDFS client), non-distributed processing technologies (Talend Open Studio and WEKA), and the Web Server making available dashboards with Chart.js (Chart.js, 2017) and the Google Maps API (Google Maps, 2017) are all deployed within node 1, again due to resource limitations. Ideally, in a production environment, these should have dedicated nodes and Kafka should be distributed across several nodes in the cluster. Node 2, besides being the Hadoop NameNode, assures several tasks related to the security, privacy, and management of the cluster, containing the Kerberos Key Distribution Center and Ambari.

8.2 SusCity Data Model

As seen in section 4.3, the main concept in the proposed data modeling method is the analytical object, representing a subject of interest to be analyzed. Typical analytical objects in smart cities may include: general indicators about buildings; buildings energy consumption; losses in the energy grid; indicators about the nodes in the energy grid; and the energy consumption recorded by smart meters.

Making the analogy to traditional DWs, analytical objects have the same capabilities of fact tables. In contrast to fact tables, they typically are fully denormalized structures, in which all the attributes needed for analyzing the subject of interest are included in one single analytical object, without the need for dimension tables, avoiding constant and demanding join operations. Join operations in Big Data environments are costly (Floratou et al., 2014; Marz & Warren, 2015; NBD-PWG, 2015; H. Wang et al., 2011), since tables may store vast amounts of data. Joining several dimensions with fact tables for each query can be significantly resource-demanding (see subsections 4.3.2 and 4.3.3 for concepts focusing on efficient dimensional patterns and join operations, and subsection 8.2.1 for their specific application in the SusCity BDW).

Guided by the approach proposed in this work, each analytical object of the SusCity data model contains two types of attributes: descriptive attributes (top half of the analytical objects in Figure 8.2) and analytical attributes (bottom half of the analytical objects in Figure 8.2). Moreover, outsourced descriptive families (see subsection 4.3.3) for each analytical object are also presented in Figure 8.2. Descriptive attributes support typical OLAP tasks by providing different perspectives for aggregations and filtering operations. These are analogous to the attributes of the dimension tables in traditional DWs, and allow the interpretation of the analytical attributes through different perspectives. Analytical attributes are an analogy to the facts in a traditional fact table, but with the particularity that they can not only contain facts (historical indicators), but also predictions derived from the application of data science models. Take as an example the analytical object *"buildings energy consumption"* in Figure 8.2, which contains a cluster defining the consumption behavior of each building and a forecast of its energy consumption (kWh) for the following days, information obtained using the WEKA's clustering and time series forecasting algorithms, as proposed by C. Costa and Santos (2015).

Descriptive and analytical attributes can store simple data types (e.g., integer, float, and varchar) or complex types (e.g., arrays, maps, and GeoJSON objects). The use of complex types to extend the capabilities of BDWs is detailed in subsection 8.2.2. Descriptive attributes are also relevant to define partition keys (PKs in Figure 8.2) and granularity keys (GKs in Figure 8.2). Certain descriptive attributes can be used to control certain aspects of data locality. In Hive, a partition key distributes the data throughout different folders according to the value of the attribute. In Cassandra, the partition key helps determining which node should be used to store/read the data, since it tries to evenly spread the data across different nodes in the cluster. There is no rigorous rule for defining partition keys, and one should evaluate the patterns of the queries and/or the refreshing rates of the analytical objects. For example, in the SusCity BDW, one uses the *"simulation scenario"* attribute for the partition key in the *"energy grid losses"* and *"energy grid nodes indicators"* Hive tables, since one loads the data in batches corresponding to yearly simulations for each stress scenario in the energy grid. Moreover, the *"simulation scenario"* is an attribute frequently used in the where clause of the queries referring these analytical objects. Regarding the *"smart meters records"* Cassandra table, it is possible to use *"sm id"* as the partition key, balancing the data throughout the nodes in the cluster according to the identifier of the smart meter. In Cassandra, the partition key is the first part of the primary key, which in this case is a compound key using *"sm id"* and *"record date"*. In Hive, one does not need to define primary keys.

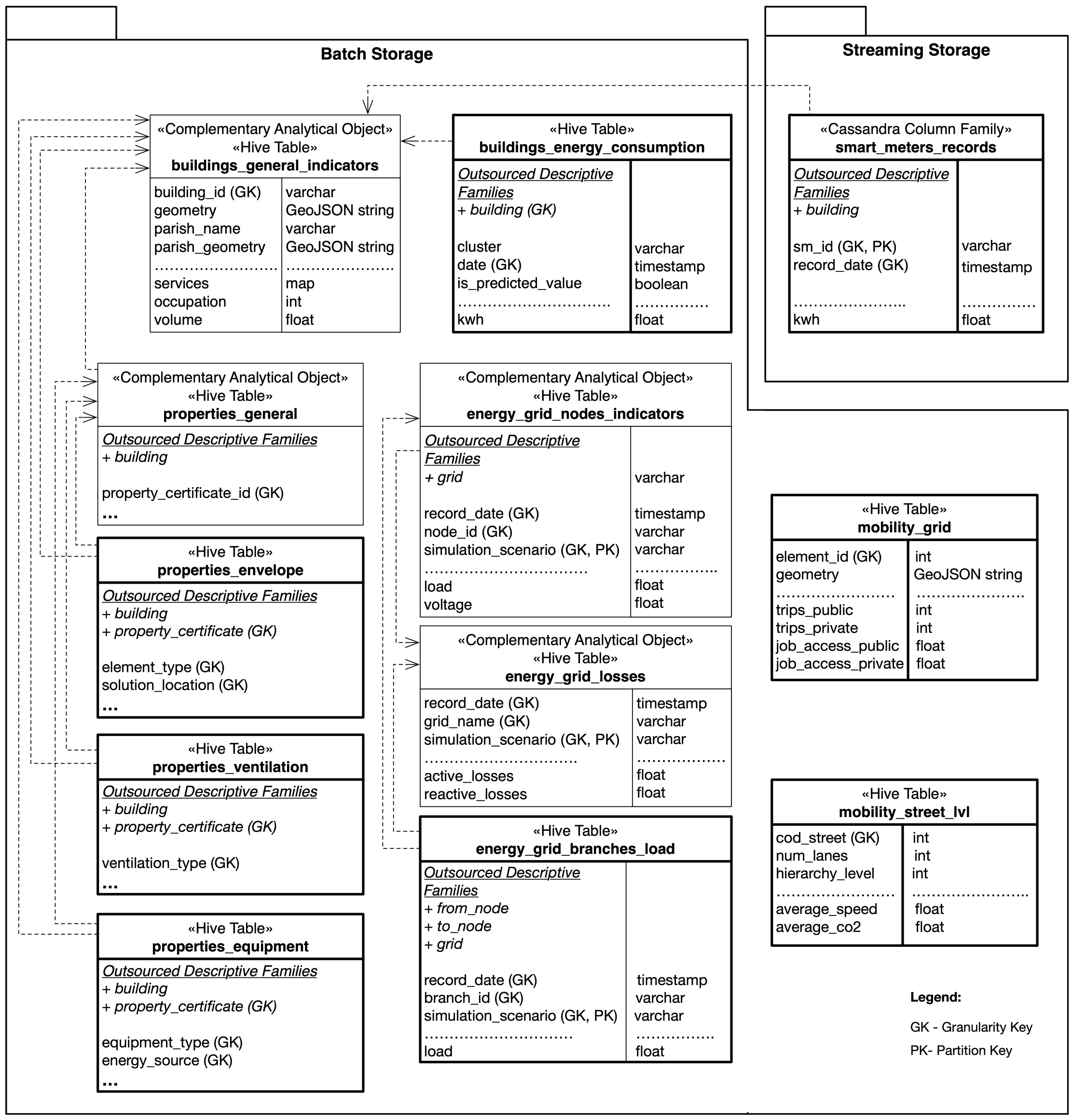

Batch Storage
Streaming Storage
«Complementary Analytical Object» «Hive Table» buildings_general_indicators
building_id (GK) varchar
geometry GeoJSON string
parish_name varchar
parish_geometry GeoJSON string
services map
occupation int
volume float
«Hive Table» buildings_energy_consumption
Outsourced Descriptive Families
+ building (GK)
cluster varchar
date (GK) timestamp
is_predicted_value boolean
kwh float
«Cassandra Column Family» smart_meters_records
Outsourced Descriptive Families
+ building
sm_id (GK, PK) varchar
record_date (GK) timestamp
kwh float
«Complementary Analytical Object» «Hive Table» properties_general
Outsourced Descriptive Families
+ building
property_certificate_id (GK)
...
«Complementary Analytical Object» «Hive Table» energy_grid_nodes_indicators
Outsourced Descriptive Families
+ grid varchar
record_date (GK) timestamp
node_id (GK) varchar
simulation_scenario (GK, PK) varchar
load float
voltage float
«Hive Table» properties_envelope
Outsourced Descriptive Families
+ building
+ property_certificate (GK)
element_type (GK)
solution_location (GK)
...
«Complementary Analytical Object» «Hive Table» energy_grid_losses
record_date (GK) timestamp
grid_name (GK) varchar
simulation_scenario (GK, PK) varchar
active_losses float
reactive_losses float
«Hive Table» properties_ventilation
Outsourced Descriptive Families
+ building
+ property_certificate (GK)
ventilation_type (GK)
...
«Hive Table» energy_grid_branches_load
Outsourced Descriptive Families
+ from_node
+ to_node
+ grid
record_date (GK) timestamp
branch_id (GK) varchar
simulation_scenario (GK, PK) varchar
load float
«Hive Table» properties_equipment
Outsourced Descriptive Families
+ building
+ property_certificate (GK)
equipment_type (GK)
energy_source (GK)
...
«Hive Table» mobility_grid
element_id (GK) int
geometry GeoJSON string
trips_public int
trips_private int
job_access_public float
job_access_private float
«Hive Table» mobility_street_lvl
cod_street (GK) int
num_lanes int
hierarchy_level int
average_speed float
average_co2 float
Legend:
GK - Granularity Key
PK- Partition Key

Figure 8.2. The SusCity BDW data model. Adapted from (C. Costa & Santos, 2017c) with extended content.

As can be seen in Figure 8.2, analytical objects can be joined together to answer certain queries. These join operations are optional, due to the fact that analytical objects can be modeled without any external references to other objects, which is the case for most of the analytical objects in the SusCity BDW. There is no need for declaring foreign keys at object creation, and analytical objects can be joined using all the attributes whose values match. This approach is detailed in subsection 8.2.1.

Complementing what was already mentioned regarding querying and OLAP, it is possible to use Presto to perform joins and unions between batch analytical objects (Hive tables) and streaming analytical objects (Cassandra tables), providing useful insights extracted from historical and real-time data, as can be seen in Figure 8.2. The size of the datasets being joined is of major relevance for an adequate query performance, and it should be taken into consideration. This is the reason why analytical objects should almost never be joined in their raw format. First, one needs to aggregate and filter (as much as possible) each analytical object involved in the join operation. The larger the inputs on each side of the join operation, the more complex and slower the query becomes. In this case, materialized views stored in Hive tables are significantly helpful for maintaining interactive response times in query execution and the responsiveness of the data visualization platform (see section 4.3.2 for more details on join operations and materialization processes).

Summarizing the main strategies for this modeling approach, one can highlight three major strategies: mainly use fully denormalized structures to avoid the cost of join operations in Big Data environments; the use of nested structures, which are not typically found in traditional modeling techniques, can provide more flexibility and performance advantages in specific scenarios; divide data flows and storage components into batch and streaming, as discussed and explored by Marz and Warren (2015), but that does not need to imply different data modeling strategies, as the proposed data modeling method demonstrates, since it can be used both for batch and streaming contexts.

8.2.1 Buildings Characteristics as an Outsourced Descriptive Family

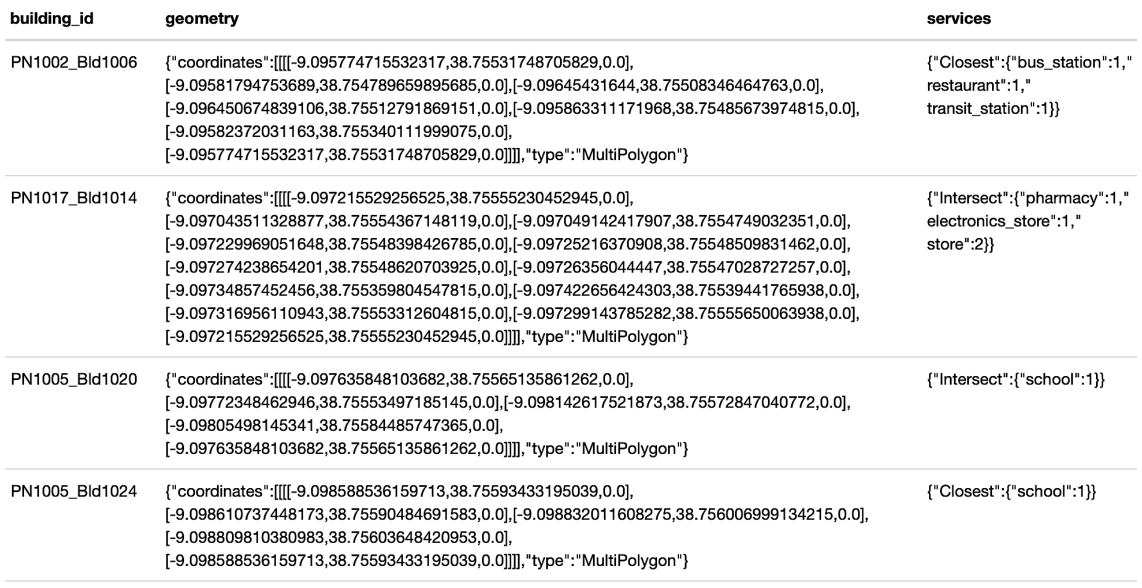
Looking at Figure 8.2, it can be seen that the Hive table *"buildings general indicators"* can be joined with the *"buildings energy consumption"* table*,* for example. This capability is useful to understand relationships between the characteristics of buildings and their energy consumption, such as: to what degree does the number of occupants influence the building's energy consumption?

These join operations are severely different from the join operations required between fact tables and dimension tables, since one only uses them in queries that relate different analytical objects, which is much less frequent than joining fact tables and dimensions for each query. Considering the SusCity BDW, the *"building id"* attribute is present in three analytical objects: *"buildings general indicators"; "buildings energy consumption";* and *"smart meters records"*. Instead of replicating the information about buildings in these three objects and creating unnecessary redundancy, taking into consideration that the *"buildings general indicators"* object is relatively small (around 60 000 records for the city of Lisbon), it can be easily joined with the other two objects, in order to answer specific questions. Consequently, one can outsource the buildings characteristics to the *"buildings general indicators"* complementary analytical object, and only place the *"building id"* in the *"buildings energy consumption"* and *"smart meters records"* analytical objects, as a link to the outsourced descriptive family (similarly to a foreign key in traditional DWs).

Nevertheless, when there is no need to relate energy consumption with buildings characteristics, all three objects are completely independent and are capable of answering different queries without relying on any join operations. Such flexibility is one of the strongest points of the proposed approach, which provides constructs and structured guidelines that practitioners can follow to solve specific problems, depending on the considerations and trade-offs previously discussed in section 4.3.

8.2.2 Nested Structures in Analytical Objects

In the SusCity BDW, complex types are used to store nested structures that will be interpreted afterwards by the data visualization component. For example, a large building can be associated with more than one service (e.g., laundry, supermarket, restaurant, and gym). Using a nested complex type like a map (e.g., HashMap), one can store this data using a single record associated with that building. Saving geometry objects in GeoJSON strings is also relevant for geospatial analysis in a smart cities context. Figure 8.3 exemplifies this data modeling technique, showing how the *"services"*



building_id	geometry	services
PN1002_Bld1006	{"coordinates":[[[[-9.095774715532317,38.75531748705829,0.0],[-9.09581794753689,38.754789659895685,0.0],[-9.09645431644,38.75508346464763,0.0],[-9.096450674839106,38.75512791869151,0.0],[-9.095863311171968,38.75485673974815,0.0],[-9.09582372031163,38.755340111999075,0.0],[-9.095774715532317,38.75531748705829,0.0]]]],"type":"MultiPolygon"}	{"Closest":{"bus_station":1,"restaurant":1,"transit_station":1}}
PN1017_Bld1014	{"coordinates":[[[[-9.097215529256525,38.75555230452945,0.0],[-9.097043511328877,38.75554367148119,0.0],[-9.097049142417907,38.7554749032351,0.0],[-9.097229969051648,38.75548398426785,0.0],[-9.09725216370908,38.75548509831462,0.0],[-9.097274238654201,38.75548620703925,0.0],[-9.09726356044447,38.75547028727257,0.0],[-9.09734857452456,38.755359804547815,0.0],[-9.097422656424303,38.75539441765938,0.0],[-9.097316956110943,38.75553312604815,0.0],[-9.097299143785282,38.75555650063938,0.0],[-9.097215529256525,38.75555230452945,0.0]]]],"type":"MultiPolygon"}	{"Intersect":{"pharmacy":1,"electronics_store":1,"store":2}}
PN1005_Bld1020	{"coordinates":[[[[-9.097635848103682,38.75565135861262,0.0],[-9.09772348462946,38.75553497185145,0.0],[-9.098142617521873,38.75572847040772,0.0],[-9.09805498145341,38.75584485747365,0.0],[-9.097635848103682,38.75565135861262,0.0]]]],"type":"MultiPolygon"}	{"Intersect":{"school":1}}
PN1005_Bld1024	{"coordinates":[[[[-9.098588536159713,38.75593433195039,0.0],[-9.098610737448173,38.75590484691583,0.0],[-9.098832011608275,38.756006999134215,0.0],[-9.098809810380983,38.75603648420953,0.0],[-9.098588536159713,38.75593433195039,0.0]]]],"type":"MultiPolygon"}	{"Closest":{"school":1}}

Figure 8.3. SusCity nested structures (example).

map and the *"geometry"* GeoJSON are stored. As can be seen in Figure 8.3, following the proposed approach, one can place the number of services by distance and type nested in the *"buildings general indicators"* analytical object, avoiding the need to create a new object to store the services for each building. This provides significant flexibility when building custom-made data visualizations (see subsection 8.4), avoiding the need to perform complex queries to join different tables.

8.3 The Inter-storage Pipeline

The need to transfer the data between storage components was already highlighted in section 8.1, but it will be quantitatively evaluated in this section. As mentioned, NoSQL databases are OLTP-oriented (Cattell, 2011), unlike Hive, which is an OLAP-oriented technology. Typically, OLTP systems relax sequential access efficiency for random access efficiency. Therefore, systems like Cassandra are adequate for the constant random write operations frequently demanded by the real-time collection of data from thousands of smart meters. However, these systems lack the efficiency to process (e.g., aggregate) large amounts of historical data, which is frequently demanded by OLAP queries. Table 8.1 presents the results from an experiment conducted in the infrastructure and testbed of the SusCity research project, evaluating the response times when submitting Presto queries to Hive

Table 8.1. Performance comparison between analytical objects stored in Hive and Cassandra. Based on (C. Costa & Santos, 2017c).

Query	Input Rows	Output Rows	Hive	Cassandra
Show the last 10 smart meters records.	~2.8 million	10	0.56s	3.08s
Calculate the average of kWh grouped by smart meter.	~2.8 million	214	0.56s	4.2s
Count how many records a certain smart meter contains.	~2.8 million	1	0.74s	0.98s

tables and Cassandra tables. As demonstrated in Table 8.1, given the same analytical object and the same amount of data, Presto OLAP queries on Hive Tables (ORC file format) perform significantly faster than the queries on Cassandra tables. This corroborates the statements previously mentioned and the decision of periodically moving historical data from Cassandra to Hive, maintaining only the most recent data in Cassandra. The periodicity of this data transfer depends on the specific requirements regarding interactivity in response times, the volume of data being stored, and the available infrastructure. Data can be transferred on an hourly, daily, weekly or monthly basis, for example.

8.4 The SusCity Data Visualization Platform

Throughout this chapter, one focused on the logical and physical layers of the BDW. In this section, one highlights some relevant use cases in which the data visualization platform can help the city's stakeholders in the decision-making process. As previously presented, the SusCity data visualization platform was developed using modern JavaScript libraries like the Google Maps API V3 and Chart.js. Obviously, since it is a Web-based platform, core languages are also present (HTML, CSS, and pure JavaScript), as well as other supporting JavaScript libraries like jQuery (jQuery, 2017). It is a platform purely based on a service-oriented architecture, using Java REST Web services to establish the communication between the JavaScript components of the platform and the querying and OLAP engine instantiated by Presto. Each query submitted to the BDW goes to this REST backend for an adequate modularity of the platform. Using this service-oriented and modular approach, it becomes easier to update or replace components and technologies, if that need arises in the future.

In this section, one will briefly present several dashboards developed in the SusCity research project, which can also be interesting applications for other smart cities initiatives. The following dashboards are just a few examples of the SusCity data visualization platform's capabilities, and the SusCity demonstration case itself considers other data sources and experiments (e.g., data mining and machine learning insights) that were not fully developed and implemented in the visualization platform. Furthermore, due to security and privacy issues, the visualizations illustrated in this section are built upon incomplete, omitted and/or changed testbed data and, therefore, results are not conclusive for any real-world based decision-making process.

8.4.1 City's Energy Consumption

The first dashboard (Figure 8.4) is based on the energy consumption of each parish in the city (2 parishes in the SusCity testbed). Decision-makers are able to understand the energy consumption in each parish and analyze the city's consumption by hour, time period (e.g., morning or afternoon) or by quarter. Users can interact with multiple parishes by clicking on them, revealing the energy consumption for specific parishes, and comparing it with the overall consumption of the city, with the goal of extracting insights regarding critical zones in the city, for example.

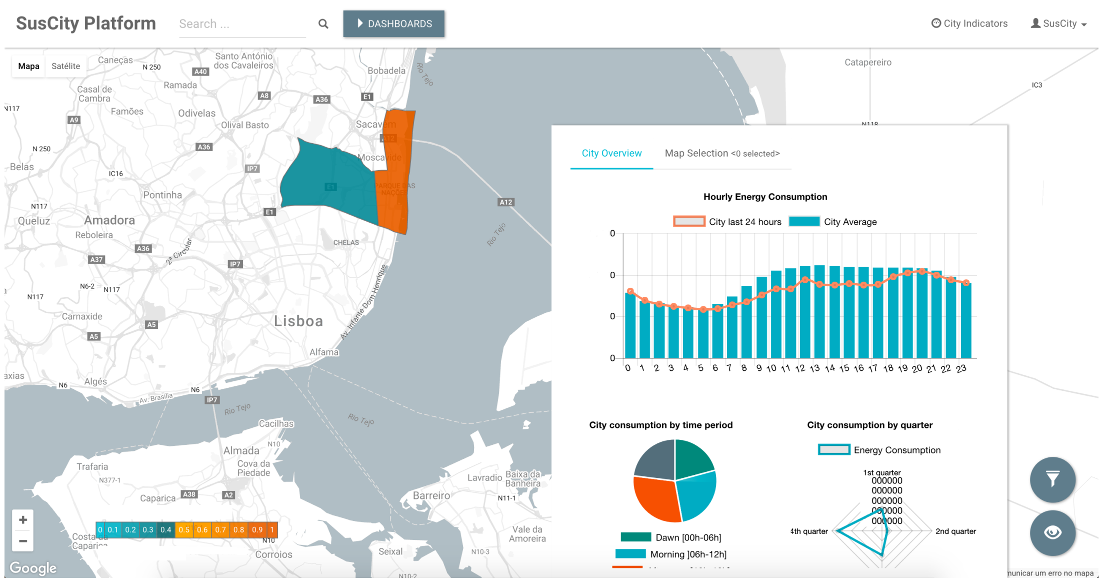
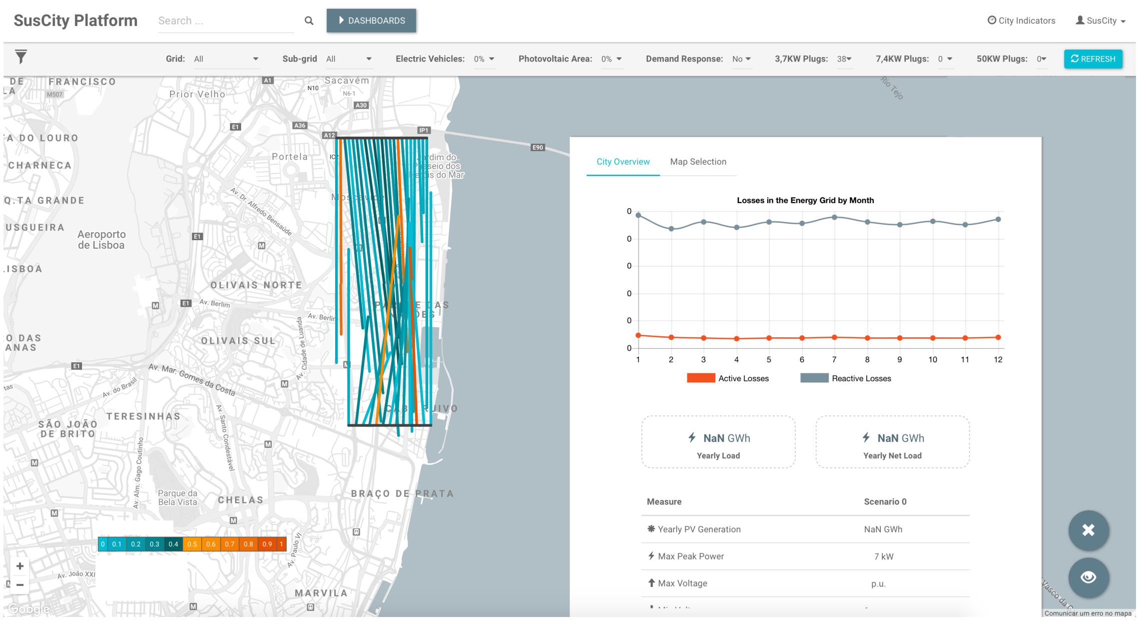


Figure 8.4. SusCity data visualization platform - energy consumption dashboard. Adapted from (C. Costa & Santos, 2017c).

Other dashboards in the SusCity platform (e.g., Figure 8.6) focus on the analysis of the buildings in the city, including information about their cooling/heating systems, energy consumption and efficiency, and envelope properties (e.g., window glass type and window materials). In the SusCity data visualization platform, one can also make available the predictive capabilities of the SusCity BDW, such as the segmentation (clustering) of buildings according to their energy consumption, and the respective energy forecasting for the next days or weeks, as conceptually explored in this work (section 8.2) and as also presented by C. Costa and Santos (2015).

8.4.2 City's Energy Grid Simulations

A dashboard to simulate and analyze stress scenarios in the energy grid can be significantly useful in the context of smart cities, as depicted in Figure 8.5. Each scenario corresponds to a set of input parameters (e.g., number of electrical vehicles, photovoltaic area, and number of charging stations) that may affect the behavior of the energy grid, such as energy losses, load, and maximum peak power. In the SusCity BDW, the results of the simulations for these scenarios are stored in analytical objects, as presented in section 8.2, and the data visualization platform can use the querying and



Grid: All | Sub-grid All | Electric Vehicles: 0% | Photovoltaic Area: 0% | Demand Response: No | 3,7KW Plugs: 38 | 7,4KW Plugs: 0 | 50KW Plugs: 0

Measure	Scenario 0
Yearly PV Generation	NaN GWh
Max Peak Power	7 kW
Max Voltage	p.u.

Figure 8.5. SusCity data visualization platform - energy grid simulation dashboard. Adapted from (C. Costa & Santos, 2017c).

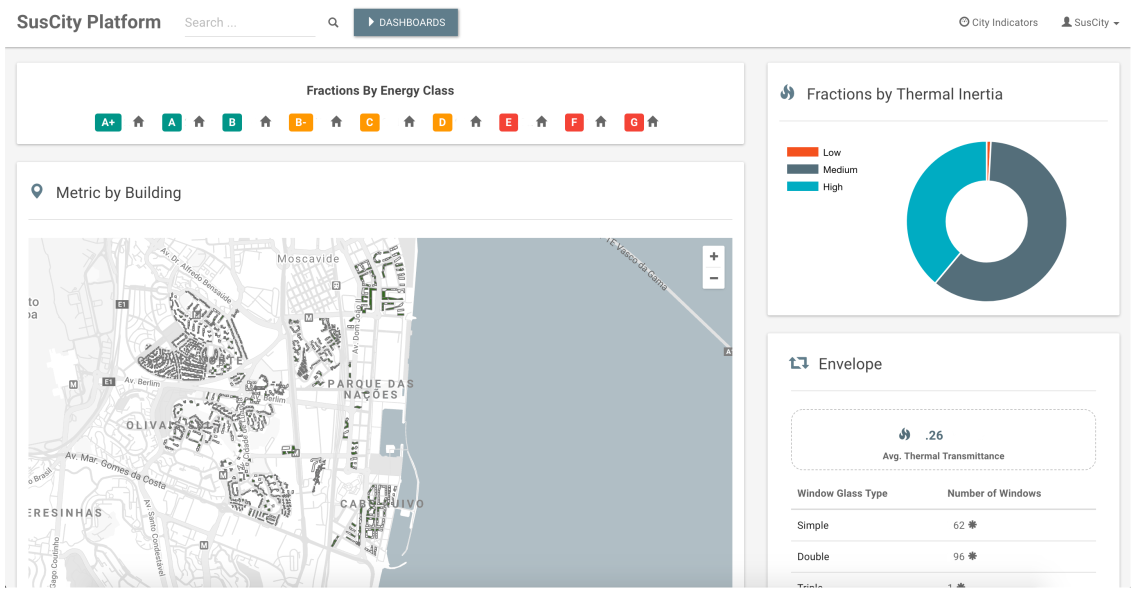
OLAP engine to extract and provide useful insights for stakeholders interested in the impact that certain initiatives have on the energy grid. Due to the modular and service-oriented nature of the SusCity data visualization platform, and the flexible and scalable SusCity BDW, one is able to provide dashboards for decision-makers, regardless of data volume, variety, and velocity, without being hold back by rigid data modeling techniques and complex data CPE pipelines.

8.4.3 Buildings' Performance Analysis and Simulation

Understanding the buildings' efficiency is a crucial aspect for a smart and sustainable city. One of the SusCity platform main focuses is the geospatial analysis of the buildings in Lisbon, based on an extensive set of characteristics, such as: geometry; construction; energy consumption and efficiency; envelope properties (e.g., type of window and type of window frame); heating and cooling systems in use; and occupation schedule.

The flexible data model and storage components of the SusCity BDW, i.e., the lack of a strict relational data model and the efficient use of GeoJSON objects, together with a rich API for geospatial analytics like the Google Maps API, provide an extensive set of analytical capabilities for the city's government. As can be seen in Figure 8.6, stakeholders can visualize the general distribution of the energy classes across Lisbon, the thermal inertia of the buildings, type of window and window frame, among other metrics georeferenced by building. Each chart is interactive and can be used as a filter to analyze how buildings are related to a certain property (e.g., metal window frame, double glass window, and low thermal inertia). As a consequence, the dashboard in Figure 8.6 is not only useful for the city's government, but it is also useful for private companies interested in promoting retrofitting initiatives to modernize buildings.

Similarly to the energy grid simulations (subsection 8.4.2), the SusCity BDW can also support simulations at the building level. Figure 8.7 demonstrates the use of simulation data to evaluate the impact of specific retrofitting initiatives (e.g., change the windows in 25% of the buildings in Lisbon). Besides analyzing the impact of retrofitting initiatives, decision-makers can also use the dashboard in Figure 8.7 to analyze the archetype of a specific building, among many other characteristics previously mentioned: geometry; construction; heating and cooling systems; and occupation schedule. Consequently, having a BDW whose data model facilitates the integration of a vast set of data sources, without rigid structures, is one of the main aspects that allows the development of these Big Data analyses in the context of smart cities.



Window Glass Type	Number of Windows
Simple	62
Double	96

Figure 8.6. SusCity data visualization platform - buildings analysis dashboard.

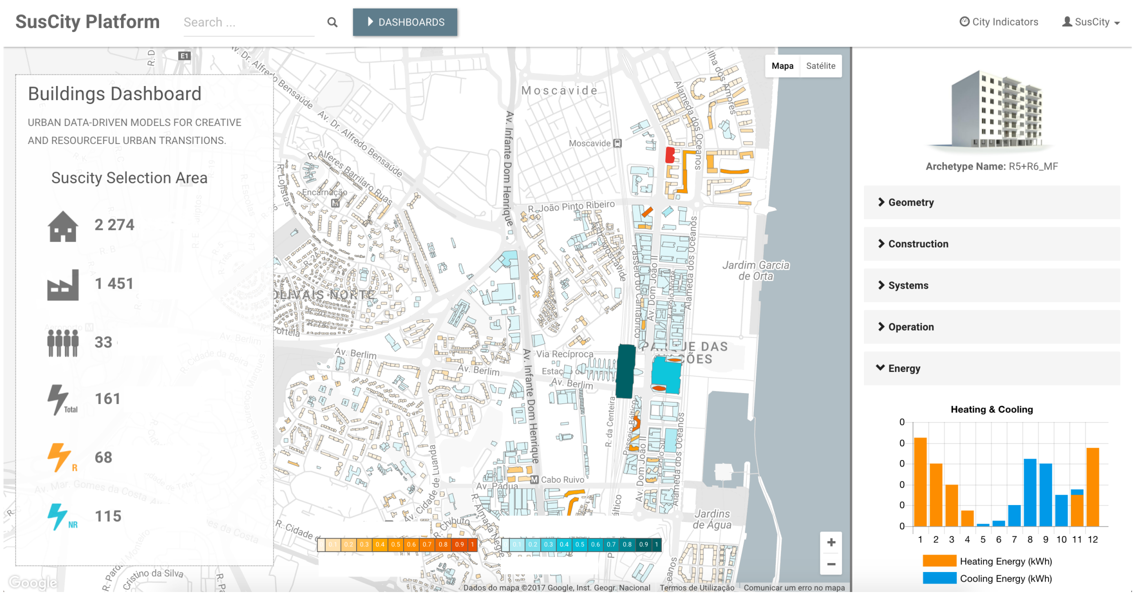


Figure 8.7. SusCity data visualization platform - buildings simulation dashboard. Adapted from (Monteiro et al., 2018).

8.4.4 Mobility Patterns Analysis

Studying mobility is a crucial aspect in a smart city. Understanding how people or goods travel within the city, or how citizens tend to use private or public transports, for example, is an interesting subject for several decision-makers, including the city's government and public/private transportation companies. Another interesting scenario is the footprint analysis of the city's streets, according to several indicators, such as CO² emissions or average speed.

Figure 8.8 focuses on the analysis of the city's mobility patterns, in order to foresee future initiatives to facilitate the use of either private or public transportation. The analysis goes according to the following steps:

1. The city is modelled as a grid with several sections;
2. Every section is colored in the map according to one of three indicators: number of daily trips; job transport accessibility; and average travel time;
3. By clicking in one section of the grid, decision-makers can understand how that section performs regarding the three indicators mentioned above.

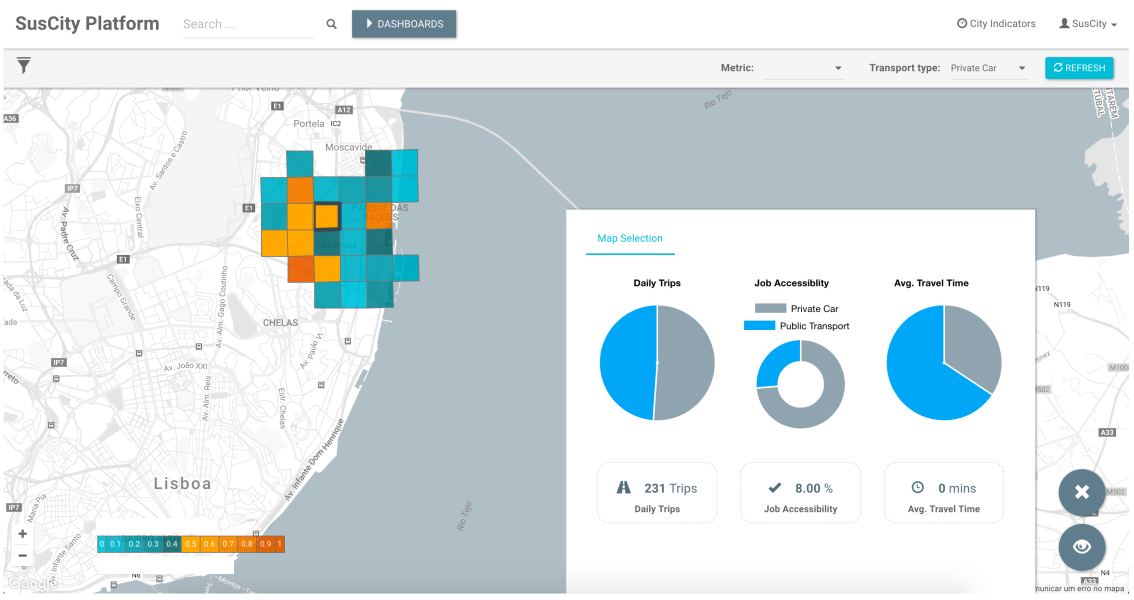


Figure 8.8. SusCity data visualization platform - mobility grid dashboard.

Analyzing the data at the grid level is interesting to understand the mobility behavior within different sections of the city. However, the flexible data model of the SusCity BDW and the geospatial capabilities of the SusCity data visualization platform also make possible the analysis of several indicators at the street level. To demonstrate these capabilities, Figure 8.9 presents several streets colored according to the ratio between average speed and maximum allowed speed, so that stakeholders can understand in which streets citizens tend to frequently overcome the speed limit. The analysis in Figure 8.9 focuses on the security concern within mobility patterns, but there are several other potential use cases for this kind of analysis, such as the identification of the busiest or more polluting streets, for example.

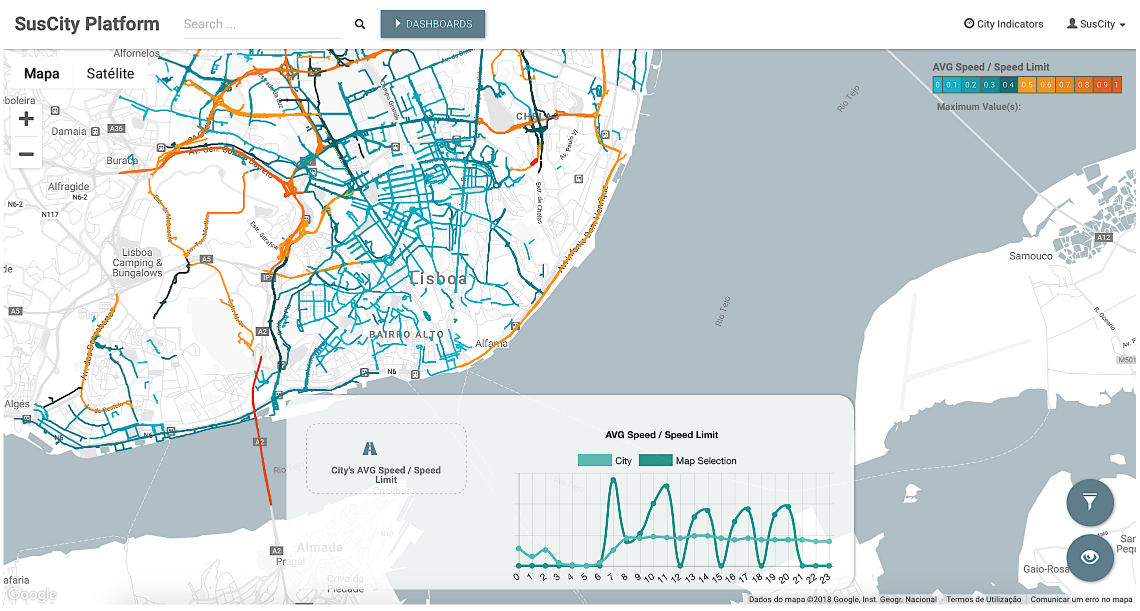


Figure 8.9. SusCity data visualization platform - mobility dashboard at street level.

Chapter 9. Conclusion

Throughout this work, Big Data was seen as a concept of major relevance in today's world, whose popularity has increased considerably during the last years. Areas like smart cities, manufacturing, retail, finance, software development, environment, digital media, among others, can benefit from the collection, storage, processing, and analysis of Big Data, leveraging unprecedented data-driven workflows, and considerably improving the decision-making processes. This new type of data is being defined not only by its characteristics (e.g., volume, variety, and velocity), but also by the limitations it imposes on traditional storage and processing technologies. Organizations seeking Big Data initiatives are facing many challenges, such as the lack of consensus in definitions, models, and architectures, and the difficulties regarding the Big Data life cycle design and implementation.

Since the DW concept has a long history as one of the most valuable enterprise data assets, in this work, it became relevant to study its role, design and implementation in Big Data environments. The concept of BDW is emerging as either an augmentation or a replacement of the traditional DW. Research in this topic is still in its infancy, and as Big Data is often synonymous of ambiguity, the same happens for the concept of BDW. After the literature review process, this work identified that the BDW can be defined using the following characteristics:

- Parallel/distributed storage and processing of large amounts of data, including fault-tolerance concerns;
- Scalability (accommodate more data, users, and analyses) and elasticity, using commodity hardware to lower the costs of implementation and maintenance;
- Flexible storage, including unstructured data;
- Real-time capabilities (stream processing, low-latency, and high-frequency updates);
- High performance with near real-time response;
- Interoperability in a federation of multiple technologies;

- Mixed and complex analytics (e.g., ad hoc or exploratory analysis, data mining, text mining, statistics, machine learning, reporting, visualization, geospatial analytics, advanced simulations, and materialized views).

Considering the state-of-the-art in BDWing, it can be concluded that there is no common approach to build BDWs, and there are innumerous Big Data technologies to choose from, each trying to stand out, which creates barriers in the design and implementation of Big Data solutions like BDWing systems, since most of the time these technologies' role is misunderstood, eventually overlapping each other. Current logical architectures and non-structured contributions only solve part of the problem by providing some general and relatively unstructured constructs and guidelines, but ambiguity regarding the BDW techniques and technologies that are more adequate for several contexts still prevails, mainly due to the lack of general-purpose, detailed, integrated, and adequately evaluated approaches.

Currently, the design and implementation of BDWs is mainly seen as a use case driven approach, instead of a data-driven one, which used to be the case for traditional DWs. Previously, data modeling was the primary concern, but, nowadays, practitioners are mainly concerned with trying to find the right technology to meet the demands of Big Data, leading to possible uncoordinated data silos. It would be a mistake to discard years of architectural best practices based on the assumption that storage for Big Data is not driven by data modeling (Clegg, 2015). Works related to the SQL-on-Hadoop movement are a suitable proof that the data structures known for a long time are still relevant, although modified and optimized. Obviously, unstructured data does not adequately fit into these structures, but, as this work demonstrated, there are data science techniques to extract value from it and subsequently fuel the BDW (e.g., data mining and text mining). Complex systems like BDWs require changes in different logical and technological components, data flows, and data structures, but this does not imply discarding the relevance of models and methods in favor of a use case driven approach.

Until now, there was no structured and general-purpose approach describing how to design and implement BDWs, with adequately evaluated models (representations of logical and technological components, data flows, and data structures), methods (structured practices), and instantiations (e.g., demonstration cases through prototyping and benchmarking). This scientific and technical gap served as the main motivation for this work, as, in one's modest opinion, the existing logical architectures, non-structured guidelines, best practices, and implementations in specific contexts, although relevant, did not provide a complete, general-purpose, detailed, and thoroughly evaluated approach that practitioners needed to design and implement BDWs according to their characteristics. The obvious gap between *"this is what a BDW should be"* and *"this is how you design and implement it"* motivated the proposal of this approach, an integrated, detailed, and general-purpose prescriptive contribution to design and implement BDWs, using models and methods that were adequately evaluated through different demonstration cases. That being said, one recognizes that the proposal of this approach was, at first glance, a possibly ambitious goal, but one also considers that the same was achieved, as practitioners and researchers have currently available a set of artifacts that can be used to build BDWs and to foster future research as techniques and technologies evolve. The following sections describe the undertaken work and achieved results, the main contributions to extend the existing knowledge barriers, and some prospects of future work.

9.1 Undertaken Work and Achieved Results

Considering the research goal and objectives of this doctoral thesis, one can state that the undertaken work and achieved results are divided into five main work fronts, namely the proposed approach for BDWing and the four demonstration cases: SSB+ Benchmark; SusCity BDWing system; data CPE workloads experimentation; and the demonstration of real-world BDW data models. Furthermore, these five work fronts took place on different activities of the research methodology (DSRM for IS), including design and development, demonstration, and evaluation.

The design and development of the proposed approach consisted in the creation of several models and methods to build BDWs. These models and methods were submitted to a continuous refinement process, wherein the several demonstration cases helped to not only facilitate the evaluation of the approach, but also to iteratively improve it. Finished this doctoral thesis, it can be highlighted that the following models and methods were developed:

1. A model of logical components and data flows (section 4.1), which can be used to understand the components that should be considered in the design of a BDW, how they interoperate, and how data flows through the system. The model is composed of several components related to BDW storage, processing, access, analytics, system administration and management, and security and privacy, detailing how they form a BDWing system that follows the constructs of Big Data standards like the NBDRA;
2. A method for collecting, preparing, and enriching batch and streaming data (subsection 4.1.2), so that practitioners can understand the different steps involved from data collection to data storage in BDWing contexts. The method not only clarifies batch and streaming data flows, but also details how data science models and insights can be incorporated into data CPE workloads, enabling predictive capabilities and allowing for the extraction of value from unstructured data (e.g., text, video, image);
3. A model of the technological infrastructure (section 4.2), resulting from an extensive research and development process that took place in this doctoral thesis, in order to identify and test several technologies suitable to instantiate the different components proposed in the model of logical components and data flows. The technological infrastructure model presents several alternatives that can be used to implement a BDWing system, including data CPE workloads, storage, querying and OLAP, data mining/machine learning, and data visualization technologies. Moreover, this model also provides some guidelines on how to deploy BDWing systems on cloud environments or on-premises;
4. A method for BDW data modeling (section 4.3), which, together with the aforementioned contributions, represents a relevant artifact to fulfil one of the main challenges in Big Data environments, i.e., the lack of standard data modeling contributions. The method presents several constructs, such as analytical objects; descriptive and analytical families; descriptive, factual, and predictive attributes (resulting from data science models and insights); nested attributes; granularity key; partition key; bucketing/clustering key; date, time, and spatial objects; materialized objects; complementary analytical objects; and outsourced descriptive families. The data modeling method provides a way of structuring batch and streaming data

using the same constructs regardless of the underlying technology supporting the storage system, providing an abstraction layer that practitioners can rely on to model BDWs supported by HDFS/Hive, NoSQL/NewSQL databases, Kudu, Druid, among other systems (see section 4.2).

The models and methods proposed in this work form a set of artifacts for the design and implementation of BDWing systems, and consider both structured, semi-structured, and unstructured batch and streaming data, providing adequate ways of collecting, storing, processing, and analyzing this data. Taking this into consideration, the proposed approach can be used by practitioners and researchers as a structured, integrated, and general-purpose approach that can be prescribed to solve several real-world BDWing problem, aiming to support structured analytics on Big Data environments while taking advantage of the several BDW characteristics. Furthermore, the approach was evaluated and refined using the several demonstration cases applied in this doctoral thesis, which provides a solid scientific and technical basis.

The first demonstration case consisted in the modeling of 6 BDW data models to solve potential real-world problems. In this demonstration case, one used several artificial datasets (e.g., Adventure Works, TPC-DS, and TPC-E), as well as other publicly available datasets, including the GitHub repositories dataset, the GDELT event database, and the open air quality API. For each dataset, one modelled a BDW data model using the proposed approach, which was significantly useful to thoroughly explain the several proposed models and methods. The approach by itself was developed to be relatively simple to follow, but one should take into consideration that a "rip and replace" approach like the one proposed in this work can be significantly disruptive and potentially confusing when designing and implementing specific parts of a BDWing system. Consequently, this demonstration case was performed in this doctoral thesis, in order to clarify when to apply specific guidelines, trying to provide different contexts and design decisions that practitioners may face in the future. Furthermore, this demonstration case, together with the considerations from the following data CPE demonstration case, the SSB+ Benchmark, and the SusCity demonstration case, was relevant to demonstrate the effectiveness and simplicity of the proposed approach, generating several BDW data models that, by avoiding some complexity related to traditional dimensional DWs (e.g., different types of dimensions, bridge tables, surrogate keys, SCDs, and late arriving dimensions), take less time to structure, fuel, maintain, and extend with new batch and streaming data (structured, semi-structured, and unstructured), which inevitably provides more storage flexibility and generally more performance, and accelerates the time from data collection to analysis. Consequently, these insights represent compelling reasons for the adoption of a BDWing strategy in organizations.

The second demonstration case was based on designing and implementing several data CPE processes focused on structured, semi-structured, and unstructured batch and streaming data, in order to cover different challenges related to collecting, preparing, and enriching data flowing to a BDW. Different data characteristics require different strategies, reason why this demonstration case was crucial to provide adequate examples to practitioners, showing the effectiveness of the proposed data CPE method. This demonstration case also highlighted the complexity differences between traditional ETL processes and data CPE workloads based on the proposed approach, namely:

- The use of denormalized structures allow for much simpler processes when the underlying data source is already flat or nested, such as sensor readings, NoSQL databases, Excel/CSV files, XML/JSON files, Web APIs, among many others sources frequently seen in Big Data environments. Not having to develop and maintain complex workloads to fuel several types of dimension tables/concepts (e.g., mini dimensions, shrunken dimensions, junk dimensions, bridge tables, late arriving dimensions), as well as avoiding the need to perform constant surrogate key lookups while loading a fact table is definitely a compelling advantage, especially in Big Data environments wherein one is focusing on accelerating the time to insight, instead of spending a significant amount of time trying to model and maintain the BDW. Nevertheless, if the underlying source is already relational, the contrasting phenomenon occurs, i.e., one may need to perform several join operations to fuel a certain denormalized analytical object;
- The lack of dimension tables also means that streaming scenarios are possible without complex operations like surrogate key lookups, or complex concepts such as SCDs or late

arriving dimensions. The descriptive attributes of an immutable analytical object behave like an SCD type 2 scenario, in which each record is associated with the current values of the descriptive attributes. When using outsourced descriptive families and complementary analytical objects, practitioners can consider different updating approaches, as discussed in subsections 4.3.3 and 5.2.1, in order to overcome some challenges that may be relatively similar to SCDs and late arriving dimensions.

The third demonstration case consisted in developing and executing an extension of the SSB benchmark (O'Neil et al., 2009), the SSB+. This benchmark served the purpose of evaluating several design and implementation guidelines of the proposed approach, in terms of effectiveness and efficiency (e.g., latency and resource usage), using as baseline, when appropriate, a star schema DW. The results provided in this work demonstrated that, generally, a fully denormalized analytical object is able to outperform a star schema throughout different SFs (some of them exceeding the amount of available memory), different SQL-on-Hadoop engines, and different descriptive attributes cardinality (dimension tables size), which means that even in contexts wherein dimension tables were relatively small to fit into memory (allowing efficient map/broadcast joins), a fully denormalized analytical object was more efficient (faster execution times and less CPU usage and memory dependability), surpassing the need for constant join operations between the fact table and the corresponding dimension tables. Analytical objects were also generally faster in drill-across and window analytics scenarios. Nevertheless, this demonstration case also shown that there is space for relational structures (see subsections 4.3.3 and 4.3.4.2), which can be beneficial for reducing the storage footprint of a BDW created using the proposed approach, avoiding extreme and unnecessary redundancy and, in certain contexts, consequently increasing processing efficiency (see subsection 7.2.3). This last insight enabled the refinement of the approach through the creation of spatial, date, and time objects, and the creation of outsourced descriptive families.

Generally, the results achieved by flat analytical objects accomplished the optimal threshold in small to medium SFs (queries executed within a few seconds) and the satisfactory threshold in large SFs (queries executed within a few tens of seconds). Considering the evaluated SFs and the available infrastructure, one can conclude that the proposed approach can be used to provide interactive query execution in BDWing contexts. Moreover, there are other strategies considered in the proposed approach that are able to provide even faster results than the ones seen in this SSB+ Benchmark, such as materialized objects. One can conclude that despite the limitations of the available infrastructure, the SSB+ Benchmark revealed adequate results, which proves that the proposed approach can be used to design and implement BDWs not only when the expected BDW size fits into memory, but also when the same exceeds it, resulting in effective and efficient BDWs capable of assuring complex ad hoc querying and OLAP on commodity hardware and shared-nothing infrastructures.

Still in this demonstration case, other workloads were also evaluated in terms of query latency, including nested attributes, data partitioning, and concurrent workloads, which provided several guidelines that practitioners can take into consideration. Furthermore, the SSB+ Benchmark also served the purpose of evaluating the streaming performance of a BDW, corroborating that the proposed approach to store, process, and combine batch and streaming data is feasible, since using a single query submitted through the querying and OLAP engine, one can combine batch and streaming data into a "unified picture". The streaming workload was significantly relevant to understand the limitations of technologies like HDFS/Hive (e.g., random access disadvantages and small files problem) and NoSQL databases (e.g., Cassandra's sequential access disadvantages) when storing and retrieving vast amounts of data, and the implications that they can have on query performance. These insights were used to not only corroborate previous assumptions, but also to complement them with further guidelines to practitioners, which are depicted throughout Chapter 4.

Finally, the fourth demonstration case consisted in applying the proposed approach in a smart cities context, namely the SusCity research project (C. Costa & Santos, 2017c; SusCity, 2016). The SusCity BDWing system was a prototype developed in the aforementioned project, in which one followed the proposed models and methods, proving the suitability of the approach to solve real-world problems. The architecture of the system follows the proposed logical components and data flows, the supporting technologies are compliant with the proposed technological infrastructure model, and the

SusCity data model is guided by the proposed data modeling method. The SusCity BDW was able to support an interactive Web-based data visualization platform focusing on several smart cities concerns, such as energy, buildings efficiency, and mobility, providing adequate response times, ranging from milliseconds to a few seconds over millions of records. The SusCity data visualization platform made available several geospatial and simulation capabilities in smart cities contexts (e.g., buildings retrofitting measures and energy grid stress scenarios), proving that using the proposed guidelines, BDWs are able to support new mixed and complex analytical workloads. Moreover, to complement the insights provided by the SSB+ Benchmark, the SusCity demonstration case also shown the relevance of clearly defining batch and streaming data CPE, storage and querying guidelines, as well as the relevance of complementary analytical objects (e.g., *"buildings indicators"* in subsection 8.2.1).

9.2 Contributions to the State-of-the-art

According to the undertaken work and achieved results presented above, to the best of one's knowledge, it can be concluded that this approach represents a relevant contribution to the scientific and technical community, making available a set of artifacts for BDW design and implementation that not only can foster future research, but above all, can help practitioners build these complex systems, which otherwise would typically fall into a use case and ad hoc driven process. The models and methods proposed in this work were scientifically backed up by a DSRM for IS research process using 4 demonstration cases that allowed the evaluation of the approach mainly in terms of effectiveness, complexity, latency, and, when applicable, resource considerations (CPU usage, memory constraints, and storage footprint). Consequently, one can conclude that this approach successfully fulfills the scientific gap previously identified, i.e., the lack of a prescriptive and integrated contribution for the design and implementation of BDWs, with adequately evaluated models and methods.

Despite the relative novelty of the topic, one tried to take into consideration previously existing contributions, reason why this approach is built upon some general constructs and guidelines provided by the Lambda Architecture (Marz & Warren, 2015), the NBDRA (NBD-PWG, 2015), the Big Data Processing Flow (Krishnan, 2013), the Data Highway Concept (Kimball & Ross, 2013), and even some data denormalization encouragements discussed in previous works (Jukic et al., 2017; Santos et al., 2017; Santos & Costa, 2016; Dehdouh et al., 2015; J. P. Costa et al., 2011). Scientific progress is often made by disruptive approaches, but it is also relevant to try to build something with a solid foundation, which was relatively difficult in this work, considering the lack of maturity and contributions related to BDWing. However, this work's contribution to the state-of-the-art in BDWing was only possible due to previously explored paths and the relevant contributions of several related works, including the vast amounts of scientific and technical works related to traditional DWing systems, shaping several academic and professional formations, whose absence would otherwise make unfeasible the advancements regarding DWs in Big Data environments.

Taking this into consideration, and now focusing on the communication activity of the DSRM for IS methodology, several scientific publications related to this research work have been positively reviewed and accepted by the scientific community, which allowed the dissemination of several results. Moreover, technical content related to the work proposed here was also presented in practice-oriented forums, and a future opportunity for the dissemination of the approach through a book publication is already taking place. The following publications (summarized in Table 9.1) represent the communication activity associated with this doctoral thesis:

- Journal Publications
    - Costa, C., & Santos, M. Y. (2017). Big Data: State-of-the-art concepts, techniques, technologies, modeling approaches and research challenges. IAENG International Journal of Computer Science, 44, 285–301;
    - Costa, C., & Santos, M. Y. (2017). The data scientist profile and its representativeness in the European e-Competence framework and the skills framework for the information age. International Journal of Information Management, 37(6), 726–734. https://doi.org/10.1016/j.ijinfomgt.2017.07.010;
    - Santos, M. Y., Martinho, B., & Costa, C. (2017). Modeling and implementing big data warehouses for decision support. Journal of Management Analytics, 4(2), 111–129;

Table 9.1. Scientific publications.

Type	Numbers	Detail
Scimago Q1 Journals	3 publications 1 submitted for publication	(2) Journal of Information Management (IJIM) (1) Energy and Buildings (1) Journal of Big Data
CORE Ranking A Conferences	1 publication	(1) International Conference on Advanced Information Systems Engineering (CAISE)
Scimago Q2 Journals	1 publication	(1) International Journal of Computer Science (IJCS)
CORE Ranking B Conferences	3 publications	(2) International Database Engineering & Applications Symposium (IDEAS) (1) European, Mediterranean, and Middle Eastern Conference on Information Systems (EMCIS)
Book Chapters	1 publication 1 in press	(1) Encyclopedia of Big Data Technologies (1) Emerging Perspectives in Big Data Warehousing
Books	1 in press	(1) Big Data: Concepts, Warehousing and Analytics. *FCA - Editora de Informática*
Other conferences and journals of international scientific circulation and review	7 publications	(1) International Conference on Computer Science & Software Engineering (1) International Conference on Data Mining and Big Data (DMBD) (3) World Conference on Information Systems and Technologies (WorldCIST) (1) International Conference on Intelligent Systems (1) Journal of Management Analytics

- Santos, M. Y., Oliveira e Sá, J., Andrade, C., Vale Lima, F., Costa, E., Costa, C., ... Galvão, J. (2017). A Big Data system supporting Bosch Braga Industry 4.0 strategy. International Journal of Information Management. https://doi.org/10.1016/ j.ijinfomgt.2017.07.012;
- Monteiro, C. S., Costa, C., Pina, A., Santos, M. Y., & Ferrão, P. (2018). An urban building database (UBD) supporting a smart city information system. Energy and Buildings, 158, 244–260. https://doi.org/10.1016/j.enbuild.2017.10.009;
- Costa, E., Costa, C., & Santos, M. Y. (2019). Evaluating Partitioning and Bucketing Strategies for Hive-based Big Data Warehousing Systems. Journal of Big Data. Manuscript submitted for publication.

- Conference Proceedings

    - Santos, M. Y., & Costa, C. (2016). Data Warehousing in Big Data: From Multidimensional to Tabular Data Models. In Proceedings of the Ninth International C* Conference on Computer Science & Software Engineering (pp. 51–60). ACM. https://doi.org/10.1145/2948992.2949024;
    - Santos, M. Y., & Costa, C. (2016). Data Models in NoSQL Databases for Big Data Contexts. In 2016 International Conference of Data Mining and Big Data (DMBD) (pp. 1–11). Springer-Verlag, LNCS 9714. http://doi.org/10.1007/978-3-319-40973-3_48;
    - Santos, M. Y., Oliveira e Sá, J., Costa, C., Galvão, J., Andrade, C., Martinho, B., ... Costa, E. (2017). A Big Data Analytics Architecture for Industry 4.0. In Á. Rocha, A. M. Correia, H. Adeli, L. P. Reis, & S. Costanzo (Eds.), Recent Advances in Information Systems and Technologies. WorldCIST 2017 (pp. 175–184). Springer International Publishing. https://doi.org/10.1007/978-3-319-56538-5_19;
    - Costa, C., & Santos, M. Y. (2017). A Conceptual Model for the Professional Profile of a Data Scientist. In Á. Rocha, A. M. Correia, H. Adeli, L. P. Reis, & S. Costanzo (Eds.), Recent Advances in Information Systems and Technologies. WorldCIST 2017 (pp. 453–463). Springer International Publishing. https://doi.org/10.1007/978-3-319-56538-5_46;
    - Costa, C., & Santos, M. Y. (2017). The SusCity Big Data Warehousing Approach for Smart Cities. In Proceedings of International Database Engineering & Applications Symposium. IDEAS 2017 (p. 10). https://doi.org/10.1145/3105831.3105841;
    - Costa, E., Costa, C., & Santos, M. Y. (2017). Efficient Big Data Modeling and Organization for Hadoop Hive-Based Data Warehouses. In Information Systems. EMCIS 2017 (pp. 3–16). Springer, Cham. https://doi.org/10.1007/978-3-319-65930-5_1;
    - 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. IDEAS 2017 (pp. 242–252). ACM. https://doi.org/10.1145/3105831.3105842;

- Costa, C., & Santos, M. Y. (2018). Evaluating Several Design Patterns and Trends in Big Data Warehousing Systems. In J. Krogstie & H. A. Reijers (Eds.), Advanced Information Systems Engineering. CAISE 2018 (pp. 459–473). Springer, Cham. https://doi.org/10.1007/978-3-319-91563-0_28;
- Costa, E., Costa, C., & Santos, M. Y. (2018). Partitioning and Bucketing in Hive-Based Big Data Warehouses. In Á. Rocha, A. Hojjat, L. P. Reis, & S. Costanzo (Eds.), Trends and Advances in Information Systems and Technologies. WorldCIST 2018 (pp. 764–774). Springer, Cham. https://doi.org/10.1007/978-3-319-77712-2_72;
- Correia, J., Santos, M. Y., Costa, C., & Andrade, C. (2018). Fast Online Analytical Processing for Big Data Warehousing. In International Conference on Intelligent Systems.

▪ Book Chapters

- Costa, C., Andrade, C., & Santos, M. Y. (2018). Big Data Warehouses for Smart Industries. In S. Sakr & A. Zomaya (Eds.), Encyclopedia of Big Data Technologies. Springer, Cham. Retrieved from https://link.springer.com/referenceworkentry/10.1007/978-3-319-63962-8_204-1;
- Vale Lima, F., Costa, C., & Santos, M. Y. (2019). Real-Time Big Data Warehousing. In D. Taniar (Ed.), Emerging Perspectives in Big Data Warehousing. IGI Global. In press.

▪ Books

- Santos, M. Y., & Costa, C. (2019). Big Data: Concepts, Warehousing and Analytics. FCA - Editora de Informática. In press.

9.3 Future Work

Regarding future work, there is space for further exploration and contributions, not only related to the proposed approach, but also to other BDWing contexts. Regarding the need to detail some claims and results presented in this doctoral thesis, one can start by highlighting the need to further evaluate other storage technologies suitable for BDWs, meaning that more specific implementation guidelines can be given to practitioners if one further understands the sequential and random access capabilities of other storage technologies like Hive transaction tables (especially in Hive 3), Druid, Kudu, NewSQL databases (e.g., Apache Ignite), and other NoSQL databases (e.g., Redis, HBase, and MongoDB). Although they should adequately fit in the general approach proposed in this work, since the logical components, data flows, and data modeling method can be generalized to different storage technologies, having other insights regarding the advantages and disadvantages of certain storage systems for batch and streaming data helps providing more implementation details (e.g., mutable analytical objects problems, inter-storage pipeline considerations, and streaming inefficiencies), and helps clarifying some uncertainty related to the technologies that were not thoroughly tested through benchmarking, prototyping, or production systems.

Still related to this question, another argument that may be raised is related to the efficiency of NewSQL systems in assuring highly scalable batch and streaming storage and processing of vast amounts of data for both transactional and analytical purposes, using traditional relational data modeling techniques, which again may not necessarily hold true for every scenario related to BDWing, just like there is a severe misconception regarding the NoSQL databases' suitability for fast sequential access to data required for BDWing scenarios. Executing a more extensive SSB+ Benchmark with more technologies would definitely serve to clarify these doubts, and understand the advantages and disadvantages of certain promising technologies. Moreover, other storage technologies could also reveal interesting and more interactive results in streaming workloads, being adequate alternatives to the evaluated ones (Hive and Cassandra). Certain technologies were already explored in recently published works, namely the benchmarking of Druid for analytical purposes (Correia et al., 2018), revealing very satisfactory performance with sub-second queries over large amounts of data, while other technologies like Kudu are still under evaluation. Access to more powerful infrastructures will allow the replication of the SSB+ Benchmark with more technologies, and, maybe more relevant, fragmenting the cluster in a more efficient way, e.g., deploying streaming and batch technologies in separate nodes, which avoids the resource starvation sometimes observed in the SSB+ streaming workloads performed in this work.

Furthermore, there are a few other implementation details that may need more attention in future works, namely how update and data movement operations really affect the performance and maintenance of the BDW, evaluating the scenarios where it is not feasible or preferable to model some analytical objects as immutable, as well as evaluating scenarios wherein the data has to be constantly moved between storage systems, or small files have to be merged together, for example. A relevant factor to consider is that, in contexts with more data redundancy, updating specific values may require scanning vast amounts of records in denormalized analytical objects, or recomputing entire partitions and, therefore, it may become interesting to rigorously evaluate the BDW's performance while executing update operations. The same applies for data movement operations, such as periodically moving data from the streaming storage to the batch storage, or compacting small files generated in Hive streaming scenarios. Other concerns related to the configuration and infrastructure of BDWs may also be relevant to explore in the future, such as providing further details on efficient deployments for BDWs in highly concurrent environments, with optimal implementation guidelines for contexts wherein hundreds or thousands of users and applications are concurrently submitting queries to support analytical applications.

Finally, one considers that the approach proposed in this work covers a wide range of BDW applications. However, by the end of this doctoral thesis, one raised interest in studying how the BDW can interact with other applications in the Big Data landscape (e.g., transactional workloads on NoSQL databases, and complex event processing with predictions and immediate actions). Consequently, the proactivity characteristic in BDWs can be considered a relevant trending topic for a near future, making sure that the analytical results and insights (e.g., aggregated measures and KPIs) can be taken into consideration in a real-time environment, wherein historical data, streaming events, and predictions lead to a set of immediate actions (sometimes automatic) that should take place according to a set of complex rules predefined in a certain organizational context like manufacturing. Consequently, studying the relationship between Big Data, BDWing, business rules, and complex event processing is certainly a research path to contemplate in the following years.

References

Ali, A. R. (2018). Real-time big data warehousing and analysis framework. *2018 IEEE 3rd International Conference on Big Data Analysis (ICBDA)*, 43–49. https://doi.org/10.1109/ICBDA.2018.8367649

Almeida, R., Bernardino, J., & Furtado, P. (2015). Testing SQL and NoSQL approaches for big data warehouse systems. *International Journal of Business Process Integration and Management*, *7*(4), 322–334. https://doi.org/10.1504/IJBPIM.2015.073656

Alsubaiee, S., Altowim, Y., Altwaijry, H., Behm, A., Borkar, V., Bu, Y., ... Westmann, T. (2014). AsterixDB: A Scalable, Open Source BDMS. *Proc. VLDB Endow.*, *7*(14), 1905–1916.

Apache Hadoop. (2018). Welcome to Apache Hadoop. Retrieved July 3, 2018, from Welcome to Apache Hadoop website: https://hadoop.apache.org/

Apache Hive. (2018). Hive Transactions. Retrieved July 27, 2018, from https://cwiki.apache.org/confluence/display/Hive/Hive+Transactions

Apache Ignite. (2018). Open source memory-centric distributed database, caching, and processing platform - Apache Ignite™. Retrieved August 7, 2018, from https://ignite.apache.org/

Armbrust, M., Xin, R. S., Lian, C., Huai, Y., Liu, D., Bradley, J. K., ... others. (2015). Spark sql: Relational data processing in spark. *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 1383–1394. Retrieved from http://dl.acm.org/citation.cfm?id=2742797

Arres, B., Kabachi, N., Boussaid, O., & Bentayeb, F. (2015). Intentional Data Placement Optimization for Distributed Data Warehouses. *2015 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 80–86. https://doi.org/10.1109/SMC.2015.27

Asif, N., Dobbie, G., & Weber, G. (2013). Big data management in the context of real- time data warehousing. In *Big Data Management, Technologies, and Applications* (pp. 150–176). https://doi.org/10.4018/978-1-4666-4699-5.ch007

Baboo, L. D. S., & Kumar, P. R. (2013). Next generation data warehouse design with big data for big analytics and better insights. *Global Journal of Computer Science and Technology*, *13*(7). Retrieved from http://computerresearch.org/index.php/computer/article/view/180

Bakshi, K. (2012). Considerations for big data: Architecture and approach. *2012 IEEE Aerospace Conference*, 1–7. https://doi.org/10.1109/AERO.2012.6187357

Barkhordari, M., & Niamanesh, M. (2017). Atrak: a MapReduce-based data warehouse for big data. *The Journal of Supercomputing*, 1–15. https://doi.org/10.1007/s11227-017-2037-3

Baru, C., Bhandarkar, M., Nambiar, R., Poess, M., & Rabl, T. (2013). Benchmarking Big Data Systems and the BigData Top100 List. *Big Data*, *1*(1), 60–64. https://doi.org/10.1089/big.2013.1509

Begoli, E., & Horey, J. (2012). Design Principles for Effective Knowledge Discovery from Big Data. *2012 Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture (ECSA)*, 215–218. https://doi.org/10.1109/WICSA-ECSA.212.32

Beheshti, S.-M.-R., Benatallah, B., & Motahari-Nezhad, H. R. (2015). Scalable graph-based OLAP analytics over process execution data. *Distributed and Parallel Databases*, *34*(3), 379–423. https://doi.org/10.1007/s10619-014-7171-9

Beyer, M. (2011, November 3). Mark Beyer, Father of the Logical Data Warehouse, Guest Post. Retrieved April 5, 2016, from Gartner Blog website: http://blogs.gartner.com/merv-adrian/2011/11/03/mark-beyer-father-of-the-logical-data-warehouse-guest-post/

Bissiriou, C. A. A., & Chaoui, H. (2014). Big Data Analysis and Query Optimization Improve HadoopDB Performance. *Proceedings of the 10th International Conference on Semantic Systems*, 1–4. https://doi.org/10.1145/2660517.2660529

Bondarev, A., & Zakirov, D. (2015). Data warehouse on Hadoop platform for decision support systems in education. *2015 Twelve International Conference on Electronics Computer and Computation (ICECCO)*, 1–4. https://doi.org/10.1109/ICECCO.2015.7416884

Boyd, D., & Crawford, K. (2012). Critical Questions for Big Data. *Information, Communication & Society*, *15*(5), 662–679. https://doi.org/10.1080/1369118X.2012.678878

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

Brown, B., Chui, M., & Manyika, J. (2011). *Are you ready for the era of ‘big data’* [Report]. Retrieved from http://www.t-systems.com/solutions/download-mckinsey-quarterly-/1148544_1/blobBinary/Study-McKinsey-Big-data.pdf

Brulé, M. R. (2013). Big data in E&P: Real-time adaptive analytics and data-flow architecture. *SPE Digital Energy Conference and Exhibition*, 305–311. https://doi.org/10.2118/163721-MS

Cattell, R. (2011). Scalable SQL and NoSQL data stores. *ACM SIGMOD Record*, *39*(4), 12–27. https://doi.org/10.1145/1978915.1978919

Chai, H., Wu, G., & Zhao, Y. (2013). A Document-Based Data Warehousing Approach for Large Scale Data Mining. In Q. Zu, B. Hu, & A. Elçi (Eds.), *Pervasive Computing and the Networked World* (pp. 69–81). Springer Berlin Heidelberg.

Chandarana, P., & Vijayalakshmi, M. (2014). Big Data analytics frameworks. *2014 International Conference on Circuits, Systems, Communication and Information Technology Applications (CSCITA)*, 430–434. https://doi.org/10.1109/CSCITA.2014.6839299

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., … Gruber, R. E. (2008). Bigtable: A Distributed Storage System for Structured Data. *ACM Trans. Comput. Syst.*, *26*(2), 4:1–4:26. https://doi.org/10.1145/1365815.1365816

Chang, L., Wang, Z., Ma, T., Jian, L., Ma, L., Goldshuv, A., ... others. (2014). HAWQ: a massively parallel processing SQL engine in hadoop. *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, 1223–1234. Retrieved from http://dl.acm.org/citation.cfm?id=2595636

Chao, L., Li, C., Liang, F., Lu, X., & Xu, Z. (2015). Accelerating Apache Hive with MPI for Data Warehouse Systems. *2015 IEEE 35th International Conference on Distributed Computing Systems (ICDCS)*, 664–673. https://doi.org/10.1109/ICDCS.2015.73

Chart.js. (2017). Chart.js | Open source HTML5 Charts. Retrieved March 5, 2017, from http://www.chartjs.org/

Chen, C. L. P., & 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

Chen, H., Chiang, R. H., & Storey, V. C. (2012). Business Intelligence and Analytics: From Big Data to Big Impact. *MIS Quarterly*, *36*(4), 1165–1188. https://doi.org/10.2307/41703503

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

Chennamsetty, H., Chalasani, S., & Riley, D. (2015). Predictive analytics on Electronic Health Records (EHRs) using Hadoop and Hive. *2015 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, 1–5. https://doi.org/10.1109/ICECCT.2015.7226129

Chevalier, M., El Malki, M., Kopliku, A., Teste, O., & Tournier, R. (2015). Implementing multidimensional data warehouses into NoSQL. *International Conference on Enterprise Information Systems (ICEIS 2015)*, 172–183. Retrieved from ftp://ftp.irit.fr/IRIT/SIG/2015_ICEIS_CEKTT.pdf

Chou, S., Yang, C., Jiang, F., & Chang, C. (2018). The Implementation of a Data-Accessing Platform Built from Big Data Warehouse of Electric Loads. *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, *01*, 87–92. https://doi.org/10.1109/COMPSAC.2018.10208

Chowdhury, S. (2014). *Big data and data warehouse augmentation*. Retrieved from IBM Corporation website: https://www.ibm.com/developerworks/analytics/library/ba-augment-data-warehouse1/ba-augment-data-warehouse1-pdf.pdf

Clegg, D. (2015). Evolving data warehouse and BI architectures: The big data challenge. *TDWI Business Intelligence Journal*, *20*(1), 19–24.

Correia, J., Santos, M. Y., Costa, C., & Andrade, C. (2018). Fast Online Analytical Processing for Big Data Warehousing. *International Conference on Intelligent Systems*.

Costa, C. (2017). SSB+ GitHub Repository. Retrieved from https://github.com/epilif1017a/bigdatabenchmarks

Costa, C., & Santos, M. Y. (2015). Improving cities sustainability through the use of data mining in a context of big city data. *2015 International Conference of Data Mining and Knowledge Engineering*, *1*, 320–325. Retrieved from https://repositorium.sdum.uminho.pt/handle/1822/36713

Costa, C., & Santos, M. Y. (2016a). BASIS: A big data architecture for smart cities. *2016 SAI Computing Conference (SAI)*, 1247–1256. https://doi.org/10.1109/SAI.2016.7556139

Costa, C., & Santos, M. Y. (2016b). Reinventing the Energy Bill in Smart Cities with NoSQL Technologies. In S. Ao, G.-C. Yang, & L. Gelman (Eds.), *Transactions on Engineering Technologies* (pp. 383–396). https://doi.org/10.1007/978-981-10-1088-0_29

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

Costa, C., & Santos, M. Y. (2017b). The data scientist profile and its representativeness in the European e-Competence framework and the skills framework for the information age. *International Journal of Information Management*, *37*(6), 726–734. https://doi.org/10.1016/j.ijinfomgt.2017.07.010

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

Costa, C., & Santos, M. Y. (2018). Evaluating Several Design Patterns and Trends in Big Data Warehousing Systems. In J. Krogstie & H. A. Reijers (Eds.), *Advanced Information Systems Engineering. CAISE 2018* (pp. 459–473). https://doi.org/10.1007/978-3-319-91563-0_28

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.), *Information Systems. EMCIS 2017* (pp. 3–16). https://doi.org/10.1007/978-3-319-65930-5_1

Costa, E., Costa, C., & Santos, M. Y. (2018). Partitioning and Bucketing in Hive-Based Big Data Warehouses. In Á. Rocha, A. Hojjat, L. P. Reis, & S. Costanzo (Eds.), *Trends and Advances in Information Systems and Technologies. WorldCIST 2018* (pp. 764–774). https://doi.org/10.1007/978-3-319-77712-2_72

Costa, J. P., Cecílio, J., Martins, P., & Furtado, P. (2011). ONE: A Predictable and Scalable DW Model. *Data Warehousing and Knowledge Discovery*, 1–13. https://doi.org/10.1007/978-3-642-23544-3_1

Cuzzocrea, A. (2013). Analytics over Big Data: Exploring the Convergence of DataWarehousing, OLAP and Data-Intensive Cloud Infrastructures. *2013 IEEE 37th Annual Computer Software and Applications Conference*, 481–483. https://doi.org/10.1109/COMPSAC.2013.152

Cuzzocrea, A. (2016). Warehousing and Protecting Big Data: State-Of-The-Art-Analysis, Methodologies, Future Challenges. *Proceedings of the International Conference on Internet of Things and Cloud Computing*, 14:1–14:7. https://doi.org/10.1145/2896387.2900335

Cuzzocrea, A., Bellatreche, L., & Song, I.-Y. (2013). Data Warehousing and OLAP over Big Data: Current Challenges and Future Research Directions. *Proceedings of the Sixteenth International Workshop on Data Warehousing and OLAP*, 67–70. https://doi.org/10.1145/2513190.2517828

Cuzzocrea, A., & Moussa, R. (2014). A Cloud-Based Framework for Supporting Effective and Efficient OLAP in Big Data Environments. *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 680–684. https://doi.org/10.1109/CCGrid.2014.129

Cuzzocrea, A., & Moussa, R. (2017). Multidimensional database modeling: Literature survey and research agenda in the big data era. *2017 International Symposium on Networks, Computers and Communications (ISNCC)*, 1–6. https://doi.org/10.1109/ISNCC.2017.8072024

Cuzzocrea, A., & Moussa, R. (2018). Towards Lambda-Based Near Real-Time OLAP over Big Data. *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, 438–441. https://doi.org/10.1109/COMPSAC.2018.00067

Cuzzocrea, A., Moussa, R., & Vercelli, G. (2018). An Innovative Lambda-Architecture-Based Data Warehouse Maintenance Framework for Effective and Efficient Near-Real-Time OLAP over Big Data. *Big Data – BigData 2018*, 149–165. https://doi.org/10.1007/978-3-319-94301-5_12

Cuzzocrea, A., Song, I.-Y., & Davis, K. C. (2011). Analytics over large-scale multidimensional data: the big data revolution! *Proceedings of the ACM 14th International Workshop on Data Warehousing and OLAP*, 101–104. Retrieved from http://dl.acm.org/citation.cfm?id=2064695

Das, K. K., Fratkin, E., Gorajek, A., Stathatos, K., & Gajjar, M. (2011). Massively Parallel In-database Predictions Using PMML. *Proceedings of the 2011 Workshop on Predictive Markup Language Modeling*, 22–27. https://doi.org/10.1145/2023598.2023601

Das, T. K., & Mohapatro, A. (2014). A Study on Big Data Integration with Data Warehouse. *International Journal of Computer Trends and Technology (IJCTT)–Volume*, *9*. Retrieved from http://www.ijcttjournal.org/Volume9/number-4/IJCTT-V9P137.pdf

Dataedo. (2017). *AdventureWorks – Data Dictionary*. Retrieved from https://dataedo.com/download/AdventureWorks.pdf

Davenport, T. H., Barth, P., & Bean, R. (2012). How big data is different. *MIT Sloan Management Review*, *54*(1), 43–46.

Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM*, *51*(1), 107–113. https://doi.org/10.1145/1327452.1327492

Dehdouh, K., Bentayeb, F., Boussaid, O., & Kabachi, N. (2014). Columnar NoSQL CUBE: Agregation operator for columnar NoSQL data warehouse. *2014 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, 3828–3833. https://doi.org/10.1109/SMC.2014.6974527

Dehdouh, K., Bentayeb, F., Boussaid, O., & Kabachi, N. (2015). Using the column oriented NoSQL model for implementing big data warehouses. *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, 469. Retrieved from http://search.proquest.com/openview/fb990658e2d8b7f76720b0a6707b9e89/1?pq-origsite=gscholar

Dumbill, E. (2013). Making sense of big data. *Big Data*, *1*(1), 1–2. https://doi.org/10.1089/big.2012.1503

Fan, W., & Bifet, A. (2013). Mining big data: current status, and forecast to the future. *ACM SIGKDD Explorations Newsletter*, *14*(2), 1–5. https://doi.org/10.1145/2481244.2481246

Ferrández, A., Maté, A., Peral, J., Trujillo, J., Gregorio, E. D., & Aufaure, M.-A. (2014). A framework for enriching Data Warehouse analysis with Question Answering systems. *Journal of Intelligent Information Systems*, *46*(1), 61–82. https://doi.org/10.1007/s10844-014-0351-2

Fisher, D., DeLine, R., Czerwinski, M., & Drucker, S. (2012). Interactions with big data analytics. *Interactions*, *19*(3), 50–59. https://doi.org/10.1145/2168931.2168943

Floratou, A., Minhas, U. F., & Özcan, F. (2014). SQL-on-Hadoop: Full Circle Back to Shared-nothing Database Architectures. *Proc. VLDB Endow.*, *7*(12), 1295–1306. https://doi.org/10.14778/2732977.2733002

Foo, A. (2013). Is the data warehouse dead? *IBM Data Management Magazine*, (5). Retrieved from https://www.ibmbigdatahub.com/blog/data-warehouse-dead

Gai, K., Qiu, M., & Sun, X. (2018). A survey on FinTech. *Journal of Network and Computer Applications*, *103*, 262–273. https://doi.org/10.1016/j.jnca.2017.10.011

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

Garber, L. (2012). Using In-Memory Analytics to Quickly Crunch Big Data. *Computer*, *45*(10), 16–18. https://doi.org/10.1109/MC.2012.358

GDELT. (2018). The GDELT Project. Retrieved August 6, 2018, from https://www.gdeltproject.org/

Ghemawat, S., Gobioff, H., & Leung, S.-T. (2003). The Google file system. *ACM SIGOPS Operating Systems Review*, *37*, 29–43. https://doi.org/10.1145/1165389.945450

Golab, L., & Johnson, T. (2014). Data stream warehousing. *2014 IEEE 30th International Conference on Data Engineering (ICDE)*, 1290–1293. https://doi.org/10.1109/ICDE.2014.6816763

Golfarelli, M., & Rizzi, S. (2018). From Star Schemas to Big Data: 20+ Years of Data Warehouse Research. In S. Flesca, S. Greco, E. Masciari, & D. Saccà (Eds.), *A Comprehensive Guide Through the Italian Database Research Over the Last 25 Years* (pp. 93–107). https://doi.org/10.1007/978-3-319-61893-7_6

Golov, N., & Rönnbäck, L. (2015). Big Data Normalization for Massively Parallel Processing Databases. In M. A. Jeusfeld & K. Karlapalem (Eds.), *Advances in Conceptual Modeling* (pp. 154–163). https://doi.org/10.1007/978-3-319-25747-1_16

Golov, N., & Rönnbäck, L. (2017). Big Data normalization for massively parallel processing databases. *Computer Standards & Interfaces*, *54, Part 2*, 86–93. https://doi.org/10.1016/j.csi.2017.01.009

Google. (2018). GitHub Repositories Dataset on Google BigQuery. Retrieved August 6, 2018, from https://bigquery.cloud.google.com/dataset/bigquery-public-data:github_repos

Google Maps. (2017). Google Maps JavaScript API. Retrieved March 5, 2017, from https://developers.google.com/maps/documentation/javascript/

Google Trends. (2018). Interest in Big Data over time. Retrieved August 7, 2018, from https://www.google.pt/trends/explore#q=big%20data

Goss, R. G., & Veeramuthu, K. (2013). Heading towards big data building a better data warehouse for more data, more speed, and more users. *Advanced Semiconductor Manufacturing Conference (ASMC), 2013 24th Annual SEMI*, 220–225. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6552808

Gröger, C., Schwarz, H., & Mitschang, B. (2014). The Deep Data Warehouse: Link-Based Integration and Enrichment of Warehouse Data and Unstructured Content. *IEEE 18th International Enterprise Distributed Object Computing Conference (EDOC)*, 210–217. https://doi.org/10.1109/EDOC.2014.36

Grolinger, K., Higashino, W. A., Tiwari, A., & Capretz, M. A. (2013). Data management in cloud environments: NoSQL and NewSQL data stores. *Journal of Cloud Computing: Advances, Systems and Applications*, *2*(1), 22.

Guo, S., Xiong, J., Wang, W., & Lee, R. (2012). Mastiff: A MapReduce-based System for Time-Based Big Data Analytics. *2012 IEEE International Conference on Cluster Computing (CLUSTER)*, 72–80. https://doi.org/10.1109/CLUSTER.2012.10

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA data mining software: an update. *ACM SIGKDD Explorations Newsletter*, *11*(1), 10–18.

Han, J., Pei, J., & Kamber, M. (2012). *Data Mining: Concepts and Techniques* (3rd ed.). Elsevier.

Hashem, I. A. T., Yaqoob, I., Anuar, N. B., Mokhtar, S., Gani, A., & Khan, S. U. (2015). The rise of “big data” on cloud computing: Review and open research issues. *Information Systems*, *47*, 98–115. https://doi.org/10.1016/j.is.2014.07.006

Hausenblas, M., & Nadeau, J. (2013). Apache drill: interactive ad-hoc analysis at scale. *Big Data*, *1*(2), 100–104.

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. *MIS Q.*, *28*(1), 75–105.

Hortonworks. (2016). *Solving Apache Hadoop Security: A Holistic Approach to a Secure Data Lake* [White paper]. Retrieved from Hortonworks website: http://hortonworks.com/info/solving-hadoop-security/

Houari, M. E., Rhanoui, M., & Asri, B. E. (2017). Hybrid big data warehouse for on-demand decision needs. *2017 International Conference on Electrical and Information Technologies (ICEIT)*, 1–6. https://doi.org/10.1109/EITech.2017.8255261

Hu, P. (2015). The Cooperative Study Between the Hadoop Big Data Platform and the Traditional Data Warehouse. *Open Automation and Control Systems Journal*, *7*, 1144–1152.

Hu, S., Liu, W., Rabl, T., Huang, S., Liang, Y., Xiao, Z., … Wang, J. (2014). DualTable: A Hybrid Storage Model for Update Optimization in Hive. *ArXiv:1404.6878 [Cs]*. Retrieved from http://arxiv.org/abs/1404.6878

Huai, Y., Chauhan, A., Gates, A., Hagleitner, G., Hanson, E. N., O'Malley, O., … Zhang, X. (2014). Major Technical Advancements in Apache Hive. *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, 1235–1246. https://doi.org/10.1145/2588555.2595630

Inmon, W. H., & Linstedt, D. (2014). *Data Architecture: A Primer for the Data Scientist: Big Data, Data Warehouse and Data Vault* (1st ed.). Morgan Kaufmann.

Intel IT Center. (2012). *Peer Research: Big Data Analytics* [Report]. Retrieved from Intel website: http://www.intel.com/content/dam/www/public/us/en/documents/reports/data-insights-peer-research-report.pdf

Jagadish, H. V., Gehrke, J., Labrinidis, A., Papakonstantinou, Y., Patel, J. M., Ramakrishnan, R., & Shahabi, C. (2014). Big data and its technical challenges. *Communications of the ACM*, *57*(7), 86–94. http://dx.doi.org/10.1145/2611567

Jara, A. J., Bocchi, Y., & Genoud, D. (2013). Determining human dynamics through the internet of things. *Proceedings of the 2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)-Volume 03*, 109–113. Retrieved from http://dl.acm.org/citation.cfm?id=2569254

Ji, C., Li, Y., Qiu, W., Awada, U., & Li, K. (2012). Big Data Processing in Cloud Computing Environments. *Proceedings of the 2012 12th International Symposium on Pervasive Systems, Algorithms, and Networks (i-Span 2012)*, 17–23. https://doi.org/10.1109/I-SPAN.2012.9

jQuery. (2017). jQuery. Retrieved March 5, 2017, from https://jquery.com/

Jukic, N., Jukic, B., Sharma, A., Nestorov, S., & Arnold, B. K. (2017). Expediting analytical databases with columnar approach. *Decision Support Systems*, *95*, 61–81. https://doi.org/10.1016/j.dss.2016.12.002

Jukic, N., Sharma, A., Nestorov, S., & Jukic, B. (2015). Augmenting Data Warehouses with Big Data. *Information Systems Management*, *32*(3), 200–209. https://doi.org/10.1080/10580530.2015.1044338

Kafka. (2018). Apache Kafka Homepage. Retrieved July 3, 2018, from https://kafka.apache.org/

Kambatla, K., Kollias, G., Kumar, V., & Grama, A. (2014). Trends in big data analytics. *Journal of Parallel and Distributed Computing*, *74*(7), 2561–2573. https://doi.org/10.1016/j.jpdc.2014.01.003

Kearney, M. (2012). Embracing big data from the warehouse. *IBM Data Management Magazine*, (4). Retrieved from http://www.ibmbigdatahub.com/blog/embracing-big-data-warehouse

Kimball, R., & Ross, M. (2013). *The data warehouse toolkit: The definitive guide to dimensional modeling* (3rd ed.). John Wiley & Sons.

Kobielus, J. (2012). Hadoop: Nucleus of the next-generation big data warehouse. *IBM Data Management Magazine*, (7). Retrieved from http://www.ibmbigdatahub.com/blog/hadoop-nucleus-next-generation-big-data-warehouse

Kornacker, M., Behm, A., Bittorf, V., Bobrovytsky, T., Choi, A., Erickson, J., ... Yoder, M. (2015). Impala: A modern, open-source sql engine for hadoop. *Proceedings of the 7th Biennial Conference on Innovative Data Systems Research (CIDR'15)*.

Krishnan, K. (2013). *Data Warehousing in the Age of Big Data* (1st ed.). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Laney, D. (2001). *3D Data Management: Controlling Data Volume, Velocity, and Variety* [Report]. Retrieved from META Group Inc website: http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf

LaValle, S., Lesser, E., Shockley, R., Hopkins, M. S., & Kruschwitz, N. (2011). Big data, analytics and the path from insights to value. *MIT Sloan Management Review*, *52*(2), 21–32.

Lebdaoui, I., Orhanou, G., & Elhajji, S. (2014). An Integration Adaptation for Real-Time Datawarehousing. *International Journal of Software Engineering and Its Applications*, *8*(11), 115–128.

Li, X., & Mao, Y. (2015). Real-Time data ETL framework for big real-time data analysis. *2015 IEEE International Conference on Information and Automation*, 1289–1294. https://doi.org/10.1109/ICInfA.2015.7279485

Lipcon, T., Alves, D., Burkert, D., Cryans, J., Dembo, A., Percy, M., ... McCabe, C. P. (2015). *Kudu: Storage for Fast Analytics on Fast Data*. Retrieved from Cloudera website: http://getkudu.io/kudu.pdf

Liu, Y., & Vitolo, T. M. (2013). Graph Data Warehouse: Steps to Integrating Graph Databases Into the Traditional Conceptual Structure of a Data Warehouse. *2013 IEEE International Congress on Big Data (BigData Congress)*, 433–434. https://doi.org/10.1109/BigData.Congress.2013.72

Mackey, G., Sehrish, S., & Wang, J. (2009). Improving metadata management for small files in HDFS. *2009 IEEE International Conference on Cluster Computing and Workshops*, 1–4. https://doi.org/10.1109/CLUSTR.2009.5289133

Madden, S. (2012). From databases to big data. *IEEE Internet Computing*, *16*(3), 4–6. http://dx.doi.org/10.1109/MIC.2012.50

Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., & Byers, A. H. (2011). *Big data: The next frontier for innovation, competition, and productivity* [Report]. Retrieved from McKinsey Global Institute website: http://www.citeulike.org/group/18242/article/9341321

Martins, D., Ramos, C. M. Q., Rodrigues, J. M. F., Cardoso, P. J. S., Lam, R., & Serra, F. (2015). Challenges in Building a Big Data Warehouse Applied to the Hotel Business Intelligence. *Proceedings of the 6th Int. Conf. on Applied Informatics and Computing Theory*. Retrieved from http://w3.ualg.pt/~jrodrig/papers_pdf/2015AICT2015.pdf

Marz, N., & Warren, J. (2015). *Big Data: Principles and best practices of scalable real-time data systems*. Manning Publications Co.

McAfee, A., Brynjolfsson, E., Davenport, T. H., Patil, D. J., & Barton, D. (2012). Big data. The Management Revolution. *The Management Revolution. Harvard Bus Rev*, *90*(10), 61–67.

Michael, K., & Miller, K. W. (2013). Big Data: New Opportunities and New Challenges [Guest editors' introduction]. *Computer*, *46*(6), 22–24. https://doi.org/10.1109/MC.2013.196

Microsoft. (2018). *SQL-server-samples: Official Microsoft GitHub Repository containing code samples for SQL Server*. Retrieved from https://github.com/Microsoft/sql-server-samples

Mohanty, S., Jagadeesh, M., & Srivatsa, H. (2013). *Big Data imperatives: enterprise Big Data warehouse, BI implementations and analytics* (1st ed.). Apress.

Monteiro, C. S., Costa, C., Pina, A., Santos, M. Y., & Ferrão, P. (2018). An urban building database (UBD) supporting a smart city information system. *Energy and Buildings*, *158*, 244–260. https://doi.org/10.1016/j.enbuild.2017.10.009

Murthy, R., & Goel, R. (2012). Peregrine: Low-latency Queries on Hive Warehouse Data. *XRDS*, *19*(1), 40–43. https://doi.org/10.1145/2331042.2331056

Nah, F. F.-H. (2004). A study on tolerable waiting time: how long are Web users willing to wait? *Behaviour & Information Technology*, *23*(3), 153–163. https://doi.org/10.1080/01449290410001669914

NBD-PWG. (2015). *NIST Big Data Interoperability Framework: Volume 6, Reference Architecture* (Technical Report No. NIST SP 1500-6). Retrieved from National Institute of Standards and Technology website: http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.1500-6.pdf

Nielsen, J. (1993). *Usability Engineering* (1st ed.). Amsterdam: Morgan Kaufmann.

NoSQL. (2018). NOSQL Databases. Retrieved November 1, 2018, from http://nosql-database.org/

O'Leary, D. E. (2014). Embedding AI and Crowdsourcing in the Big Data Lake. *IEEE Intelligent Systems*, *29*(5), 70–73. https://doi.org/10.1109/MIS.2014.82

O'Neil, P. E., O'Neil, E. J., & Chen, X. (2009). *The star schema benchmark (SSB)*. Retrieved from http://www.cs.umb.edu/~poneil/StarSchemaB.PDF

OpenAQ. (2018). OpenAQ Platform. Retrieved August 6, 2018, from OpenAQ website: https://openaq.org/

O'Sullivan, P., Thompson, G., & Clifford, A. (2014). Applying data models to big data architectures. *IBM Journal of Research and Development*, *58*(5/6), 18–1.

Parquet. (2018). Apache Parquet Homepage. Retrieved March 27, 2016, from Apache Parquet Homepage website: https://parquet.apache.org/

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

Presto. (2016, October). Presto | Distributed SQL Query Engine for Big Data. Retrieved October 23, 2016, from https://prestodb.io/

Provost, F., & Fawcett, T. (2013). Data Science and its Relationship to Big Data and Data-Driven Decision Making. *Big Data*, *1*(1), 51–59. https://doi.org/10.1089/big.2013.1508

Pujari, A. K. (2001). *Data mining techniques* (1st ed.). Universities press.

Qiao, L., Li, Y., Takiar, S., Liu, Z., Veeramreddy, N., Tu, M., … others. (2015). Gobblin: Unifying data ingestion for Hadoop. *Proceedings of the VLDB Endowment*, *8*(12), 1764–1769.

Qu, W., Rappold, M., & Dessloch, S. (2013). Adaptive prejoin approach for performance optimization in mapreduce-based warehouses. *CEUR Workshop Proceedings*, *1020*, 5–9. Retrieved from Scopus.

Ramos, C. M. Q., Correia, M. B., Rodrigues, J. M. F., Martins, D., & Serra, F. (2015). Big data warehouse framework for smart revenue management. *Advances in Environmental Science and Energy Planning*, 13–22.

Ramos, C. M. Q., Martins, D. J., Serra, F., Lam, R., Cardoso, P. J. S., Correia, M. B., & Rodrigues, J. M. F. (2017). Framework for a Hospitality Big Data Warehouse: The Implementation of an Efficient Hospitality Business Intelligence System. *International Journal of Information Systems in the Service Sector (IJISSS)*, *9*(2), 27–45. https://doi.org/10.4018/IJISSS.2017040102

Redis. (2018). Redis Homepage. Retrieved August 7, 2018, from https://redis.io/

Russom, P. (2011). *Big data analytics* (pp. 1–35) [Best Practices Report]. Retrieved from TDWI Research website: https://tdwi.org/~/media/0C630BCFD9064A9287148F1FA33460E4.pdf

Russom, P. (2014). *Evolving Data Warehouse Architectures in the Age of Big Data*. Retrieved from The Data Warehouse Institute website: https://tdwi.org/research/2014/04/best-practices-report-evolving-data-warehouse-architectures-in-the-age-of-big-data.aspx

Russom, P. (2016). *Data Warehouse Modernization in the Age of Big Data Analytics*. Retrieved from The Data Warehouse Institute website: https://tdwi.org/research/2016/03/best-practices-report-data-warehouse-modernization/asset.aspx?tc=assetpg

Sagiroglu, S., & Sinanc, D. (2013). Big data: A review. *2013 International Conference on Collaboration Technologies and Systems (CTS)*, 42–47. https://doi.org/10.1109/CTS.2013.6567202

Santos, M. Y., & Costa, C. (2016). Data Warehousing in Big Data: From Multidimensional to Tabular Data Models. *Proceedings of the Ninth International C* Conference on Computer Science & Software Engineering*, 51–60. https://doi.org/10.1145/2948992.2949024

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

Santos, M. Y., & Ramos, I. (2009). *Business Intelligence: Tecnologias da informação na gestão de conhecimento* (2nd ed.). FCA - Editora de Informática.

Santoso, L. W., & Yulia. (2017). Data Warehouse with Big Data Technology for Higher Education. *Procedia Computer Science*, *124*, 93–99. https://doi.org/10.1016/j.procs.2017.12.134

Schroeck, M., Shockley, R., Janet, S., Romero-Morales, D., & Tufano, P. (2012). *Analytics: The real-world use of big data. How innovative enterprises extract value from uncertain data*. Retrieved from https://www.ibm.com/smarterplanet/global/files/se__sv_se__intelligence__Analytics_-_The_real-world_use_of_big_data.pdf

Sebaa, A., Chikh, F., Nouicer, A., & Tari, A. (2018). Medical Big Data Warehouse: Architecture and System Design, a Case Study: Improving Healthcare Resources Distribution. *Journal of Medical Systems*, *42*(4), 59. https://doi.org/10.1007/s10916-018-0894-9

Shanahan, J. G., & Dai, L. (2015). Large scale distributed data science using apache spark. *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2323–2324. Retrieved from http://dl.acm.org/citation.cfm?id=2789993

Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010). The Hadoop Distributed File System. *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 1–10. https://doi.org/10.1109/MSST.2010.5496972

Soliman, M. A. (2017). Big Data Query Engines. In A. Y. Zomaya & S. Sakr (Eds.), *Handbook of Big Data Technologies* (pp. 179–217). https://doi.org/10.1007/978-3-319-49340-4_6

Song, J., Guo, C., Wang, Z., Zhang, Y., Yu, G., & Pierson, J.-M. (2015). HaoLap: A Hadoop based OLAP system for big data. *Journal of Systems and Software*, *102*, 167–181. https://doi.org/10.1016/j.jss.2014.09.024

Spark. (2017). Spark SQL and DataFrames - Spark 2.1.1 Documentation. Retrieved May 30, 2017, from https://spark.apache.org/docs/latest/sql-programming-guide.html#datasets-and-dataframes

Sun, L., Hu, M., Ren, K., & Ren, M. (2013). Present Situation and Prospect of Data Warehouse Architecture under the Background of Big Data. *Information Science and Cloud Computing*

Companion (ISCC-C), 2013 International Conference On, 529–535. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6973646

Sureshrao, G. S., & Ambulgekar, H. P. (2014). MapReduce-based warehouse systems: A survey. *2014 International Conference on Advances in Engineering and Technology Research (ICAETR)*, 1–8. https://doi.org/10.1109/ICAETR.2014.7012854

SusCity. (2016). SUSCITY – An MIT Portugal Project. Retrieved May 4, 2016, from http://suscity-project.eu/inicio/

Talend. (2017). Talend Open Studio for Big Data Product Details. Retrieved March 5, 2017, from https://www.talend.com/download_page_type/talend-open-studio/

Tardío, R., Mate, A., & Trujillo, J. (2015). An iterative methodology for big data management, analysis and visualization. *2015 IEEE International Conference on Big Data (Big Data)*, 545–550. https://doi.org/10.1109/BigData.2015.7363798

Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Zhang, N., ... Murthy, R. (2010). Hive-a petabyte scale data warehouse using hadoop. *IEEE 26th International Conference on Data Engineering (ICDE)*, 996–1005. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5447738

Thusoo, A., Shao, Z., Anthony, S., Borthakur, D., Jain, N., Sen Sarma, J., ... Liu, H. (2010). Data Warehousing and Analytics Infrastructure at Facebook. *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, 1013–1020. https://doi.org/10.1145/1807167.1807278

Tian, Y., Özcan, F., Zou, T., Goncalves, R., & Pirahesh, H. (2016). Building a Hybrid Warehouse: Efficient Joins Between Data Stored in HDFS and Enterprise Warehouse. *ACM Trans. Database Syst.*, *41*(4), 21:1–21:38. https://doi.org/10.1145/2972950

Tien, J. M. (2013). Big Data: Unleashing information. *Journal of Systems Science and Systems Engineering*, *22*(2), 127–151. https://doi.org/10.1007/s11518-013-5219-4

TPC. (2017a). TPC-DS - Homepage. Retrieved August 16, 2017, from http://www.tpc.org/tpcds/

TPC. (2017b). TPC-H - Homepage. Retrieved August 16, 2017, from http://www.tpc.org/tpch/

TPC. (2018). TPC-E - Homepage. Retrieved August 3, 2018, from http://www.tpc.org/tpce/

Tria, F. D., Lefons, E., & Tangorra, F. (2014). Design process for Big Data Warehouses. *2014 International Conference on Data Science and Advanced Analytics (DSAA)*, 512–518. https://doi.org/10.1109/DSAA.2014.7058120

Tria, F. D., Lefons, E., & Tangorra, F. (2018). A Framework for Evaluating Design Methodologies for Big Data Warehouses: Measurement of the Design Process. *International Journal of Data Warehousing and Mining*, *14*(1), 15–39. https://doi.org/10.4018/IJDWM.2018010102

Tudorica, B. G., & Bucur, C. (2011). A comparison between several NoSQL databases with comments and notes. *Roedunet International Conference (RoEduNet), 2011 10th*, 1–5. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5993686

Vale Lima, F., Costa, C., & Santos, M. Y. (2019). Real-Time Big Data Warehousing. In D. Taniar (Ed.), *Emerging Perspectives in Big Data Warehousing*. IGI Global. In press.

Vardarlier, P., & Silahtaroglu, G. (2016). Gossip Management at Universities Using Big Data Warehouse Model Integrated with a Decision Support System. *International Journal of Research in Business and Social Science (2147-4478)*, *5*(1), 1–14.

Vijayakumar, V., & Nedunchezhian, R. (2012). A study on video data mining. *International Journal of Multimedia Information Retrieval*, *1*(3), 153–172. https://doi.org/10.1007/s13735-012-0016-2

Villars, R. L., Olofson, C. W., & Eastwood, M. (2011). *Big data: What it is and why you should care* [Report]. Retrieved from IDC website: http://www.tracemyflows.com/uploads/big_data/idc_amd_big_data_whitepaper.pdf

Wang, H., Qin, X., Zhang, Y., Wang, S., & Wang, Z. (2011). LinearDB: A Relational Approach to Make Data Warehouse Scale Like MapReduce. In J. X. Yu, M. H. Kim, & R. Unland (Eds.), *Database Systems for Advanced Applications* (pp. 306–320). https://doi.org/10.1007/978-3-642-20152-3_23

Wang, H., Qin, X., Zhou, X., Li, F., Qin, Z., Zhu, Q., & Wang, S. (2015). Efficient query processing framework for big data warehouse: an almost join-free approach. *Frontiers of Computer Science*, *9*(2), 224–236.

Wang, S., Pandis, I., Wu, C., He, S., Johnson, D., Emam, I., … Guo, Y. (2014). High dimensional biological data retrieval optimization with NoSQL technology. *BMC Genomics*, *15 Suppl 8*, S3–S3. https://doi.org/10.1186/1471-2164-15-S8-S3

Ward, J. S., & Barker, A. (2013). Undefined By Data: A Survey of Big Data Definitions. *ArXiv:1309.5821 [Cs.DB]*. Retrieved from http://arxiv.org/abs/1309.5821

Weidner, M., Dees, J., & Sanders, P. (2013). Fast OLAP query execution in main memory on large data in a cluster. *2013 IEEE International Conference on Big Data*, 518–524. https://doi.org/10.1109/BigData.2013.6691616

White, T. (2015). *Hadoop: The Definitive Guide* (4th ed.). O'Reilly Media.

Wigan, M. R., & Clarke, R. (2013). Big Data's Big Unintended Consequences. *Computer*, *46*(6), 46–53. https://doi.org/10.1109/MC.2013.195

Xu, C., Chen, Y., Liu, Q., Rao, W., Min, H., & Su, G. (2015). A Unified Computation Engine for Big Data Analytics. *2015 IEEE/ACM 2nd International Symposium on Big Data Computing (BDC)*, 73–77. https://doi.org/10.1109/BDC.2015.41

Xu, W., Luo, W., & Woodward, N. (2012). Analysis and Optimization of Data Import with Hadoop. *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum*, 1058–1066. https://doi.org/10.1109/IPDPSW.2012.129

Yang, F., Tschetter, E., Léauté, X., Ray, N., Merlino, G., & Ganguli, D. (2014). *Druid: a real-time analytical data store*. 157–168. https://doi.org/10.1145/2588555.2595631

Yang, Q., & Helfert, M. (2017). Revisiting Arguments for a Three Layered Data Warehousing Architecture in the Context of the Hadoop Platform. *6th International Conference on Cloud Computing and Services Science*, 329–334. Retrieved from http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220%2f0005912703290334

Zhang, J., Hsu, W., & Lee, M. L. (2001). Image Mining: Issues, Frameworks and Techniques. *Proceedings of the Second International Conference on Multimedia Data Mining*, 13–20. Retrieved from http://dl.acm.org/citation.cfm?id=3012377.3012378

Zikopoulos, P., & Eaton, C. (2011). *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data* (1st ed.). McGraw-Hill Osborne Media.