CURARE : curating and managing big data collections on the cloud

Gavin Kemp


N°d'ordre NNT : xxx

THESE de DOCTORAT DE L'UNIVERSITE DE LYON
opérée au sein de
l'Université Claude Bernard Lyon 1

Ecole Doctorale ED 512
(InfoMaths)

Spécialité de doctorat :
Discipline : **Informatique**

Soutenue publiquement le 26/09/2018, par :
Gavin KEMP

CURARE: CURATING AND MANAGING BIG DATA COLLECTIONS ON THE CLOUD

Devant le jury composé de :

BELLATRECHE, Ladjel	
Professeur, ENSMA Poitiers	Rapporteur
EXPOSITO, Ernesto	
Professeur, Université de Pau et des Pays de l'Adour	Rapporteur
D'ORAZIO, Laurent	
Professeur, Université de Rennes	Examinateur
ABDERRAFIAA, Koukam	
Professeur, Université de Technologie de Belfort-Montbéliard	Examinateur
HASSAS, Salima	
Professeure, Université de Lyon 1	Examinatrice
GHODOUS, Parisa	
Professeure, Université de Lyon 1	Co-directrice de thèse
VARGAS-SOLAR, Genoveva	
Chargée de Recherches, CNRS	Co-directrice de thèse
FERREIRA DA SILVA, Catarina	
Maître de Conférences, Université de Lyon 1	Co-directrice de thèse

UNIVERSITE CLAUDE BERNARD - LYON 1

Président de l'Université	**M. le Professeur Frédéric FLEURY**
Président du Conseil Académique	M. le Professeur Hamda BEN HADID
Vice-président du Conseil d'Administration	M. le Professeur Didier REVEL
Vice-président du Conseil Formation et Vie Universitaire	M. le Professeur Philippe CHEVALIER
Vice-président de la Commission Recherche	M. Fabrice VALLÉE
Directeur Général des Services	M. Alain HELLEU

COMPOSANTES SANTE

Faculté de Médecine Lyon Est – Claude Bernard	Directeur : M. le Professeur J. ETIENNE
Faculté de Médecine et de Maïeutique Lyon Sud – Charles Mérieux	Directeur : Mme la Professeure C. BURILLON
Faculté d'Odontologie	Directeur : M. le Professeur D. BOURGEOIS
Institut des Sciences Pharmaceutiques et Biologiques	Directeur : Mme la Professeure C. VINCIGUERRA
Institut des Sciences et Techniques de la Réadaptation	Directeur : M. le Professeur Y. MATILLON
Département de formation et Centre de Recherche en Biologie Humaine	Directeur : Mme la Professeure A-M. SCHOTT

COMPOSANTES ET DEPARTEMENTS DE SCIENCES ET TECHNOLOGIE

Faculté des Sciences et Technologies	Directeur : M. F. DE MARCHI
Département Biologie	Directeur : M. le Professeur F. THEVENARD
Département Chimie Biochimie	Directeur : Mme C. FELIX
Département GEP	Directeur : M. Hassan HAMMOURI
Département Informatique	Directeur : M. le Professeur S. AKKOUCHE
Département Mathématiques	Directeur : M. le Professeur G. TOMANOV
Département Mécanique	Directeur : M. le Professeur H. BEN HADID
Département Physique	Directeur : M. le Professeur J-C PLENET
UFR Sciences et Techniques des Activités Physiques et Sportives	Directeur : M. Y.VANPOULLE
Observatoire des Sciences de l'Univers de Lyon	Directeur : M. B. GUIDERDONI
Polytech Lyon	Directeur : M. le Professeur E.PERRIN
Ecole Supérieure de Chimie Physique Electronique	Directeur : M. G. PIGNAULT
Institut Universitaire de Technologie de Lyon 1	Directeur : M. le Professeur C. VITON
Ecole Supérieure du Professorat et de l'Education	Directeur : M. le Professeur A. MOUGNIOTTE
Institut de Science Financière et d'Assurances	Directeur : M. N. LEBOISNE

KEY WORDS:

big data, cloud services, data curation, data exploration and cloud services oriented architecture

SUMMARY:

The emergence of new platforms for decentralized data creation, such as sensor and mobile platforms and the increasing availability of open data on the Web, is adding to the increase in the number of data sources inside organizations and brings an unprecedented Big Data to be explored. The notion of data curation has emerged to refer to the maintenance of data collections and the preparation and integration of datasets, combining them to perform analytics. Curation tasks include extracting explicit and implicit meta-data; semantic meta-data matching and enrichment to add quality to the data. Next generation data management engines should promote techniques with a new philosophy to cope with the deluge of data. They should aid the user in understanding the data collections' content and provide guidance to explore data. A scientist can stepwise explore into data collections and stop when the content and quality reach a satisfaction point. Our work adopts this philosophy and the main contribution is a data collections' curation approach and exploration environment named CURARE.

CURARE is a service-based system for curating and exploring Big Data. CURARE implements a data collection model that we propose, used for representing their content in terms of structural and statistical meta-data organised under the concept of view. A view is a data structure that provides an aggregated perspective of the content of a data collection and its

several associated releases. CURARE provides tools focused on computing and extracting views using data analytics methods and also functions for exploring (querying) meta-data. Exploiting Big Data requires a substantial number of decisions to be performed by data analysts to determine which is the best way to store, share and process data collections to get the maximum benefit and knowledge from them. Instead of manually exploring data collections, CURARE provides tools integrated in an environment for assisting data analysts determining which are the best collections that can be used for achieving an analytics objective. We implemented CURARE and explained how to deploy it on the cloud using data science services on top of which CURARE services are plugged. We have conducted experiments to measure the cost of computing views based on datasets of Grand Lyon and Twitter to provide insight about the interest of our data curation approach and environment.

MOTS CLEF :

données volumineuses, services cloud, curation de collections de données et architectures orientées services cloud.

RESUME :

L'émergence de nouvelles plateformes décentralisées pour la création de données, tel que les plateformes mobiles, les capteurs et l'augmentation de la disponibilité *d'open data* sur le Web, s'ajoute à l'augmentation du nombre de sources de données disponibles et apporte des données massives sans précédent à être explorées. La notion de curation de données qui a émergé se réfère à la maintenance des collections de données, à la préparation et à l'intégration d'ensembles de données (*data set*), les combinant avec une plateforme analytique. La tâche de curation inclut l'extraction de métadonnées implicites et explicites ; faire la correspondance et l'enrichissement des métadonnées sémantiques afin d'améliorer la qualité des données. La prochaine génération de moteurs de gestion de données devrait promouvoir des techniques avec une nouvelle philosophie pour faire face au déluge des données. Ils devraient aider les utilisateurs à comprendre le contenue des collections de données et à apporter une direction pour explorer les données. Un scientifique peut explorer les collections de données pas à pas, puis s'arrêter quand le contenu et la qualité atteignent des niveaux satisfaisants. Notre travail adopte cette philosophie et la principale contribution est une approche de curation des données et un environnement d'exploration que nous avons appelé CURARE.

CURARE est un système à base de services pour curer et explorer des données volumineuses sur les aspects variété et variabilité. CURARE implémente un modèle de collection de données, que nous proposons, visant représenter le contenu structurel des collections des données et les métadonnées statistiques. Le modèle de collection de données est organisé sous le concept de vue et celle-ci est une structure de données qui pourvoit une perspective agrégée du contenu des collections des données et de ses parutions (*releases*) associées. CURARE pourvoit des outils pour explorer (interroger) des métadonnées et pour extraire des vues en utilisant des méthodes analytiques. Exploiter les données massives requière un nombre considérable de décisions de la part de l'analyste des données pour trouver quelle est la meilleure façon pour stocker, partager et traiter les collections de données afin d'en obtenir le maximum de bénéfice et de connaissances à partir de ces données. Au lieu d'explorer manuellement les collections des données, CURARE fournit de outils intégrés à un environnement pour assister les analystes des données à trouver quelle est la meilleure collection qui peut être utilisée pour accomplir un objectif analytique donné. Nous avons implémenté CURARE et expliqué comment le déployer selon un modèle d'informatique dans les nuages (*cloud computing*) utilisant des services de science des donnés sur lesquels les services CURARE sont branchés. Nous avons conçu des expériences pour mesurer les coûts de la construction des vues à partir des ensembles des données du Grand Lyon et de Twitter, afin de pourvoir un aperçu de l'intérêt de notre approche et notre environnement de curation de données.

TABLE OF CONTENT

1 INTRODUCTION 1-12

1.1 CONTEXT AND MOTIVATION 1-12

1.1.1 Curating and Exploring Data Collections 1-13

1.1.2 Service Oriented Data Analytics 1-15

1.2 PROBLEM STATEMENT AND OBJETIVE 1-16

1.3 APPROACH AND CONTRIBUTIONS 1-18

1.3.1 Data Curation Approach 1-19

1.3.2 Data Curation Environment 1-20

1.4 ORGANISATION 1-20

2 BIG DATA CURATION AS A SERVICE - A STATE OF THE ART AND FUNDAMENTAL CONCEPTS 2-23

2.1 BIG DATA DEFINITIONS 2-23

2.2 BIG DATA LIFE CYCLE MANAGEMENT 2-28

2.2.1 Big Data life cycle management according to E. Curry Et al. 2-28

2.2.2 Big Data life cycle management according to H. V. Jagadish 2-31

2.3 BIG DATA CURATION 2-34

2.3.1 Requirements of data curation 2-35

2.3.2 Data acquisition and cleansing 2-40

2.3.3 Content creation: meta-data models 2-49

2.3.4 Data curation at its core 2-60

2.4 BIG DATA AS A SERVICE 2-61

2.4.1 BDaaS reference architectures 2-65

2.4.2 BDaaS tools 2-73

2.4.3 Synthesis of the state of the art regarding BDaaS 2-79

2.5 CONCLUSION 2-80

3 CURARE: SERVICE ORIENTED ARCHITECTURE FOR CURATING DATA COLLECTIONS 3-82

3.1 DATA CURATION PROCESS 3-82
3.1.1 Data Collections Structural Meta-Data 3-84
3.1.2 Data Collections Statistical Meta-Data 3-85
3.1.3 Exploring Data Collections Meta-Data 3-85
3.2 DATA CURATION ENVIRONMENT: GENERAL ARCHITECTURE 3-86
3.2.1 Data harvesting and cleaning services 3-87
3.2.2 Distributed data storage and access services 3-88
3.2.3 Data processing and exploration services 3-90
3.2.4 Big Data analytics and decision support services 3-91
3.3 DEPLOYING CURARE ON A TARGET ARCHITECTURE 3-92
3.3.1 CURARE Services and underlying Data Science Virtual Machine 3-92
3.3.2 CURARE Data Curation Life Cycle 3-97
3.4 CONCLUSION 3-100

4 DATA CURATION AS A SERVICE FOR DATA COLLECTIONS 4-102

4.1 MODELLING DATA COLLECTIONS: GENERAL PRINCIPLE 4-102
4.2 PRELIMINARIES: DATA TYPES 4-105
4.2.1 Atomic and Complex Data Types 4-106
4.2.2 Function Types 4-107
4.2.3 Relation Types 4-108
4.3 VIEW MODEL 4-109
4.3.1 Data collection 4-110
4.3.2 View 4-113
4.4 MANIPULATING VIEWS 4-122
4.4.1 Similarity 4-123
4.4.2 Union 4-124
4.4.3 Intersection 4-125
4.4.4 Difference 4-126
4.4.5 Product 4-127
4.5 MAINTAINING VIEWS 4-128

4.5.1 Insert *4-129*
4.5.2 Modify *4-130*
4.5.3 Delete *4-130*
4.6 DISCUSSION AND FINAL REMARKS 4-131

5 IMPLEMENTING THE VIEWS MODEL AND EXPERIMENTING CURARE 5-133

5.1 IMPLEMENTATION OF THE VIEW MODEL 5-133
5.1.1 Creating a `Data Collection` *5-137*
5.1.2 Creating a View *5-139*
5.2 MANIPULATING VIEWS 5-142
5.2.1 Inserting, Removing and Replacing an Item *5-142*
5.2.2 Comparing and Combining Views *5-143*
5.3 EXPERIMENTS AND USE CASE 5-145
5.3.1 Estimating the cost of creating `Data Collections` & Views *5-146*
5.3.2 Making decisions for storing Data Collections *5-152*
5.3.3 Making decisions using Views *5-157*
5.4 ANALYSIS OF EXPERIMENTS 5-164

6 CONCLUSIONS AND PERSPECTIVES 6-166

6.1 SUMMARY OF THE WORK AND CONTRIBUTION 6-166
6.2 FUTURE WORK AND PERSPECTIVES 6-168
6.2.1 Composing ad-hoc data curation and exploration environments *6-169*
6.2.2 Human in the Loop based Data Exploration *6-169*
6.2.3 Data Collections and big data service Market *6-170*

APPENDIX 6-183

1 INTRODUCTION

1.1 CONTEXT AND MOTIVATION

The appearance of deluge of data from new platforms for decentralized data creation such as social networks, sensor networks, Web open data [1], mobile applications, Internet of Things (IoT) environments brings about digital collections, known as Big Data, that can be used for new modes to reuse and to extract value from data for supporting analysis, decision making, modelling and prediction tasks. The substantial amount of variety of data collections increases the difficulty to maintain and exploit them. Forbes [2] estimates in 2025 the world will have 168 zettabytes of data, i.e. 10^{21} bytes. Gartner estimates that more than 25% of critical data in the world's top companies is flawed [3].

The introduction of the concept of data lake, a centralized repository containing virtually inexhaustible amounts of raw (or minimally curated) data that is readily made available anytime to anyone authorized to perform analytical activities, has added extra challenge. Data lakes tend to grow ever bigger and more complex to the point that some have coined the term data swamp [4]. Data collections in data lakes have somehow similar properties as the ones proposed for Big Data by authors like Edward Curry [5] and the NIST [6]. Indeed, these are "data collections with volume, velocity, variety and/or variability that is difficult to run on single machines or traditional off-the-shelf database systems". Therefore, according to these authors [4, 5], the Big Data movement has led to "a new generation of tools, methods and technologies used to collect, process and analyze massive amounts of data". Both authors

refer to the inability of traditional data architectures to efficiently handle the new datasets and have thus brought a shift in data-intensive application to parallel architectures.

Next generation of data management engines should promote techniques with a new philosophy (architecture, data processing and sharing patterns) to cope with the deluge of observational data. These should aid the user in understanding the database's content and provide guidance to explore data. A scientist can stepwise explore into data collections and stop when the content and quality reach her satisfaction point. Our work adopts this philosophy and addresses data collections curation and exploration for supporting data science tasks.

1.1.1 CURATING AND EXPLORING DATA COLLECTIONS

Big Data allows user to make surprising insights and prediction [7]. One of the key principles of data analytics is that the quality of the analysis is dependent on the quality of the information analyzed. The notion of *data curation* has emerged to refer to the maintenance of data collections and the preparation and integration of datasets, combining them to perform analytics. *Data curation* is the art of processing data to maintain it and improve its interest, value and usefulness through its lifecycle i.e. improves the quality of the data. Therefore, it implies discovering a data collection(s) of interest, cleaning and transforming new data, semantically integrating it with other local data collections and deduplicating the resulting composites if required. Data curation provides the methodological and technological data management support to address data quality issues maximizing the usability of the data for analytics and knowledge discovery purposes.

Thus, data curation tasks include extracting explicit and implicit meta-data; semantic meta-data matching and enrichment to add quality to the data. R. Y. Wang and D. M. Strong [8]

describe data quality in a number of dimensions grouped into four major dimensions: *intrinsic*, which includes accuracy and objectivity; *relevancy*, which evaluates if the data is relevant to a particular project; *representation*, "is the data explainable" and *accessibility*, which corresponds to who and how can the data be used.

There are multiple reasons why one cannot easily exploit curated data collections, the type of data, the values used, the absence of some values, the criteria used for representing such absence and the meaning of the absence (i.e., the value is unknown, the observation could not be collected, the collection is erroneous). This is hard to know without further exploration of the data collections, but more importantly the events one is trying to detect probably will not stand out but be the consequence of complex set value scattered within the data. Data exploration [9] is about efficiently extracting knowledge from data even if we do not know exactly what we are looking for. Data exploration uses algorithms and queries to discover patterns in the data.

Exploring and understanding data collections can be long and resource intensive. A quantitative view of the content of data collections is necessary to provide data analysts with aggregated views of their content. Together with meta-data, exploration techniques are required to go through the data collection without analyzing item per item.

The objective of data querying is to obtain all the data tuples respecting a defined often in the objective of answering a related question with correct and complete results. This means knowing the content of the database and its structure. In digital data collections this cannot be guaranteed. Often users are not sure which patterns they want to find and can be exploitable to answer their questions. Data exploration approaches are emerging for helping data scientists express queries that can help them understand the data collections content.

Data curation and exploration require the use of analytics and statistical algorithms that can process data collections. Such algorithms must be applied considering the characteristics of data collections, that is their volume, velocity, variety and veracity. They can require important computing resources to be executed for curating the data collections and providing tools for exploring them. They can rely in current results regarding Big Data and Data Science platforms and also on enabling architectures like the cloud that can support the execution of costly processes providing the necessary computing and storage resources.

1.1.2 SERVICE ORIENTED DATA ANALYTICS

Cloud architectures provide unlimited resources that can support data collections management and exploitation. The essential characteristics of cloud computing lie in on-demand self-service, broad network access, resource pooling, rapid elasticity and measured services [10]. These characteristics make it possible to design and implement services to deal with data collections processing and exploration using cloud resources. During the last ten years, the problem of providing intelligent data collections management using cloud computing technologies has attracted more and more attention from both academic researchers, e.g. P. Valduriez team in France [11], H. V. Jagadish team in the United States [12] or Z. Zheng from China [13] and industrial practitioners like Google Big Query, IBM and Thales.

Given data collections different attributes and given the greedy algorithms that are sometimes applied to it for giving value and making it useful for applications, requires enabling infrastructures. Thus, running data processing and analytics tasks calls for new data management strategies able to cope with the different characteristics of data collections, namely volume, variety and velocity; and also, with the quality of the data regarding its veracity, freshness, cleanness. Moving data curation and exploration to the cloud can be

interesting because it allows process of huge amounts of data in an efficient way with the existence of unlimited and adaptable computation and storage resources.

1.2 PROBLEM STATEMENT AND OBJETIVE

Big Data analytics introduces challenges when data collections must be integrated, stored, and processed. The diversity of data collections makes it difficult to determine whether it is possible to integrate, correlate, and fusion data collections collected under different conditions and with different underlying purposes. Besides, a data collection is not a static entity, providers periodically open and share releases with different characteristics (i.e., size, scope, structure, precision, freshness). For example, the Sloan Sky Server Survey[1] releases every year the astronomical observations done in the previous year, so that people not having access to the observatory can still experiment on top of these observations. Stack Overflow[2] and Wikipedia[3] release datasets of different sizes so that people can perform analysis on top of machines with specific computing and storage capacities. Important computing, storage and memory resources must be efficiently managed and provided to exploit and analyze data collections and thereby support target applications like Smart Cities managers to benefit from bulky mobility and transportation data analysis, decision making for piloting smart cities, financial markets, adapting transport infrastructure according to traffic and environmental constraints, providing alternative strategies for transporting people in the presence of disasters or exceptional situations.

[1] http://skyserver.sdss.org/dr14/en/home.aspx
[2] https://www.kaggle.com/stackoverflow/datasets
[3] https://meta.wikimedia.org/wiki/Research:Detox/Data_Release

The problem addressed in this work is can be stated in the following statements. Given a set of releases containing datasets with variable structures and content:

(1) *Compute, discover and deduce* meta-data that can provide an aggregated view summarizing:
   - structural knowledge of the dataset (e.g., number of attributes in tabular structures, item schemata in JSON like documents);
   - conditions in which data are produced (e.g., type of sensor, reading frequency, location of the producer, provenance), quantitative knowledge (e.g., attribute values distribution);
   - semantic knowledge (e.g. functional, temporal and causal dependencies among attributes in a tabular entity).

(2) Groupe *computed, deduced and discovered* meta-data into data model entities that can organize these them and ease the exploration of data collections for making decisions about the resources required to best maintain data collections and their associated meta-data, and about the way they can be used and combined for supporting analytics implementing modelling and prediction tasks.

The objective of this thesis is twofold:

- First, propose a data curation model that can model meta-data describing the structure, content and conditions in which data collections are produced.
- Second, propose a service-oriented data curation environment for harvesting, cleaning, processing data collections for computing discovering and deducing meta-data, and storing data for supporting the design of data centric experiments though exploration operations.

1.3 APPROACH AND CONTRIBUTIONS

This thesis was done in the *Laboratoire d'InfoRmatique en Image et Systèmes d'information* (LIRIS) under the supervision of Parisa Ghodous and Catarina Ferreira da Silva and with the substantial collaboration and contribution of Genoveva Vargas-Solar from the *Laboratoire d'Informatique de Grenoble* (LIG) and the *French-Mexican Laboratory of Informatics and Automatic Control* (LAFMIA). The PhD was financed by the region Rhône-Alpes via the ARC 7 program. The PhD was carried on in the context of the project ***A**ggregating and **M**anaging **B**ig r**E**altime **D**ata (AMBED) in the Cloud: application to intelligent transport for Smart Cities*. AMBED was labelled by the "*pole de competitivité*" Lyon Urban Truck and Bus (LUTB) now Cluster of the French region Auvergne-Rhône-Alpes and the *Pôle de Compétitivité* CARA. The aim to: master the concepts, methods, tools and technologies, such as anything-as-a-service and business Big Data analytics; adopt a multi cloud-based service-oriented approach for collecting, integrating, storing, and intelligently analysing digital data collections. Given that raw data collections with V's properties were at the core of these objectives, our work addressed the problem of curating data collections using data analytics techniques to enable their exploration and exploitation in target applications in the context of Transport and Smart Cities.

First, we looked into the service-oriented architectures and tried to identify what tasks could be distributed into individual cloud services for processing data collections. For this, we were inspired by the work of H.V. Jagadish [12] and his Big Data life cycle. We defined services based on harvesting, pre-processing, storage, processing, data analytics and decision support. Harvest focuses on collecting data from outside. Pre-processing runs cleaning, initial information extraction in an isolated way. Storage is responsible for storing and manipulating data in a distributed parallelized way required for Big Data. Processing runs extra phases of

cleaning and data curation using all the data at once. Data analysis runs special algorithms designed to identify patterns in the data that could be useful for end user. Finally, decision support provides the interface for end user to visualize the data to support their decisions. According to our study on these academic and industrial results addressing data collections processing particularly data curation and exploration we designed a data curation approach based on a View model and a service-based data curation environment deployed on the cloud.

1.3.1 DATA CURATION APPROACH

We propose a data curation approach designed to support the decision making of the data analysts from service selection to storage management. This approach is based on a view model designed to provide quantitative information on the data sets used and available to the data analyst. Our model provides two families of concepts:

- The data collection family is designed to reorganize data set into what we call releases. These releases correspond to data produced by a source periodically. The idea is to maintain a structure that helps track how a data collection evolves over time.
- The data view family is designed to provide the data analyst with important statistical information on the content of the data set provided by a release. The idea is to help the data analyst explore the content of a data set and have an overview of the type of data within data set (integer or float values, strings, Boolean) its completeness (number of null and missing values per attribute), possible dependencies among values, for instance whether the value of an attribute is computed using the values of other attributes. The latitude and longitude values of two attributes can be used to determine the name of the location declared in a third attribute of the same record.

The *data collection view model* provides concepts for representing quantitative and analytic aspects of the releases of a data collection, such as statistics of the content including the distribution of the values of each attribute across the items of a release, missing values, null values. Depending on the attribute type (only atomic types are considered for statistics) the strategies for computing measures can change.

1.3.2 DATA CURATION ENVIRONMENT

We proposed data curation model and tools implemented by CURARE addressing the difficulty of semi-manual data analysts' tasks by providing quantitative information on the data they are curating. The aim of CURARE is to help data analysts' decision making that consists in organizing and maintaining data according to available computing and storage resources for supporting integration and analytics processes. The *data processing and exploration services layer* of CURARE provides tools for computing the analytic view of every release and provides operators (*compare, correlate, fusion*) for exploring releases' quantitative views. The CURARE environment goes beyond the traditional approaches of maintaining and exploring data collections and enables its application to data based-sciences.

We developed a prototype running the services from harvesting to data processing used to test our data curation model and we showed how to deploy it on a cloud using a Data Science Virtual Machine giving access to cloud tools and services. As an application field, we have chosen Intelligent Transport Systems (ITS) and our test experiments have been conducted using urban transport data from the Grand Lyon portal (http://data.grandlyon.com/).

1.4 ORGANISATION

The remainder of the document is organized as follows.

- **Chapter 2** introduces the fundamental concepts related to Big Data which serve as background domain of our research. It then investigates concepts and proposals of data curation and exploration including its associated life cycle and existing systems. Since data curation and exploration approaches combine processing and analytics algorithms and methods with the objective of extracting as much knowledge as possible, the chapter summarizes the main families of these algorithms and methods. Such processes are in general computationally costly, so the chapter studies enabling architectures and environments based on parallel programming and stacks of services for analysing and storing Big Data. Finally, the chapter exhibits the main characteristics of data curation and exploration environments together with data analytics stack that are at the origin of our data curation approach and system.
- **Chapter 3** introduces our data curation approach and describes the general service-oriented architecture of the CURARE data curation environment proposed in this work. The approach identifies meta-data that can be associated to data collections and that can give a structural and statistical view of their content. These meta-data are organised into a set of entities that provide an aggregated view of the content of a dataset. Regarding the data curation environment, it organises the data curation services into three layers that define its general architecture. The chapter describes the role of the layers of services within the data curation life cycle adopted by the data curation approach.
- **Chapter 4** defines the data curation model proposed in this thesis. The *data collection model* provides concepts for representing data collections as sets of releases of raw data, where each release consists of a set of items (e.g. records). The *data collections model* is used by the data harvesting and cleansing services for representing structural and context meta-data related to collections (*provider, objective, URL, item structure*).

- **Chapter 5** describes the implementation of the data curation model within CURARE the data curation environment proposed in this thesis. Thereby, CURARE provides abstract view of the releases related to a data collection and gives the possibility of exploring the releases without having to zoom in item per item. The chapter describes the experimental setting and the experiments performed using the data curation model, to compute the cost of creating data collections and data view as well as the power it has to support decision making.
- **Chapter 6** concludes the document by summarizing the work done and its contributions. It then discusses future work and research perspectives.

2 BIG DATA CURATION AS A SERVICE - A STATE OF THE ART AND FUNDAMENTAL CONCEPTS

According to Stiftelsen for Industriell of Teknisk Forskning (SINTEF) in 2013 [14], 90% of the world's data had been produced in the 2 previous years. IBM have reiterated that trend in 2016 [15]. In fact, IBM says we are producing 2.5 quintillion bytes of data a day. That is 2.5 exabytes. People and machines are producing data at an exponential rate. This chapter investigates the state of the art on the topics of Big Data, Big Data as a service in relation to cloud computing and data curation. In section 2.1, we define what is Big Data and its properties. In section 2.2 describes the steps of the Big Data life cycle. In section 2.3 we look into data curation, the popular techniques to process Big Data and existing data curation systems. In section 2.4 we present Big Data systems and stacks intended to provide data management and analytics functions adapted to the scalability requirements of Big Data. Section 2.5 concludes the chapter and discusses about the aspects that we develop and enhance for proposing a data curation approach.

2.1 BIG DATA DEFINITIONS

The term Big Data started to be used by different major players to designate data with various characteristics and that pushed to the limits the existing data management and processing solutions. Several definitions of Big Data are available in the literature. According to D. Laney [16], "Big Data is high volume, high velocity, and/or high variety information assets that

require new forms of processing to enable enhanced decision making, insight discovery and process optimization."

The National Institute of Standards and Technologies (NIST) has begun publishing their work on standardizing Big Data. They resume their definition of Big Data as [6] :

> *"Essentially, Big Data refers to the extensibility of data repositories and data processing across resources working in parallel, in the same way that the compute-intensive simulation community embraced massively parallel processing two decades ago. By working out methods for communication among resources, the same scaling is now available to data-intensive applications.*"

In other words, Big Data refers to the inability of traditional data architectures to efficiently handle the new datasets and has thus bought a shift in data-intensive application to parallel architectures.

R. Hillard [17] from MIKE 2.0 provides an interesting insight in his explanation of Big Data "Big Data can really be very small and not all large datasets are big! It is time to find a new definition for Big Data."

- E. Curry [5] compiled a list of Big Data definitions (A new generation of tools, methods and technologies used to collect, process and analyse massive amount of data.

Table 1) that we resumed into the two following categories:

- Data collection with volume, velocity, variety and/or variability that is difficult to run on single machines or traditional databases.

– A new generation of tools, methods and technologies used to collect, process and analyse massive amount of data.

Table 1: Big Data definitions, adapted from [5].

Big Data definitions	Sources
"Big Data is high volume, high velocity, and/or high variety information assets that require new forms of processing to enable enhanced decision making, insight discovery and process optimization."	[16]
"Essentially, Big Data refers to the extensibility of data repositories and data processing across resources working in parallel, in the same way that the compute-intensive simulation community embraced massively parallel processing two decades ago. By working out methods for communication among resources, the same scaling is now available to data-intensive applications."	[6]
"Big Data can really be very small and not all large datasets are big! It's time to find a new definition for Big Data."	[17]
"When the size of the data itself becomes part of the problem and traditional techniques for working with data run out of steam"	[18]
Big Data is "data whose size forces us to look beyond the tried and true methods that are prevalent at that time"	[19]
"Big Data is a term encompassing the use of techniques to capture, process, analyse and visualize potentially large datasets in a reasonable timeframe not accessible to standard IT technologies." By extension, the platform, tools and software used for this purpose are collectively called "Big Data technologies"	[20]
"Big Data can mean big volume, big velocity, or big variety"	[21]

According to the NIST [6], Big Data has the 4 following characteristics colloquially called the 4 V's: Volume (data collections size), Velocity (data continuous production rate), Variety (different data types and format) and Variability (constant changes of data meaning and data inconsistencies).

Volume and Velocity (i.e., continuous production of new data) have an important impact in the way data is collected, archived and continuously processed. These days, data is generated at high speed and continuously by arrays of sensors or multiple events produced by devices and social media. For example, transport data can be collected from buses GPS, cars GPS, bikes station counts, trains ticket counters. This data needs to be processed in real-time, near real-time or in batch, or as streams.

Variety refers to the fact that data can be very heterogeneous in terms of formats and models (unstructured, semi-structured and structured) and content imposes new requirements to data storage and to read and write operations that must be efficient. For instance, some data sources can be a conveniently structured Java Script Object Notation (JSON) report or relational databases, other sources can be essentially graphs of measurement or images requiring processing before being useful. This imposes to have multiple technologies storing accessing and processing the data.

Variability represents the continuous change in data meaning and data structure within a data set, for example addition of attributes; and the number of inconsistencies in the data. These need to be found by anomaly and outlier detection methods in order for any meaningful analytics to occur. For example, as people update their services they will often modify the data structure with different organisation of attributes. This imposes user to use or design data

systems capable of dynamically scaling to the needs of the data flow and capable of dealing with a change in the way attributes are organised. Database design and data applications should dynamically adapt to the data format and scale to deal with the Volume and Velocity as these change over time. Other V's and non V's have been proposed like Value, which is a measure of how much usable information is in the data and that there is an economic value behind that needs to be measured. Veracity, which is a measure of how much truth and consistency is in the data, which is the case of IBM [15], but, according to NIST [6], these are not characteristics that pushed toward parallelization present in Big Data processes.

Other propose 3V's [12] focussing on the Volume, Velocity and Variety. As well as 10 Vs models [22] and 4V's model [23]. Overall, we agree with the opinion of NIST as Volume, Velocity, Variety, and Variability impose a change in technology to process and explore the data, with Veracity and Value being very important characteristics for Big Data but not defining characteristics of Big Data but to its market and dissemination. Adding value to data, given the degree of volume and variety, can require important computing, storage and memory resources. Value can be related to Big Data quality (veracity) concerning (1) data consistency related to its associated statistical reliability; (2) data provenance and trust defined by data origin, collection and processing methods, including trusted infrastructure and facility.

The constant production of data with 4 V's characteristics has introduced the concept of data lake [24]. A Data Lake stores no treated or lightly treated data having multiple underlying data models, e.g. relational, document, key value, etc. The danger with data lakes is that they can become data silos since they are often built to target consumers, and subsequently must be combined with other data (e.g., CRM, ERP, or other data lakes) for analysis. Thereby they can run into what we call data swamps. The term data swamp denotes repositories of heterogeneous collections that are difficult to exploit because they are archived with few

description of their content. Data swamps are data lakes which become over saturated variety and variability making using this data is significantly difficult to explore and navigate since as humans we deal poorly with changing models making designing data analysis algorithms a steep curb to climb.

Organizations are exploring data lakes as consolidated repositories of massive volumes of raw, detailed data of various types and formats. Data lakes stem from industrial initiatives and they are addressed according to different perspectives. Such perspectives are addressed in [25]. Indeed, creating a physical data lake presents its own hurdles, one of which is the need to store the data. This can lead to governance challenges regarding data access and quality. Data Wrangling [25] presents and describes some of the challenges inherent in creating, filing, maintaining, and governing a data lake, and propose the concept of curated data lake. This work addresses the challenges of managing data from legal and security level. Provided that a domain data analyst might not be familiar to managing legal, practical and defensive aspect of using and exposing data, this approach provides tools for dealing with this issue.

2.2 BIG DATA LIFE CYCLE MANAGEMENT

Several Big Data life cycle management strategies are proposed in literature [12], [5], [26]. In this section, we present two of these strategies, which are more focused on the steps pre-processing and preparation of data rather than on the data analysis and data visualization. We analyse the life cycle of Big Data in the eyes of Edward Curry and H.V. Jagadish.

2.2.1 BIG DATA LIFE CYCLE MANAGEMENT ACCORDING TO E. CURRY ET AL.

E. Curry et. al [5] propose a Big Data life cycle (seen in Figure 1), which they call value chain. Value chains have been used as a decision support tool to model the chain of activities that

an organization performs in order to deliver a valuable product or service to the market. The authors separate the process into 5 distinct blocks, which are: data acquisition, data analysis, data curation, data storage and data usage. We present each of these blocks hereafter.

- *Data acquisition*: is the process of gathering, filtering, and cleaning data before it is put in a data warehouse or any other storage solution on which data analysis can be carried out. Data acquisition is one of the major Big Data challenges in terms of infrastructure requirements. The infrastructure required to support the acquisition of Big Data must deliver low, predictable latency in both capturing data and in executing queries; be able to handle very high transaction volumes, often in a distributed environment; and support flexible and dynamic data structures.
- *Data analysis*: is concerned with making the raw data acquired amenable to use in decision-making as well as domain-specific usage. Data analysis involves exploring, transforming, and modelling data with the goal of highlighting relevant data, synthesizing and extracting useful hidden information with high potential from a business point of view. Related areas include data mining, business intelligence, and machine learning.

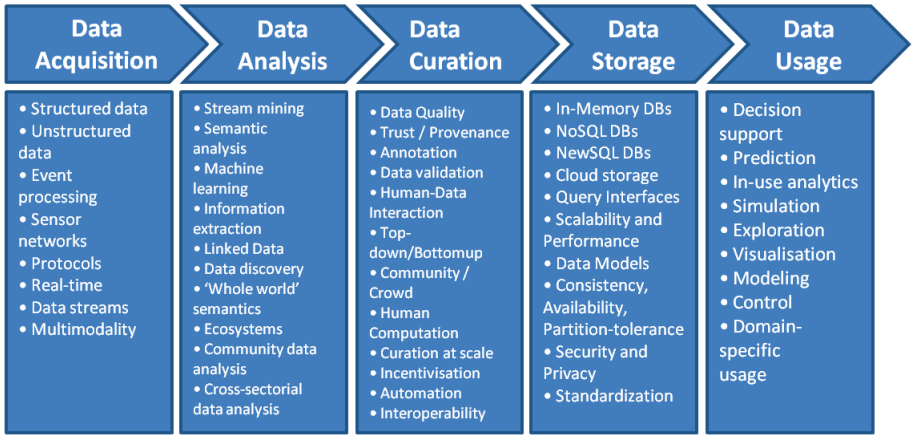

Figure 1: Big Data life cycle according to Edward Curry and colleagues [5]

- *Data curation*: is the active management of data over its life cycle to ensure it meets the necessary data quality requirements for its effective usage. Data curation processes can be categorised into different activities such as content creation, selection, classification, transformation, validation, and preservation. Data curation is responsible for ensuring that data is trustworthy, discoverable, accessible, reusable, and fit their purpose.
- *Data storage*: is the persistence and management of data in a scalable way that satisfies the needs of applications that require fast access to the data. Relational Database Management Systems (RDBMS) have been the main, and almost unique, solution to the storage paradigm for nearly 40 years. However, the ACID (Atomicity, Consistency, Isolation, and Durability) properties that guarantee database transactions lack flexibility with regard to schema changes and the performance and fault tolerance when data volumes and complexity grow, making them unsuitable for Big Data scenarios. NoSQL technologies have been designed with the scalability goal in mind and present a wide range of solutions based on alternative data models, such as MapReduce programming model [27], a technology using mapper to collect information from multiple machines then a reducer aggregating the information from those mappers, and MongoDB sharding [28], a data distribution model allowing the user to define the localization of pieces of data based on the need of user.
- *Data usage*: covers the data-driven business activities that need access to data, its analysis, and the tools needed to integrate the data analysis within the business activity. Data usage in business decision-making can enhance competitiveness through reduction of costs, increased added value, or any other parameter that can be measured against existing performance criteria.

2.2.2 BIG DATA LIFE CYCLE MANAGEMENT ACCORDING TO H. V. JAGADISH

According to H.V. Jagadish and colleagues [12], many works unfortunately focus just on the analysis/modelling step. Whilst this step is crucial, it is of little use without the other phases of the data analysis pipeline. As a consequence, they proposed the widely accepted vision of the steps of Big Data life cycle (Figure 2): (1) data acquisition, the collection and storage of data from varying sources; (2) data cleaning and information extraction, used to remove dirty/false data (generally outliers) and initial transformation of data to a more manipulatable data form; (3) data integration and aggregation, grouping of data and information from relevant data stores; (4) Big Data analysis and data interpretation, the extraction of useful information from large data groups mainly to find useful natural relations mainly through the use of statistics and decision support tools, uses the discovered information to support decision making.

Figure 2: Big Data life cycle adapted from [12]

The phases of the Big Data life cycle management according to H.V. Jagadish and colleagues are described as follows.

- *Data collection*: according to [12], data collection is the first step in Big Data life cycle. This involves basically the hardware layer and services that produce data and the service collecting and archiving the data into appropriate NoSQL data stores according to their characteristics.
- *Data cleaning and extraction*: raw data from sensors can rarely be used directly to perform analytics processes. The data is often incomplete, can contain noise or simply be wrong.

On top of that, the data structure can be impractical to use. What is more, many data analytics algorithms use statistical models to find useful information. These models are highly sensitive to extreme value and thus the data has to be sometimes trimmed for said algorithm.

- *Data Integration and aggregation*: effective large-scale analysis often requires the collection of heterogeneous data from multiple sources. For maximum value, these data sources have to combined effectively to extract the information in spite of differences in structure and production rates. A set of data transformation and integration tools helps the data analyst to resolve heterogeneities in data structure and semantics. This heterogeneity resolution leads to integrated data that is uniformly interpretable within a community, as they fit its standardization schemes and analysis needs.
- *Data analysis and interpretation*: the whole point of Big Data is to identify and extract meaningful information. Predictive tools can be developed to anticipate the future or exploratory tools used to identify useful patterns that give interesting insights on the data. Of course, the analysis is only half the work. There must then be a step where the data analyst evaluates the resulting model to identify aberrations, if present, and, in the case of exploratory analysis, interpret the new data model for useful insight. In the latter context, exploratory data analysis can only be as good as the visualization they can be used with.

When comparing the model proposed by H. V. Jagadish with the one of E. Curry and colleagues, we observe slightly different definitions for specific tasks (Table 2). With E. Curry grouping the “data analysis” and “data visualization” phases of H. V. Jagadish under “data usage”; H. V. Jagadish “data extraction” phase is included in “data analysis” in the E. Curry cycle, and H. V. Jagadish “data aggregation and integration” under Curry’s “data curation”

phase. This shows a slightly different vision between both models as Edward Curry et al. model focuses more on data preparation whereas H. V. Jagadish et al. model focuses on data usage. Table 2 summarises the correspondences between the phases of the Big Data life cycles. This brief comparison shows there is no consensus in the community regarding the phases of Big Data life cycle, their naming and definitions.

Table 2: Phase correspondences between Big Data life cycles

H. V. Jagadish and colleagues [12]	**E. Curry and colleagues [9]**
data analysis + data visualization	data usage
data extraction	data analysis
data aggregation and integration	data curation

Of course, these life cycles are designed with live Big Data applications. Developing Big Data applications requires a lot more inside knowledge of data collections than for traditional applications. This calls for a phase of data exploration [9], [29], [30] devoted to promoting the understanding of data collections content to determine what kind of analysis can be run on top of them. Data exploration uses algorithms and queries to discover patterns in the data. Each step requires knowledge of the data to run efficiently. Exploration algorithms provide the data analyst with point of view maximizing variance or correlation. If data exploration provides knowledge of the data to the user, data curation is that knowledge put into data collections. In other words, data curation can use data exploration to organise its documents. Data curation can help future users in both finding the algorithms, methods and technologies and communicate between machines and between humans and machines.

2.3 BIG DATA CURATION

Data curation[4] is the active and on-going management of data through its lifecycle of interest and usefulness to scholarly and educational activities across the sciences, social sciences, and the humanities [31]. Data curation activities enable data discovery and retrieval, maintain data quality, add value, and provide for re-use over time. This new field includes representation, archiving, authentication, management, preservation, retrieval, and use of the data.

Data curation practices may apply very broadly, for instance, the creation and enhancement of meta-data. Subject-specific practices might include migrating data from its raw state to a usable state for a given purpose; then migrating data from one standard to another or the creation of subject-specific meta-data, such as topical keywords or identification of named entities. This might require detailed knowledge of subject-specific methods of data representation or familiarity with a subject-specific schema.

Proprietary data and file formats pose significant challenges for data curation simply because they may not remain current, and the tools and software necessary to use these formats may become inaccessible (for reasons of cost or obsolescence). Any data or file format that undergoes repeated, substantive changes is also challenging because data submitted over time is likely to vary and will probably require frequent updating to maintain concurrency and consistency.

According to some authors [32] the process starts in the data acquisition phase which is the process of gathering, filtering, and cleaning data before it is put in a data warehouse or any

[4] http://guide.dhcuration.org/faq/

other storage solution on which data analysis can be carried out. Other authors start data curation once data have been stored and concern the active management of data over its life cycle to ensure it meets the necessary data quality requirements for its effective usage [33]. Data curation processes include different activities such as content creation, selection, classification, transformation, validation, and preservation.

2.3.1 REQUIREMENTS OF DATA CURATION

Data curation is performed by expert curators that are responsible for improving the accessibility and quality of data. Data curators (also known as scientific curators or data annotators) hold the responsibility of ensuring that data are trustworthy, discoverable, accessible, reusable, and fit their purpose. A key trend for the curation of Big Data utilises community and crowd sourcing approaches [32], relying on the wisdom of the crowd and community platforms to process and curate data.

2.3.1.1 THE LONG TAIL OF DATA VARIETY

Long tail refers to the part of a graph of a power distribution (see Figure 3) which approaches the limit but never reaches it. In this case, the long tail corresponds to datasets that are rarely used in the right of the graph shown in Figure 3. One of the major changes that essentially brought Big Data to the front page is what type of data we use [5]. Traditional relational data management environments were focused on data that mapped to frequent business processes and were regular enough to fit into a relational model. This data is usually characterized as structured, consistent, centralized, relatively small and highly used. Big Data on the other hand, as seen in section 2.1, is defined by the NIST as having 4 V's [6]: volume, velocity, variety and variability. This implies data that is at best semi structured, generally coming from multiple source at very high speeds. The high variability and velocity of Big Data

poses a challenge to data usage. E. Curry [34] defines this type of data a long tail data variety. Long tail data is opposed to data coming from highly structured and used source such as relational database. The long tail allows data consumers to have a more comprehensive model of their domain since it makes use of much larger set of data. The value of data curation on this type of data is substantial as shown in Figure 3. But this comes at an increased cost, since the cost of processing data increases with higher volume, more variable and less structure.

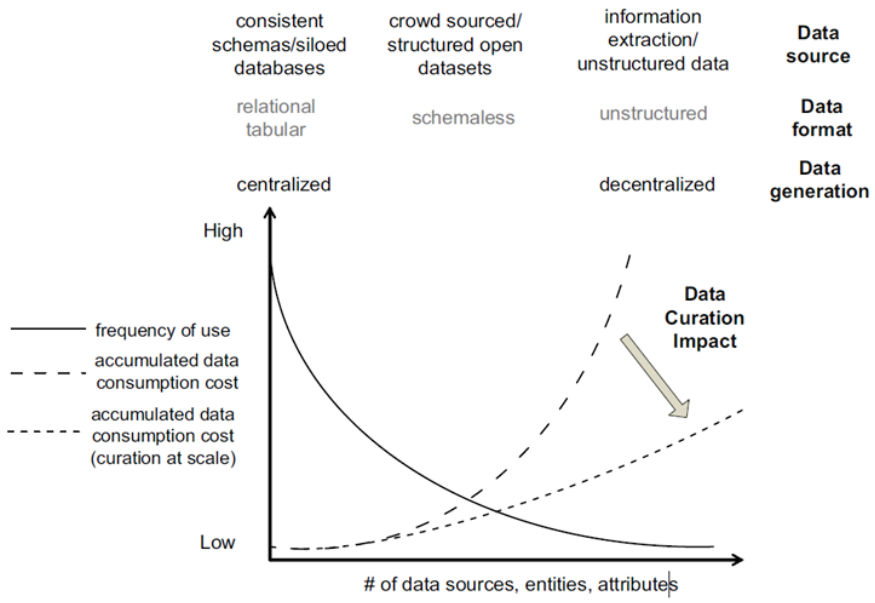

Figure 3: The long tail of data curation and the impact of data curation activities [5]

2.3.1.2 DATA QUALITY

Data quality is a metric which describes the usability of data. This is dependent on several aspects, such as available information, reliability of the source and accessibility. R. Y. Wang and D. M. Strong [8] described data quality grouped into 4 major dimensions: intrinsic, contextual, representation and accessibility.

- The intrinsic dimension corresponds to the quality of the data independently from the context of the data use. This dimension includes accuracy and objectivity but also believability, i.e. the reputation of the data and reputation of the source; since in making decision is easier when the data is given expected values and comes from reliable source.
- The contextual dimension corresponds to the quality of the data when used in a specific context, i.e. does this data answer a specific question. The criteria in this dimension are value-added, i.e. I have this data, so I have a competitive advantage; relevancy, i.e. this data has the information needed; timeliness; completeness and appropriate volume.
- The representation dimension corresponds to the quality of the data to provide understanding. This includes interpretability, i.e. the answer is easy to identify, understandability, i.e. easy to explain, concise representation and consistent representation.
- The accessibility dimension represents the cost, risks and easiness of access. Essentially do I have to pay for it, when is it accessible and is it safe. These are summarised as accessibility and access security. This dimension is different from the previous ones as there is essentially nothing we can do to improve this dimension since it in the hands of the data holders. But it is still a dimension to take into account when choosing data.

This means data quality is only limited to the source of the data on some dimensions, namely in intrinsic and accessibility but can be improved in particular in representation. This is where data curation comes in, as it can ameliorate the data representation.

When it comes to data quality, trust in the user and source is paramount [5]. As a consequence, tools helping identify the provenance and tools managing data access are important. Provenance consists in identifying who, how, where and when at each step of the data processing. Provenance can be used to explicitly capture and represent the curation

decisions that are made. However, identifying data provenance at each step can be a time-consuming process if not well documented. On the other side of the spectrum, a fine control on who uses and modifies which data is important for collaborations. Most systems manage permission at the data set level and over sees general contributors. This can be enough for general purposes, but finer grained distributed system down the data item level is needed for some projects.

Standardization is also important since it removes the need for translation, vocabulary learning and enables easier data interoperability. A large part of the data curation effort consists of integrating and repurposing data created under different contexts and in many cases involves hundreds of sources. Data model standards such as the Resource Description Framework (RDF) facilitate data integration at the data model level. Defining a vocabulary in a project is an important step as this decision can carry out over the whole life time of the project. Projects can grow to becomes whole domain standard like in the case of the Protein Data Bank (PDB) [35].

2.3.1.3 DATA CURATION SCENARIOS

The growing availability of data brings the opportunity for people to use them to inform their decision-making process, allowing data consumers to have a more complete data-supported picture of reality. While some Big Data use cases are based on large-scale datasets but with small and regular schema, other decision-making scenarios depend on the integration of complex, multi-domain, and distributed data. The extraction of value from information coming from different data sources is dependent on the feasibility of integrating and analysing these data sources. Whilst unstructured data (such as text resources) can support the

decision-making process, the lack of structure inhibits the capacity to aggregate, compare and transform consistently the data.

Pharmaceutical companies, the media industry and government agencies where early adopters of data curation. The first because of the huge variety of compounds and proteins involved in the human body, the second to ease the consumption and reuse of media and the later to allow for more transparency through open data projects.

Tools born from data curation include ChemSpider [36], a chemical compound search engine which can take molecular structures as input. It is used by chemists to identify provider and chemical reactions to produce such compounds. Data curation in ChemSpider consists of the manual annotation and correction of data. Master curators validate the information whilst normal curators can participate by posting comments and new compounds. Computer algorithms check if the data submitted validates chemical structure rules. The protein data bank contains 3D structures of biological macro molecules. A significant amount of the curation process at PDB consists of providing standardized vocabulary for describing the relationships between biological entities, varying from organ tissue to the description of the molecular structure. In order to implement a global hierarchical governance approach to the data curation workflow, PDB uses review and annotation of each submitted entry before robotic curation checks for plausibility as part of the data deposition, processing, and distribution. FoldIT [37] employs human computation to resolve complex problems like protein folding. The developers of FoldIT have used gamification to enable human computation. Through these games people can predict protein structure that might help in targeting drugs at particular disease.

These projects are particularly effective forms of crowd sourcing projects due to the massive community supporting then and the strong incentive for the community to support these projects.

2.3.2 DATA ACQUISITION AND CLEANSING

Two major phases in Big Data life cycle comprise the process involved in collecting and preparing data before storage and the process involved in processing and analysing the data as a whole after storage. In the first case, data is viewed as a stream of individual item, in the latter case data it is viewed as a whole collection. We are first going to look into the processes from collection to storage.

2.3.2.1 HARVESTING DATA

Data harvesting relies on robust protocol to queue up transfer data from source to destination. Several organizations relying internally on Big Data processing have devised confidential enterprise-specific protocols but there are a few public ones, which we present hereafter.

AMQP (Advanced Message Queuing Protocol) [38] is a protocol produced from the collaboration of 23 companies. It uses 4 layers: (1) the message layer, which describes the structure of a valid message; (2) transport layer, defines how AMQP messages are to be processed; (3) the transaction layer allows for the "coordinated outcome of otherwise independent transfers"; (4) security layer, which enables the definition of means to encrypt the content of AMQP messages. The characteristics of AMQP are ubiquity, safety, fidelity, applicability, interoperability and manageability.

- Ubiquity is about making the protocol easy to extend to new industries and needs.

- Safety is about both providing integration of encryption solutions and maintaining data transfer even when the sender and receiver aren't both online.
- Fidelity is the means to ensure that the sender can express the semantics of the message and thus allow the receiver to understand what it is receiving.
- Applicability is about allowing users to use several different transmission protocols.
- Interoperability in the context of the AMQP [38] is about making the protocol independent from the implementation.
- Manageability proposes to insure the wire protocol is fault-tolerant and lossless.

Apache Kafka [39] (Figure 4) is a messaging system which aims to unify offline and online processing by providing tools to load data into Hadoop or over a cluster of machines. Kafka allows the user to read online data responding to a specific key. Essentially writing a simple query to filter of unwanted data from a data stream. Apache Flume [40] is not a pure data acquisition system but acts as a stream and event manager. It is used both for collecting data streams and storing it in a Hadoop Distributed File System (HDFS) but also to allow request in the stream, the result being redirected to the node managing that data.

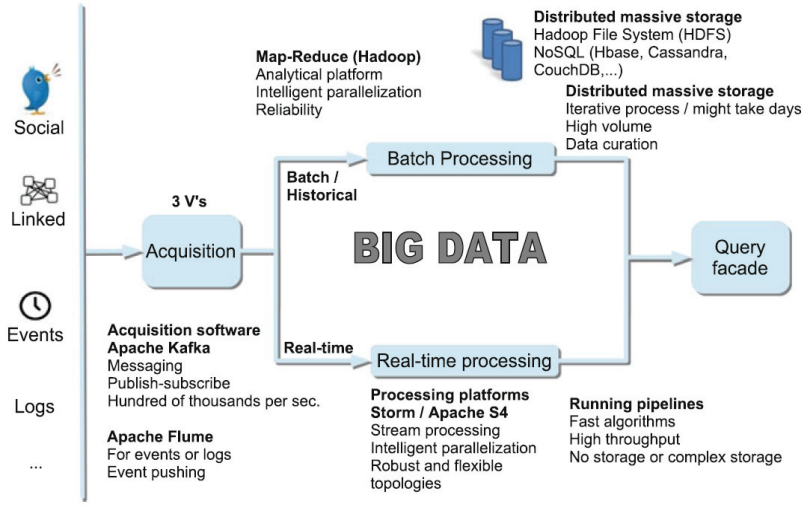

Figure 4: Tools of Big Data [5]

Beyond protocols and tools scientists have implemented several solutions to collect effectively data.

Human crowd sensing is a method of data collection using human as a sensor through the use in particular of smart phones sensors and GPS. R. Rana et al. [41] collected data to create a noise mapping using smart phones and an application called MobSLM. This application measures noise level from the environment through the smartphone microphone when the latter is not used.

Human crowd sourcing is the act of collecting data from the population through the use of pools and applications encouraging users to give information. OpenStreetMap [42] is a map generated by users through an application. It works very much in the same way than Wikipedia, where a community of ordinary users collaboratively produce maps and

geographic information. Urban insight [43] uses crowd sourcing to confirm accidents using a mobile application that pools the population in the vicinity of a supposed accident.

Passive crowd sourcing uses tools interacting with humans to passively collect data. Ticketing Public systems [44] uses the ticket validating booths to collect data on origin destination of the population automated fare. N. Lathia and L. Capra [44] collected data using London Oystercard to minimise travellers spending when going through London. This is done by analysing the user average travel from zone to zone and predicting the amount of travels based on this analysis. P. Borgnat and colleagues [45] analysed data from Lyonnaise bike renting programme called Velo'v to understand the dynamic movement of population in Lyon. J. Candia and colleagues [46] collected phone logs for mobiles phone operators to analyses human behaviour. J. Bao and colleagues [47] use the data from location based social networks like Foursquare, Loopt, and GeoLife to produce a recommendation tool that compares the user history with the history of location experts to provide location the user would be interested in visiting.

Vehicle as a sensor uses vehicles to collect data on the state of transportation by analysing their movement. J. Yuan et al. [48] used what is known as floating vehicles. These vehicles, taxi cabs this case, act as mobile GPS sensors mirroring the fluidity of the roads.

2.3.2.2 CLEANSING AND PRE-PROCESSING DATA

Exploiting data requires reasoning [5] on the data. Reasoning requires certain premises with the data such as soundness and completeness. But the reality is that data from the web, where most Big Data come from, is contradictory, incomplete, and is often very large in size. In other words, this data can be considered dirty and may require cleaning. Moreover, there is a difference between reasoning at web scale and the more tailored first-order logic, which

assumes many aspects which may differ from reality. What is more, certain analytical model, statistical one in particular, are very sensitive to extreme data, data that stand out considerably from the norm. Filtering techniques are often needed to not so much remove and resolve missing data, outlier and extreme values but at least mute them in the models using it. Noise can sometimes become an issue making the data difficult to interpret and sometimes becoming over dominant component during analysis. Techniques using smoothing or attribute reduction can be required.

Filtering techniques are mostly used in recommendation systems. It consists of identifying similarities between the data accepted and the incoming data to predict if the user wants that data [49]. Another use of filtering is to remove none pertinent data points to reduce the required processing and the readability of the data. As such Douglas–Peucker algorithm [50] is an algorithm designed to reduce the number of points on a curve. The precision of the final curve is defined by epsilon. It starts of by drawing a segment in between the first and the last point and then seeks for the point furthest from this segment, if this point is less than epsilon away from the segment it is removed, else it draws a new segment between itself and the first point and starts again then proceeds to do the same thing with the last point.

As part of data collection and data cleansing, several tools (Figure 4) implementing the MapReduce framework have been developed to easy the parallelisation processes of collection and cleaning data. Storm [51] and S4 [52] are tools that process and distribute data stream across multiple nodes. Storm identifies 3 types of nodes, Numbus which uploads the logic and distributes the code across the cluster, Zookeeper coordinates the cluster and the supervisor daemon spawn the cluster of workers. In contrast Hadoop [53] is a highly popular MapReduce framework developed in Java used for processing batches of data. These tools are

designed to process data streams of data across multiple machines and scale as the stream increase or decrease in size.

2.3.2.3 STORAGE OF DATA

As data collections get bigger, the harder they are to store. This is also for data. Whilst producing machines with large disks to store data is fairly easy these days; the cost of data storage is comparatively low. This is just one aspect of data storage; another role of data storage is manipulating processing data and with the volumes involved, this requires multiple machines to process the data. The issue is data transfer over a network is still the slowest process in information technology. This makes it challenging databases relying on references, like relational databases, since the databases will have to exchange a lot of data between machines to complete a query. A new set of database model is need and they are usually known as NoSQL databases.

V. Abramova and J. Bernardino [54] present the use of NoSQL databases and modern web technologies in particular cloud computing. In one hand, we have NoSQL database models, in particular document-oriented, key-value and wide columns stores, are famous for their great horizontal scaling mainly because their avoidance of reference system in exchange of more complex data structures and sometimes less efficient storage. Indeed, cross machine communication remains a relatively slow process making databases relying on references, namely in SQL, graph or object databases, slower if those references cross multiple nodes, work is being done to maintain queries within a node. Dynamic scalability has proven to be a particularly essential problem for databases. Top level web sites are distinguished by massive scalability, low latency, the ability to grow the capacity of the database on demand and an easy programming model. These and other features, current RDBMS just do not provide in a

cost-effective way. Relational databases (traditionally) reside on one server, which can be scaled by adding more processors, more memory and external storage. Relational database residing on multiple servers usually uses replications to keep database synchronization. One of fundamental requirements for processing applications with massive data processing is a platform for supporting of database scalability. Popular relational database like Oracle have a great expressivity, but it is difficult to scale them up by increasing the number of computers instead of a single database server. The avoidance of references used in NoSQL databases means queries are less likely to rely on data travelling between nodes to perform queries. This means when running queries, all machines involved can run at maximum speed without having to wait for data from another node travelling slowly over the network. Horizontal data distribution enables us to divide computation into concurrently processed tasks. It is obviously not easily realizable for arbitrary algorithm and arbitrary programming language. Complexity of tasks for data processing is minimized using specialized programming languages, e.g. MapReduce developed by Google, and occurring especially in context of NoSQL databases. It is worth to mention that computing in such languages does not enable effective implementation of the relational operation join.

R. Cattell [55] presents a group of horizontally scalable NoSQL databases using the BASE (Basically Available, Soft state, Eventually consistent) model thus excluding graph databases systems, object-oriented databases systems and distributed object-oriented stores that, whilst providing tools to distribute data across multiple machines, cannot scale to the same extent as key-value or document databases for example. They are though extremely effective at reference following especially if the data fit in memory.

We present 5 types of databases hereafter:

- *key-value stores*: associate a piece of data to a specific index. The most known is probably Memcached used to manage data in the RAM but other systems exist for more persistent data, like the project Voldemort[5], which supports a complex MVCC[6] (Multi Version Concurrency Control) allowing to update data within a specific version and automatic data sharding. Key-value stores should be used mainly when the application needs to use one type of object.
- *document stores storing:* is very similar to key values store in that they use a key value model. Document stores allow for more complex queries upon the attribute within the document. The most known document-oriented database is probably MongoDB[7] which on top of an effective auto-sharding system provides a very deep query and updating system, including atomic operations to attributes.
- *scalable relational databases*: store data using the relational model. MySQL cluster [8] uses a "shared nothing" architecture for scalability, as with most of the other solutions in this section, it is the most mature solution here. VoltDB[9] promotes horizontal scaling as well as a bottom-up redesign to provide very high per-node performance. Relational database Clustrix [10] supports solid state disks, but it is based on proprietary software and hardware. In theory, RDBMSs should be able to deliver scalability as long as applications avoid cross-

[5] https://www.project-voldemort.com/voldemort/
[6] Multiversion concurrency control (MCC or MVCC), is a concurrency control method commonly used by database management systems to provide concurrent access to the database and in programming languages to implement transactional memory, https://en.wikipedia.org/wiki/Multiversion_concurrency_control
[7] https://www.mongodb.com/
[8] https://www.mysql.com/
[9] https://www.voltdb.com
[10] https://www.clustrix.com/

node operations. If this proves true in practice, the built-in query language SQL with optimisation strategies and ACID transactions as execution model would give them an advantage over NoSQL for most applications.

- *extensible record stores*: sometimes called wide column stores storing extensible records; a hybrid between tuple and documents. The extensible record stores seem to have been motivated by Google's success with BigTable[11]. Their basic data model is rows and columns, and their basic scalability model is splitting both rows and columns over multiple nodes. Rows are split across nodes through sharding on the primary key. They typically split by range rather than a hash function. This means that queries on ranges of values do not have to go to every node. Columns of a table are distributed over multiple nodes by using "column groups". These may seem like a new complexity, but column groups are a simple way for the customer to indicate which columns are best stored together.
- *graph databases* [56]: replace the relational tables of relational databases with a set of items connected together with a group of relationships. This type of database is useful when the relationships between the data is more important than the data itself. These are optimised for relationship traversing rather than querying and these can be very efficient especially when the data can fit in one machine. It does have issue though with horizontal scaling for the same reason than the relational databases but can find it uses when relationships between items is important like in semantic models.

Overall each datastore brings it lot of advantages (Figure 5), key-value for their speed and simplicity, document stores for it robustness independently of the data structure allowing to organise data by topic rather than structure. Extensible records for vertical and horizontal

[11] https://cloud.google.com/bigtable/

partitioning allowing the design of more complex data stores, relational databases for using the world standard SQL language. Graph databases for its power to explore relationships between items, the last also having the more consistent ACID model, but the main challenge being can it really scale like other NoSQL systems.

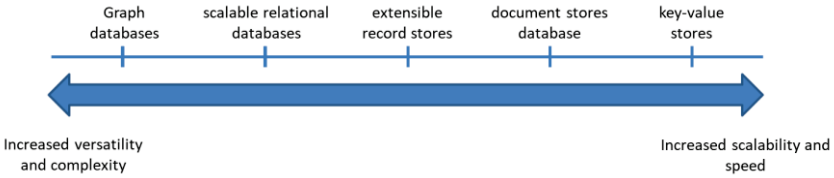

Figure 5: Comparing NoSQL databases

2.3.3 CONTENT CREATION: META-DATA MODELS

Another form of data curation involves using and extracting meta-data. As its name implies, meta-data is data about data. Meta-data is important because it provides the primary description of an attribute. For example, suppose we perform a principal component analysis on a data set and we observe that a particular attribute is heavily involved in the first component. If we have no meta-data, we cannot find out why this is the case or what are the effects that caused this. The model is still usable predictively but provides little option for understanding the data. Maintaining meta-data is a lot harder than it seems. Different data source will have different ways of describing their data and sometimes this can change within a data set as an update reclassifies or add data.

Havely et al. [57] worked on a project to rethink how to organize structured datasets at scale, in a setting where teams use diverse and often idiosyncratic ways to produce the datasets and where there is no centralized system for storing and querying the data. They classify meta-data as:

- Basic meta-data: information on files access rights, time stamp, owner, file format
- Provenance: process, tools, logs used to produce and modify the data
- Schema: seems organized by the attributes of the data
- Content summary: makes a summary of the data set search for potential keys for data cardinality
- User provided annotation: description from the author
- Semantic linking: connects terms of similar meaning together

Part of the strategy is to cluster datasets, i.e. produce a graph representing the similarity between data set. This means once cluster have been identified, analysing small number of data sets will give more of information on the other data sets. This make for a highly scalable system with large numbers of small data sets since one only has to investigate a small portion of the data. On the other hand, this will be less effective in small numbers of large data sets since getting a similarity value will have less relative value resulting in more data being analysed.

M. Stonebraker [58] proposes an environment with 3 development levels based on the amount of knowledge available about data. This approach tackles the issue of data clarity and gives a strategy to manage it. It also proposes aggregation methods. In level 3, the data source gives the information for the classification with complete knowledge in a top down strategy, where the end product can easily be imagined. In level 2, the data source gives clues and using this partial knowledge, a decision is made based on whether the knowledge available is usable by the end product or if further analysis must be done to determine uses for this data. Level 1, there is little information on the data and the end product is hard to see. In this case, a bottom up approach is favoured since the data is explored to allow climbing through levels of an end product insight.

Data Wrangling [59] present and describes some of the challenges inherent in creating, filling, maintaining, and governing a data lake, and proposes the concept of curated data lake. The authors of this work say that 70% of data analytics is identifying, cleansing, and integrating data. They present the challenges of managing data from legal and security perspective, a domain data analyst might not be familiar with, to managing data conservation. On the question of meta-data, it must be used to answer several questions: How is the data represented? Where did the data come from? (Can I trust it?) How old is the data? Can one connect this data to data she already has? This process is important because what is human readable might not be machine readable. Some data can be inconsistent or illogical. Meta-data comes under several forms: schematic meta-data extracted from the file structure, semantic meta-data extracted through the use of keywords and context awareness, idiosyncratic meta-data, confusing meta-data.

R. Hai and colleagues present Constance [60], a data lake system with sophisticated meta-data management over raw data extracted from heterogeneous data sources. Data lakes (DLs) have been conceptualized as Big Data repositories which store raw data and provide functionality for on-demand integration with the help of meta-data descriptions. Constance proposes management systems to extract explicit and implicit meta-data, as well as semantic meta-data matching and enriching to add quality to the data stored in data lakes by creating ontology model for the meta-data representing the relation between them. These forms of meta-data and the mapping involved allow the user of the data lake to query the data and get a complete answer, by using the ontology model to find related attributes to the ones in the query, as well track the usefulness of the data over time.

2.3.3.1 EXPLORING, CLUSTERING AND QUERYING DATA

Most of the Big Data information on the web and in organizations [5] is available as, at best, semi structured, but more commonly unstructured data like plain text or video. As opposed to structured data, unstructured data cannot be directly compared, aggregated and operated. One of the main novel approaches to this challenge is to perform a relationship mapping by mainly focusing on the semantics used in the data [29, 31, 33, 34]. This produces automatically a semi structured text document containing information of the data. On top of that humans are rather poor at interactions with massive data, especially if the data is unstructured. According to E. Curry [5], Carole Goble in a Data Curation Interview in 2014 said: "from a Big Data perspective, the challenges are around finding the slices, views or ways into the dataset that enables to find the bits that need to be edited, changed" [5].

Visualization and summarization [9] is key for not only to understand the data but maintain it as well. Structured query languages and the graphical interfaces developed over the top are the standard procedure for accessing data in a database. This requires knowledge of the SQL syntax for example and databases schema which is not expected knowledge from a domain expert. A solution revolves around querying a semantic model using a much more natural language using key word searches for example. Many tools exist to perform data visualization from web visualization tools such as D3.js [12] or other tools such as Matlab [13] or R [14] programming language.

[12] https://d3js.org/
[13] https://www.mathworks.com/products/matlab.html
[14] https://www.r-project.org/

Good visualization requires good algorithms to assist the user in finding the useful points of view in the data. Two major types of algorithms exist, those changing the perspective of the data and those helping identify groups of data. These algorithms rely on modelling data as data points in N dimensional space. Let us start with the perspective changing algorithms.

- Principal Component Analysis/ Singular Value Decomposition (PCA/SVD) [61] is probably the most known algorithm for exploring data sets. It is used for dimensional reduction of high dimensional data represented as a matrix. From a practical perspective, it searches for the combination of weighted attributes that express the most information. This allows data analysts to work with the more practical 2 or 3 dimensional graphs. From a geometric perspective, these techniques search for the vectors with the highest variance and then express the original matrix according to this new system of dimensions. Using the Eigen values, we can estimate the amount of information in each dimension. For instance, this can be used to identify traffic anomalies [62].
- Partial Least Square Regression (PLSR) [63] is a regression algorithm working on similar principles to PCA in that it uses dimensional reduction to condense information. The difference lies in rather than looking of the axis of maximum variance, it searches for the axis that expresses a maximum on the regression values.

Other variances of the perspective changing algorithms exist usually by changing the objective of the algorithm or the type of data. PARAFAC, for example, is regarded as the generalisation of SVD to tensors. The weighted attributes used by these algorithms have the added benefit of identifying the major contributor to changes in data. Clustering techniques can be performed on top of the perspective changing algorithms to improve these clustering techniques by removing part of the noise. Noise should neither be the major component of

the data or be a major contributor to what one is trying to measure and thus would get lost in the lower components of dimensional reducing technics.

Speaking of clustering, this is the second group of algorithms used to explore data, those helping identify groups of data, which we present hereafter.

- K-mean [45] is a classification method in which given data and the number of cluster will group the points into number of clusters. It is done by randomly selecting data elements as starting points then each other point is classified with one of the starting points. It then calculates the mean point of each cluster and uses those points as starting points for the next step. It continues for a defined number of turns.
- Hierarchical / dendrogram clustering uses classification algorithms which create a tree linking element based on their Pythagorean distance in the data space.
- Partitioning Around Medoids (PAM) K-mediod works on a similar way to k-mean but uses medians instead. Medians are more robust to extreme values.
- CLARA is a K-mediod algorithm with a sampling. It selects a small portion of the data and is then checked to be the selected in a fairly random manner and be representative of the whole data.

Another process that is used to explore data is query exploration. This provides a less quantitative approach and a more qualitative view on what the data is about.

Traditional query systems are expected to provide an exact and complete set of tuples answering the query from the user. The issue is these query systems can be slow whilst providing data or information not all that useful to the data analyst. Whilst most databases propose an indexing system to accelerate requests choosing those indexes based on the data and the use of the data, two things we do not have an answer to at the exploration phase. In

Big Data patterns are often more interesting than exact and complete answer. The reason is, as stated before, human have a poor interaction with massive data set. Patterns are more human sized and help identify relations. M. L. Kersten et al. [64] have compiled 4 methods to explore data set querying. These 4 query systems provide a broader (i.e. less precise but with a wider scope) approach, discarding exactness and completeness for speed and a more global vision of the data. The 4 query systems are presented hereafter.

- *One minute DB kernels*: as opposed to traditional databases, which focus on correctness and completeness, one-minute database kernels focus on executing a query within a strict time limit. Such a kernel differs from conventional kernels by trying to identify and avoid performance degradation points on the-fly and to answer part of the query but also without changing the query focus. One minute DB kernels sacrifice correctness and completeness for performance. This help data exploration by allowing users to perform more less accurate queries till he or she identifies the best query to answer a specific question.
- *Multi-scale queries*: as the user becomes more knowledgeable and confident in the data she is using and the direction of her queries, she will be willing to commit more resources and data to his search. This means start small and go big. Databases are a priori partitioned according to collection and databases into a large number of files, for instance 10^8 files on the case of CERN Large Hydro Collider. This means that queries can be performed on subsets of the files. The multi-scale queries method proposes, rather than one performing a query on the whole databases, to split it into multiple queries executed on different fragments of the database and then performing a union of those queries. This allow for a natural scaling of the size of the query as the user gets more confident in his query.

- *Result set post-processing and query morphing*: goes on the premise that the user probably doesn't need the exact answer to her query. Result set post-processing assumes an array of simple statistical information such as min, max, and mean to be more useful especially on massive data sets. Query morphing on the other hand assumes queries can be miss formulated. Query morphing still focuses on answering the query given by the user but will also use a small portion of its resources in searching data around the original query.
- *Queries as answers*: whilst the previous assisted user in their data exploration by either giving a broader response than the query demanded or by allowing the user to commit less resources thus allowing them to perform more queries, they don't tackle the lack of knowledge a user may have on the dataset. Queries as an answer proposes, rather than responding to a priori bad queries (too long to run, too many tuples), a new list queries based on the information within the database optimised in various directions. The key challenge of this solution is identifying bad queries, which can be done using the optimizer statistical information or massive scientific databases and identifying interesting queries to return. This could be done using a list of frequently used queries and returning them based on user feedback. Unfortunately, this can still be challenging in cases of new databases.

Exploring data is a major step in Big Data analytics for several reasons. The heterogeneous nature of Big Data and the unstructured nature of web data makes understanding the data a challenge. What is more the volumes involved interact poorly with the humans involved in using it. This is because patterns can involve hundreds of attributes amongst thousands of other attributes. Even proposing query to explore the data can be overwhelming as well as slow. This is a major issue since the information needed for the data analyst to build her application is in the data but in an inaccessible way. Data exploration transforms the large

volume of attributes into smaller chunks in the form of components, in the case of dimensional reduction and classes in the case of clustering. Query exploration technics allows to make more queries or broader queries to help the user find patterns and information in the data.

2.3.3.2 DATA MAINTENANCE

Data maintenance focuses on main challenges [65] of data recovery, data placement and data redundancy. Data recovery and data redundancy are the conflicting idea of making data recoverable. On one side data recovery looks at the process used to recover lost data usually by recovering the data from duplicate nodes. In the case of MongoDB, a replica set nodes elect a primary node which is the recognized as the holder of the truth and all queries and data manipulation is done on that node, and then replicated to other replica sets. On the other hand, replicating data has a cost which should be minimized as much as possible. As an example, MongoDB replica sets provide no added value to a database beyond its backup functionalities, but still requires a fully-fledged server to run the software in the case the primary fails. Some solutions like in couchDB[15] use replica sets to improve performance by performing what is known as a dirty query upon the servers, but this presents a challenge in data consistency and resolving conflicts between nodes receiving conflicting orders.

Maintaining data collections focus on the sets of tools created to manage the version of each element. With the growth of parallel databases, the risk of getting duplicate data items or conflicting changes increases especially with the growth of the BASE databases since data can be modified multiple times before the data has been synchronized over the whole database. This requires precise tracking of the use of data. Goods [57] for example uses a complex

[15] http://couchdb.apache.org/

version tracking index using a tree model to track the index of the version and the dates at which it has been made or manipulated. Many databases use a Multi Version Concurrency Control (MVCC). MVCC [66] uses a graph model that links the various version of the data to a common key identifying the data set.

2.3.3.3 DATA CURATION MODELS AND PLATFORMS

As said before data curation is about making data more accessible by exposing it value more efficiently. The challenging part is working out how to do to get that value. There are currently 3 main approaches for Big Data curation models in literature [5], which are Master Data Management [67], Curation at Source [32], Crowdsourcing and Collaboration spaces [68] . A brief presentation of these 3 approaches follows.

- According to H.D. Morris and D. Vesset [67], who propose Master Data Management (MDM) [67], master data is information that represents different views of the business, i.e. the different tools and business entities. MDM focuses on ensuring that an organization does not use multiple and inconsistent versions of the same master data in different parts of its systems. The three main objectives of MDM are:
    - Synchronizing master data across multiple instances of an enterprise application
    - Coordinating master data management during an application migration
    - Compliance and performance management reporting across multiple analytic systems
- Curation at Source [32] consists of performing light weight data curation as part of the normal workflow. It is used to avoid data deposit by integrating with normal workflow

tools, capture provenance information of the workflow, seamlessly interfacing with data curation infrastructure.

- Crowdsourcing and Collaboration spaces [68] is a solution to distribute data curation across multiple individuals for complex and resource intensive data curation. It relies on a collective of users to classify and structure the data such as Wikipedia. These solutions rely on the notion of "wisdom of the crowd" in which potentially large groups of non-experts can solve complex problems usually considered to be solvable only by experts. Crowdsourcing has been fuelled by the rapid development in web technologies that facilitate contributions from millions of online users. The underlying assumption is that large-scale and cheap labour can be acquired on the web.

A. Freitas and E. Curry [5] list the following key data curation platforms.

- Data Tamer[16]: is a prototype aiming to replace the current developer-centric extract-transform-load (ETL) process with automated data integration. The system uses a suit of algorithms to automatically map schemas and de-duplicate entities. However, human experts and crowds are leveraged to verify integration updates that are particularly difficult for algorithms.

- ZenCrowd[17]: this system tries to address the problem of linking named entities in text with a knowledge base. ZenCrowd bridges the gap between automated and manual linking by improving the results of automated linking with humans. The prototype was

[16] http://bigdata.csail.mit.edu/node/243
[17] http://diuf.unifr.ch/main/xi/zencrowd

demonstrated for linking named entities in news articles with entities in linked open data cloud.

- CrowdDB[18]: this database system answers SQL queries that cannot be answered by a database management system or a search engine. As opposed to the exact operation in databases, CrowdDB allows fuzzy operations with the help of humans, for example, ranking items by relevance or comparing equivalence of images.
- Qurk[19]: although similar to CrowdDB, this system tries to improve costs and latency of human-powered sorts and joins. In this regard, Qurk applies techniques such as batching, filtering, and output agreement.
- Wikipedia Bots[20]: Wikipedia runs scheduled algorithms to access quality of text articles, known as bots. These bots also flag articles that require further review by experts. SuggestBot recommends flagged articles to a Wikipedia editor based on their profile.

2.3.4 DATA CURATION AT ITS CORE

Data curation is the active and on-going management of data through its lifecycle with the objective of maintaining quality in the information it provides. It is believed that there is no size fits all solution to data curation as the quality of the data is dependent on the topic since each topic has different points of interest and therefore there are issues with the data curation models (Table 3).

MDM [67] main objective is to maintain standard data forms across a whole business, but does not offer any improvement on the existing data. Curation at Source [32] is a great

[18] http://amberonrails.com/crowddb/
[19] http://db.csail.mit.edu/qurk/
[20] https://en.wikipedia.org/wiki/Wikipedia:Bots

solution to reduce and pre-process the data before storage but lacks the whole context that curating a whole data store would provide. Crowdsourcing and Collaboration spaces [68] are an effective technique but require a large amount man power and a community interested and invested to contribute and maintain the tool provided.

The meta-data models provide an interesting tool to produce semantic models of the data. This can be very useful for users when trying to identify related attributes during exploration of the data set, which is an interesting tool for data analyst. On the other hand, such tool provides data analyst limited possibilities to answering certain questions linked to data management, in particular quantitative information.

Table 3: Data curation methods

Data curation method	Pros	Cons
MDM [67]	Standardizes the language	Doesn't improve existing data structures
Curation at Source [32]	Reduces and preprocesses data	Does it without awareness of the rest of the data
Crowdsourcing and Collaboration spaces [68]	Improves the existing data	Requires a lot of human resources

2.4 BIG DATA AS A SERVICE

Big Data as a Service (BDaaS) market is expected to grow to about USA$2.55 billion by 2021, which is 15 percent of the Big Data market, according to Dataversity [69]. The Big Data piece of "Big Data as a Service", as described in section 2.1, has 2 major definitions: big, fast and complex data sets or set of tools and techniques used to process huge volume of data. The NIST gives 4 defining V characteristics (4 V's) [6] as seen in section 2.1; volume, velocity, variability and variety; chosen as being the main contributors to having to parallelise the data processing.

The “as a Service” part is linked the service model used in information technology more specifically cloud computing in this case. The INSEE [70] defines a service as follows. “A service activity is essentially characterised by the act of making a technical or intellectual capacity/delivery available”.

Cloud computing defined by the NIST [10] as: “Cloud computing is a model for enabling convenient, on demand network access to a shared pool of configurable computing resources (e.g. network, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction” over Internet network. Cloud computing promotes the use of Service Oriented Computing and considers Everything as a Service (XaaS).

Whilst Web services provide automated IT services and are of lesser granularity than cloud services, the latter are services provided and bound to cloud providers and instantiated at the user’s demand. Cloud computing observe 3 major service types (Figure 6) representing different level of abstraction for the user:

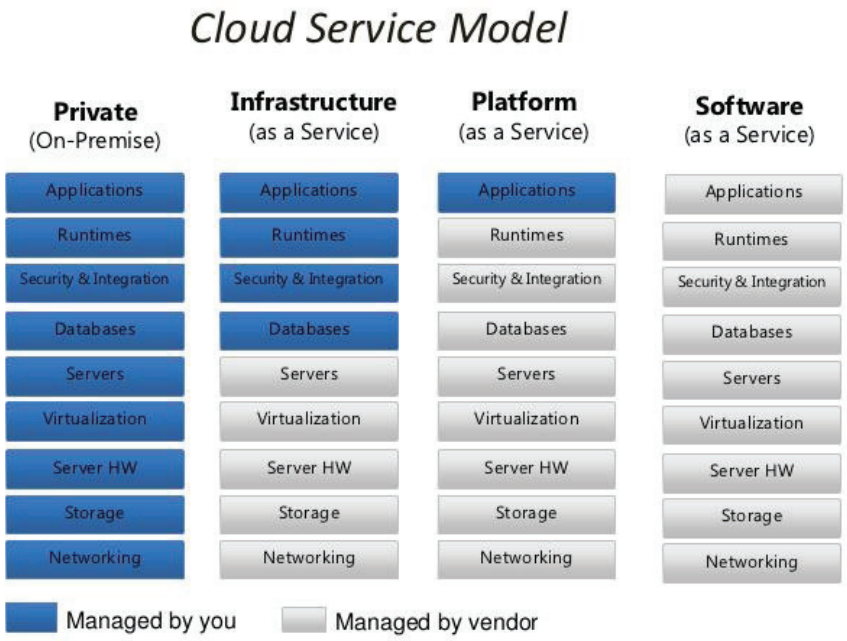

Figure 6: service models[71]

- IaaS (Infrastructure as a service) are services used to deploy and manage resources from disk space to computing cluster. Deploying services over an IaaS requires managing everything which is not machine related from databases to the application itself.
- PaaS (Platform as a Service) provide tools as a service specifically designed to manage the resources made available by the IaaS such as databases, frameworks or MapReduce managers. Thus, they are tools used by developers to implement and deploy their applications. This requires writing the logic and configuration to use these tools.
- SaaS (Software as a Service) are services used to make available to final users fully fledged applications. These can range from blog motor to specific professional tools. This requires from the user little management or configuration efforts.

Many other more specific models have been proposed but generally fit into one of the previous categories, such as Network as a Service (NaaS) which is a specific type of IaaS. Big Data is also guilty of defining these “aaS” acronyms. These are BDIaaS, BDPaaS, BDSaaS, which provide specialised definitions useful within it field of work and which imply their own constraints in design. We present these services hereafter.

- Big Data Infrastructure as a Service (BDIaaS): represents a family of services used to deploy the resources used by Big Data. This is different from traditional IaaS, the technologies for processing Big Data have to combine with storage designs, due to the massive amount of data being processed and exchanged. Both data and data analytics approaches need to be closely located to reduce the unnecessary network traffic [72].

- There are two major services which would qualify as IaaS: Storage as a service, this includes cloud storage and database as a service which should not be confused with databases management system. A DBMS is software used to manage resource and thus qualifies as a PaaS but corresponds to services used to deploy databases and their required resources. Both deploy storage resources and compute as a service focusses on cluster deployment used for parallel processing.

- Big Data Platform as a Service (BDPaaS): corresponds to the tools supported by the cloud provider and used by data analysts to manipulate data [73], the main ones being distributed Data Base Management Systems (DBMS) like MongoDB and cluster computing tools like Hadoop.
- Big Data Software as a Service (BDSaaS)[13]: corresponds to final users’ applications, with more or less user-friendly graphical interfaces, specialised in one or more of the steps of

Big Data life cycle, often using BDPaaS like DBMS and cluster computing tools. This also Includes tools providing Data as a Service.

Big Data as a Service (BDaaS) is a hot topic at the moment because combining a set of tools that need heavy horizontal scaling with tools enabling elastic, on demand horizontal scaling must be a recipe for success. Eric E. Schadt et al. [74] demonstrate the efficiency that cloud computing could have for Big Data analytics, demonstrating the analysis of 1 Petabytes of biological data in 350 minutes over 1000 nodes for the price of 2040 dollars.

2.4.1 BDAAS REFERENCE ARCHITECTURES

Architectures for managing Big Data require many tools to process the data. Here we will look into how to combine those tools and how those tools a designed.

2.4.1.1 BIG DATABASE LAYER MODEL

Different people like and need to access data at different levels. For example, a programmer would want to track words would prefer accessing the data via a Hadoop[21] accessing HDFS files, a manager looking for a specific set of information will prefer simply performing a querying on the data. Big Data databases are generally built using a layer-based approach. Each layer is used to perform a specific group of tasks to manage data (**Erreur ! Source du renvoi introuvable.**) [61, 62]. Whilst each layer is used primarily by the DBMS, most layers should also be accessible to the user as each layer allow for different type of operations. We present each layer as follows.

[21] http://hadoop.apache.org/

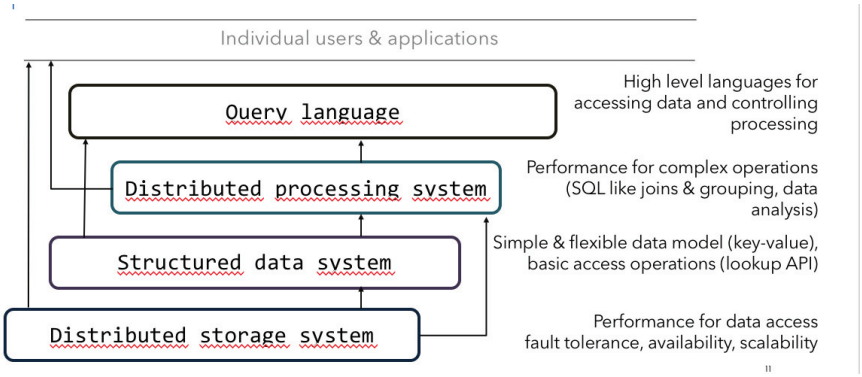

Figure 7: Big Data bases layer model [5]

- Distributed storage system: Big Data relies on distributed computing and storage. This layer is responsible in abstracting the distribution and access of data across multiple machines. This is built as a combination of node manager such as Hadoop Yarn[22], Apache Mesos[23] and Hyrack[24] and a distributed file system of which the most know is probably Hadoop Distributed Files System (HDFS).
- Structured data system: is the data model by which one's data will be represented to the user and probably in the file system. In traditional databases, we use tables and the relational model, graph or object-oriented databases. When using distributed databases, we usually opt in for a database that does not rely on links between data which can be quite heavy from a networking perspective. These are the extensible record databases, document databases and key value databases.

[22] https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html
[23] http://mesos.apache.org/
[24] http://hyracks.org/

- Distributed processing system: this layer provides the tools to process data, as required by the user, in a distributed pattern. The most well-known distributed pattern is MapReduce implemented by tools like Hadoop and Spark[25]. This should be accessible to the user as it allows for more versatile data processing than queries.
- Query language: is the language used to operate the database function. The most popular is SQL but that is limited to relational data bases. Many databases propose their own query language, such as MongoDB which uses Javascript[26] functions to access and process the data. Others have proposed to combine multiple query languages into one, often inspired from SQL, using polyglot programming[27] to achieve translation between the multiple query languages like HiveQL [77] or by directly communicating with the distributed processing system such as Pig Latin [78].

2.4.1.2 BIG DATA MANAGEMENT SYSTEMS

BDAS, the Berkeley Data Analytics Stack, is an open source software stack that integrates software components being built by the AMPLab[28] to make sense of Big Data (see **Erreur ! Source du renvoi introuvable.**). The objectives were to provide a solution where it was easy to combine batch, streaming, and interactive computations and easy to develop sophisticated algorithms in the same environment compatible with existing open source ecosystem (Hadoop/HDFS). BDAS can interoperate with existing storage and input formats (e.g., HDFS,

[25] https://spark.apache.org/
[26] https://www.javascript.com/
[27] https://deanwampler.github.io/polyglotprogramming/
[28] https://amplab.cs.berkeley.edu/software/

Hive[29], Flume[30], ..) and supports existing execution models (e.g., Hive, GraphLab). BDAS provides tools popular for data analyst like SparkR [31] that allows the user to perform MapReduce function on a Spark platform, but the user can still use a straight MapReduce using Java or directly manipulate the HDFS files if needed.

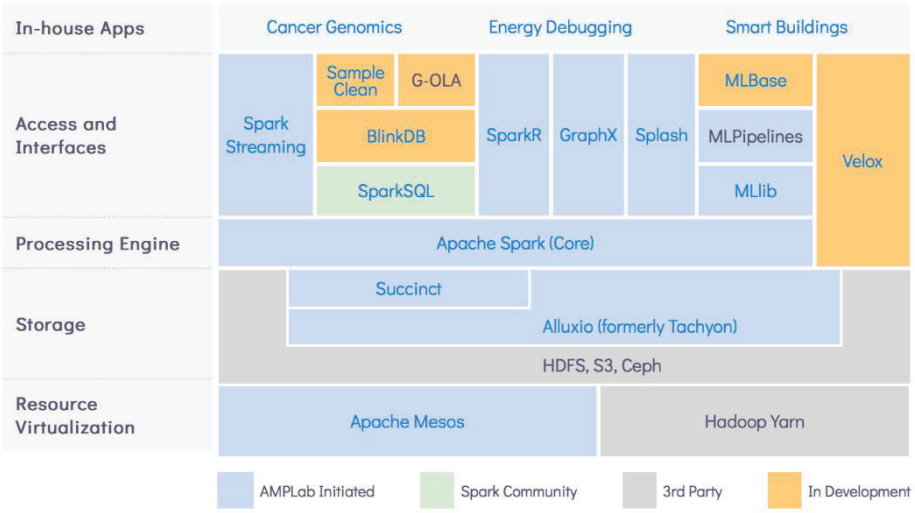

Figure 8: BDAS architecture taken from [75]

AsterixDB[32] shown Figure 9 is a scalable, open source Big Data Management System (BDMS). It supports a flexible data model, distributed storage and transaction, fast data ingestion, scalable, data-parallel query execution runtime and a declarative query language AsterixQL[33].

[29] https://hive.apache.org/
[30] https://flume.apache.org/
[31] https://spark.apache.org/docs/latest/sparkr.html
[32] https://asterixdb.apache.org/
[33] https://asterixdb.apache.org/docs/0.9.3/aql/manual.html

AsterixDB supports various storage and indexing options: managed datasets, internal LSM-based storage, external datasets (e.g., data on HDFS) and secondary indexes, for both storage options. It has access to high level query languages like AsterixQL[32] and HiveQL[34] for high level database querying but the user can still use the underlying MapReduce when needing more complex task.

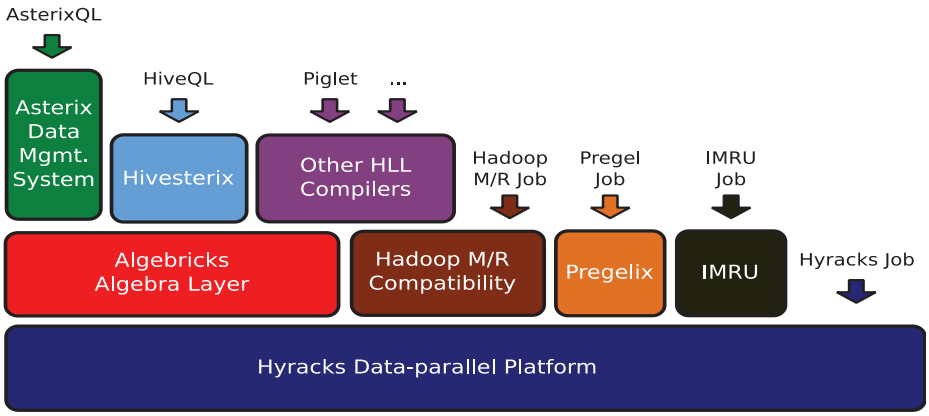

Figure 9: AsterixDB architecture [79]

Unlike analytics engines like Apache Hive or Apache Spark, it stores and manages data, so AsterixDB can exploit its knowledge of data partitioning and the availability of indexes to avoid always scanning data set(s) to process queries. “Somewhat surprisingly, there is no open source parallel database system (relational or otherwise) available to developers today – AsterixDB aims to fill this need” [79].

[34] https://cwiki.apache.org/confluence/display/Hive/LanguageManual

2.4.1.3 BIG DATA ANALYTICS SERVICE ORIENTED ARCHITECTURES

Beyond the existing storage stacks, Big Data as a Service has developed a small number Service Oriented Architectures (SOA) designed to run from data collection to decision making phases. We present some representative examples hereafter.

H. Demirkan and D. Delen [80] propose a service oriented decision support system using Big Data and the cloud based on a layered architecture separated into 3 types of services (Figure 10): the first Data as a Service (DaaS) correspond to the whole storage level of this architecture. Information coming from source like Enterprise Resource Planning (ERP) software or external data sources is transferred and stored in the enterprise data warehouse. The data is then processed using Master Data Management (MDM) to improve its quality and ensuring the data is accessible. The data can be redistributed into data marts, subsets of data that support specific decision making or analytics. The Information as a Service allows them to uniform and set a standard truth for all user of the data base. Finally, Analytics as a Service represents a platform with which to do Big Data analytics available for the company.

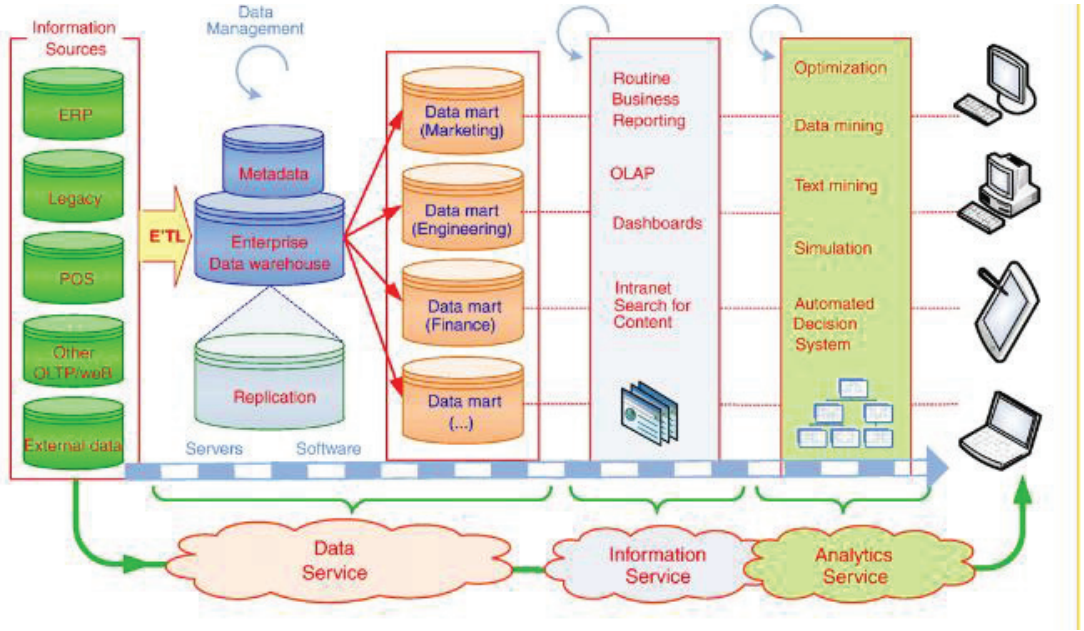

Figure 10: Decision support SOA taken from [80]

Z. Li and colleagues [81] propose a SOA for geoscience data where they separate the modelling service for geoscience, the data services, processing service and the cloud infrastructure. It presents a service model as a service (Input Parameters, Input Data)→(Output Data, Output Parameters). The services are subdivided into 4 types (Figure 11): the processing services provide the analytics and visualization to the input data. These include MapReduce but also data querying software. Data services are to fetch mainly meta-data from major geoscience data stores. Model service run geoscience model on the data. Infrastructure service that provides the tools to easily instantiate new virtual machines used for the data processing.

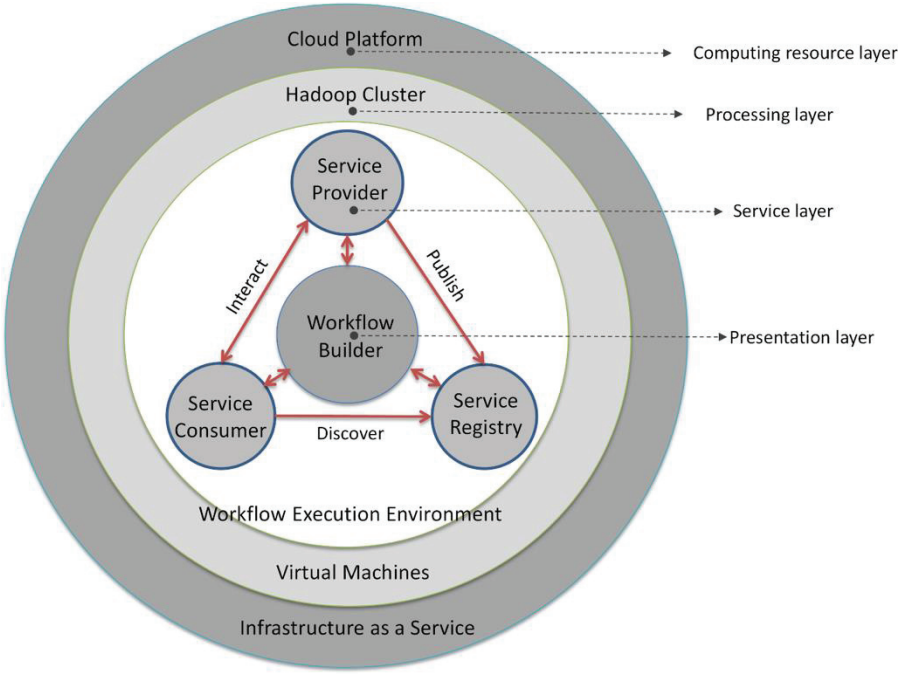

Figure 11: GeoScience SOA taken from [81]

K. Abuosba [26] decomposes the Big Data processing into multiple steps: Data Acquisition Process, Data Serialization Process, Data Aggregation Process, Data Analysis Process, Data Mining Process, Knowledge Representation Process, Information Dissemination Process. Then

maps their SOA conceptual model to Big Data to produce the SOA Big Data Processing Conceptual Model (Figure 12).

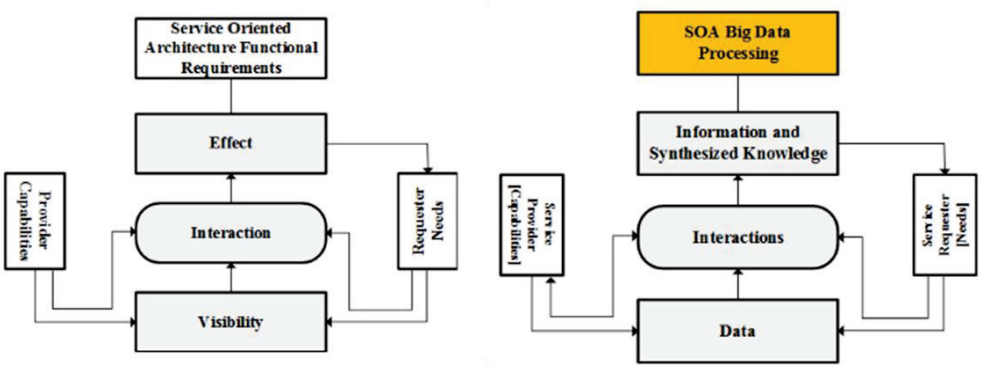

Figure 12: SOA mapping of Big Data [26]

To conclude, Service Oriented Big Data Architecture is a relatively new field due to mostly the sudden growth in cloud computing and Big Data. A lot of work has been produced demonstrating the advantages of Big Data and cloud mostly from a data storage point of view because of the high elasticity of cloud computing, although work has also been done for highly scalable one-shot analytics. From a SOA point of view, work has been done to integrate Big Data analytics tools to existing SOA [76], but few frameworks have worked on separating individual processes into individual services. This means that rather than building an application as an assembling of services. with the exception for K. Abuosba [26], the majority of the proposed works produce the services and integrate them into their existing framework.

2.4.2 BDAAS TOOLS

Big Data relies on 2 major families of tools: distributed storage and distributed processing. Distributed storage is mostly managed by the growing trend of NoSQL data bases in particular

document store, key-value stores and extensible record store. Distributed process is mainly managed by MapReduce programming model with managers such as Spark or Hadoop. In the two following subsections, we present Hadoop as a Service and NoSQL as a Service tools.

2.4.2.1 HADOOP AS A SERVICE

Hadoop is a popular MapReduce framework for Java. MapReduce is a programming model used to perform distributed parallel processing. MapReduce is composed of 2 basic components: the Mapper, which is run on every item of the wanted data collection and is used to pre-process and emit the data to the Reducer. The latter aggregates and processes the data collected from multiple Mappers into a new data collection for the user to interpret. MapReduce programs are used at the data cleaning, data aggregation and data analysis levels.

According to R. Stata [82], MapReduce frameworks require substantial knowledge to use and operate. This has spawned the development of multiple HaaS (Hadoop as a Service) ranging from basic Hadoop frameworks and virtual machines to full service support options that include monitoring and tuning, passing by the ones providing “run it yourself” preconfigured MapReduce programs and environments. R. Stata [82] offers 5 criteria for to identify the right HaaS:

- HaaS should work for both data scientist and Administrators: data scientists require rich functionalities to process and analyse their data. Ideally, one wants the HaaS to provide the common tools used by data scientists such as Hive, Pig, R or Mahout and run them on Hadoop. Hive [83] is a data warehouse software running on Hadoop, it is particularly well known for its SQL like HiveQL query language easing the transition from the common relational database queries to the newer distributed model. Pig is a high-level platform providing as simple procedural language programming to run MapReduce programs. R is

a popular programming language for data analyst, its main standout is it native support of matrix arithmetic's, very useful for many data analytics algorithms. Mahout [83] is an library of Hadoop software designed for machine learning used to perform collaborative filtering, clustering, categorization/classification.

The administrators, on the other hand, requires an easy and straight forward to use interface to manage the platform with low level monitoring detail managed by the provider.

- HaaS Should Store "Data at Rest" in HDFS: HaaS should prevent delays and time load and transferring data around. On this front, cloud IaaS will often have edges as their object storage is often compatible with Hadoop as is the case with Amazons S3[35] or Openstack Swift[36].
- HaaS Should Provide Elasticity: different jobs require different workloads. The HaaS should adapt the resources, else one needs an admin on call or delay the jobs.
- HaaS Should Support Non-stop Operations: Hadoop can be a complex environment that brings its own list of challenges. They have to restart subprocess to prevent rebooting a hole job, jobs starved of resources when resources are available and deadlock. Nonstop operation should address these issues.
- HaaS Should Be Self-Configuring: the point of cloud services is to limit the amount of set up and configuration required by the user. Thus, HaaS should self-configure to the right number and types of nodes for a particular job.

[35] https://aws.amazon.com/s3/
[36] https://docs.openstack.org/swift/latest/

Many HaaS exist today [84], such as amazons web service Amazon Elastic MapReduce [85], a MapReduce framework which can be run on Hadoop, Spark or Presto; EMC² [86] which drives Dell's data warehousing and analytics; IBM®'s InfoSphere® BigInsights™ [87] which proposes wide range of tools to explore the data such as JaQL query language; BigSheets [88], an excel like data exploration tool and an SQL engine; Microsoft HDInsight designed to run Hadoop, Spark, and R on Windows Azure; Altiscale [89], Cask [90], Cloudera CDH [91] which provide a SQL engine and administrator tools like roles-based access control and security integration; FICO® [92] Big Data Analyzer which provides tools for business intelligent and analytical tools for business users; Google's BigQuery [93], a self-managing Hadoop tool which takes advantage of the very efficient google platform; Hortonworks Data Platform [94] which provides a wide range of data management tool; Infochimps™ [95] cloud which provides capabilities to scale elastically and tools like Hive, Pig[37] and Wukong[38] for data exploration; MapR [96] provides the complete set of tools proposed by Apache for Hadoop; Datadog [97] which provides a unified view of the data from servers, databases, applications, tools and services; Pentaho [98] provides a set of tools to run the data life cycle from end to end and a visual development tool to assist; Terradata [99] claims to provide tools for all scales and user level of skills.

2.4.2.2 NOSQL AS A SERVICE

On the storage end many new NoSQL databases implementing one of the models seen in section 2.3.2.3. Many commercial and opensource solutions (**Erreur ! Source du renvoi introuvable.**) have been proposed with a limited list presented hereafter, inspired from [100]

[37] https://pig.apache.org/
[38] https://rubygems.org/gems/wukong-hadoop/versions/0.2.0

yet we do not consider that one or another strategy is advantageous and disadvantageous but that it can respond to specific objectives and requirements. The important is to be aware as programmers of the properties provided by different stores so that she can make decisions about the store to use and the implications in terms of the properties and functions it provides. In Table 4 the comparison is done with respect to storage strategies and properties.

Table 4: Comparison of NoSQL as a Service tools

Database	Description	Storage strategies	Properties
Voldemort [101]	Advanced key-value store supported by LinkedIn. Supports a Multi-Version Concurrency Control (MVCC) for its data, provides hashed sharding.	+ Support storage engines + MVCC	+ BASE
Riak [102]	Described as an advanced key-value store with limited functionalities of a document stores like storing multiple field with JSON but lacks querying and indexing mechanism on anything other than the key.	+ MVCC + RAM stored, but disk backup + Updates	+ BASE
Redis [103]	Key-value store written in C. It requires client library update when protocol updated.	+ Fast	N/A
Scalaris [104]	Written in Erlang.	+ Ranged sharding + RAM stored	+ ACID
Tokyo cabinet [105]	Key-value store which supports 6 models: hash indexes in memory or on disk, B-trees in memory or on disk, fixed-size record tables, and variable-length record tables.	+ No auto sharding	+ ACID
memBase	Based and upgradable from on MemCached an in-memory indexing system.	+ Disk storage + Remarkable flash performances	N/A
SimpleDB [106]	Document store produced and used by cloud in the Simple Storage Service (S3) and Elastic Compute Cloud (EC2) services.	+ Amazon support + SQL based query language	+ BASE

		+ No embeded documents + No automatic sharding	
CouchDB [107]	Document store supporting a complex data structures included in the set of data types of its data model. Queries are performed using MapReduce views.	+ MVCC + Built-in MapReduce + JavaScript queries	+ Limited ACID functionalities
MongoDB [108]	Document store written in C++ supports auto sharding, atomic operators, dynamic queries. Uses replication of backup and does not allow dirty reads.	+ Field level atomic operation + A complete query language with atomic operations + Auto sharding + No query scaling + Built-in MapReduce	+ BASE
Terrastore [109]	Document store and provides tools to automatically redistribute data as nodes are added.	+ Automated redistribution of nodes	+ BASE
HBase [110]	Extensible record store written in Java and patterned after Google BigTables but using HDFS instead.	+ HDFS + Row level atomic operations + Transparent distribution system + Log and lock consistency	+ BASE
HyperTable [111]	C++ extensible record store. Updates are done in memory then forwarded to disk.	+ MVCC + Log and lock consistency + Requires a distributed file system	- BASE
Cassandra [112]	Extensible record store written in Java and used by Facebook. It supports a weaker consistency check than MVCC. Proposes the concept of super column for high-level of grouping.	+ Column level atomic operations + Super column's providing and extra layer of grouping	- BASE

Regarding *storage strategies* we consider data residing in main memory or in disk with explicit or implicit persistence, the exposure of an interface for tuning sharding strategies for distributing data across cluster nodes, and the provision of declarative or programmable queries. Regarding data mangagement *properties*, we consider properties as consistency, availability, fault tolerance ensured by two existing protocols ACID and BASE adopted by existing data management systems.

Whereas in general ACID properties are normally built-in within systems, that eventually export interfaces to tune them (for example, the level of consistency or isolation), BASE properties provide eventual consistency as model with availability and durability ensured through replication. Adopting one solution or the other relies on the complexity of data management required: performant reads/writes of huge data collections are ensured through BASE whereas dependable, strict consistent states of the database are ensured by ACID. Regarding performance it is always all about tuning the systems and programming burden. ACID oriented systems tend to demand less programming burden to ensure these properties, as almost everything is externalised and delegated to the system. BASE oriented systems call for expert programming skills where decision making is necessary to tune solutions that can respond to performance expectations of data consumers. This implies less possibility for delegating data management completely to the system.

2.4.3 SYNTHESIS OF THE STATE OF THE ART REGARDING BDAAS

Big Data as a Service includes a wide range of topics from service model specialized for Big Data to specific tools specialized in extracting value from the data, passing by complex architectures designed to take advantage of the scalability and elasticity of the service model.

On one hand, we can see that there are dozens of specialized tools for Big Data analysis each with their strength and weakness. On the other, most service oriented Big Data architectures are attempts to integrate Big Data to an existing platform or are solely focused on the specific task at hand. There is little work done in the providing services focused on proposing a generic architecture to produce a full Big Data analytics workflow.

2.5 CONCLUSION

In this chapter we investigated most of the aspects of Big Data in the cloud. In section 2.1, we defined what Big Data is and its characteristics. In section 2.2, we presented two Big Data life cycle models. In section 2.3, we investigated data curation and data exploration. For data curation we saw why it's important to improve data quality and saw how it can be improved. We concluded that there is a lack of data curation model for data analysts to use when investigating their data. For data exploration, we noted that when working with Big Data completeness and exactitude are not as useful characteristic as the capacity to perform many queries and the capacity to find patterns requiring a broader exploration method. In 2.4, we explored Big Data on the cloud, which as some say is a match up made in heaven combining Big Data tools in an environment enabling scalability. It also introduced the aspects of services-oriented architecture to provide Big Data curation services.

Exploiting Big Data on the cloud for the decision making requires, ironically, a lot of decision making. In both the life cycles seen in section 2.3, there are decisions to be made, decision relying on information from the data and how the data is going to be used. Answers for these questions are often found using data curation and data exploration technics, and these have been focused on the important task of extracting value from the data. Whilst these are key task, many other decisions have to be made which are not focused on extracting value but are

instead focused on making sure the data is easily accessible and optimized for the tools produced as a consequence of the data exploration. Tools are needed for the data manager to explore the data to make sure the databases set up are the best possible. What is more, as opposed to traditional database applications which rely on the end goal to answer questions, Big Data applications rely on information from both the end goal, if there is one, it is possible the data analyst is looking for one, and information about the data. This means one also needs information from the data to choose such application.

The objective of this PhD is 2-fold; first we are going to define service used in Big Data and how to compose these for a Big Data application. Second, we are going to define a data curation model designed for data analysts and data managers to investigate the data and help them make quick meaningful decision on their data and the type of services would be most effective to use.

3 CURARE: SERVICE ORIENTED ARCHITECTURE FOR CURATING DATA COLLECTIONS

As discussed in the state of the art, Big Data processing combined with cloud computing can be an appropriate approach for dealing with the execution of operations that can require important computing and storage resources because of their complexity and the volume or velocity characterising the data collections. The cloud elasticity and scalability can provide an interesting solution for running experiments with different requirements where the underlying infrastructure does not need to be tuned manually since cloud providers can deliver a well-suited solution easy to evolve. In this spirit, we propose CURARE a service-oriented architecture for exploring and curating Big Data.

Accordingly, this chapter is organised as follows. First, in section 3.2 we give a preliminary view on our data curation approach and how it guides the design of CURARE. Then we look into the service-oriented architecture we propose in section 3.1. We look at how to deploy CURARE on top of a cloud in section 3.3. Finally, we finish by concluding in section 3.4.

3.1 DATA CURATION PROCESS

To explain our vision of data curation, let us consider the following scenario. A data analyst is given access to a collection containing data from the towns road infrastructure ranging from car counter and taxi GPS tracking to police traffic reports. The objective is to investigate how

to use this data and later releases produced in real time to make a useful application for the city. Immediately, she realises the challenge ahead before being even able to investigate the content because as stated by IBM [25] “70% of the time spent on analytic projects is concerned with identifying, cleansing, and integrating data due to the difficulties of locating data that is scattered among many business applications”. To make her life easier, she has to reorganise the data to be able to more easily compare collections and latter choose data and methods for her application. She decides to focus on how to organise and store the data.

This process is data curation that involves obtaining information from the data collection, looking at the available meta-data, exploring the data collections and maintaining information as the collections evolves. This process is highly dependent on managing meta-data.

In our approach we view data collections as data sets composed of releases (data produced at one time) and releases composed of items (individual documents). So, let us first look at what type of meta-data is available and how it can be used.

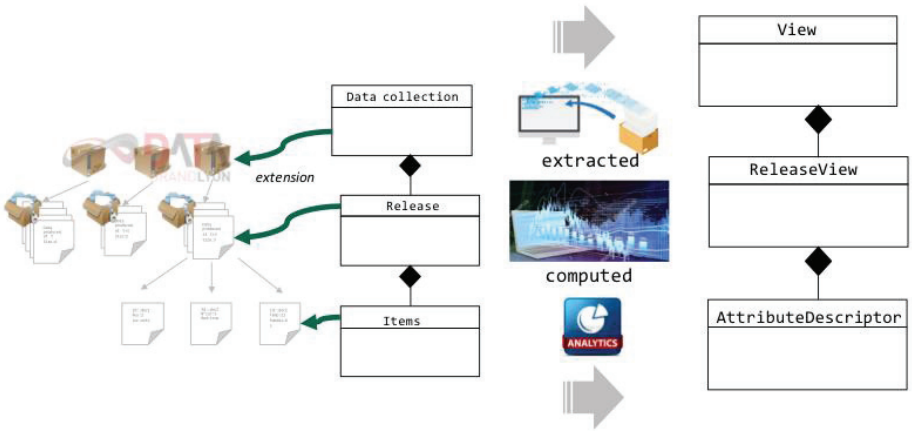

Figure 13: Collection view model

3.1.1 DATA COLLECTIONS STRUCTURAL META-DATA

Data collections can be described using 2 major types of meta-data:

- Explicit: this meta-data is self-contained associated to the data collection and its provider. This can be something as simple as the name of the file "police traffic reports" or the description of the collection. In the case of the police report of our example, it would say something: "*this collection contains documents describing traffic event reported by police officers, we insure that the report a linked to other report when a link of causality can be established*".
- Implicit: requires an analysis process to be extracted. This ranges from statistical information on the values of individual attributes to semantic relations between attributes. For example, in the case of police report, count the number of times a particular attribute is not used within these reports.

In our vision, we propose the concept called data collection (Figure 13), to model an entity that contains information how the data is produced and by whom; and the concept view exposing with statistical and implicit information on in the individual attributes of the items of the data collection.

In our example, the entities producing the data would provide various information of their data collection. The company may describe how the data is collected, with which frequency the data is modified, who to contact for questions. For example, in the case of the police logs the person to contact will probably be the officer logging in and linking all the police reports. The `view` level may be produced by some independent entity and as such will provide information on the data collection items. For instance, how the `view` was produced, e.g. the

algorithm used to compute some statistical operation, or to discover the casual relation between two attributes.

3.1.2 DATA COLLECTIONS STATISTICAL META-DATA

Data sets evolve over time. To help users tracking the evolution of a data collection over time we assume that data collections represent in fact a set of releases. A release corresponding to a data set produced at a specific time. For example, the Sloan Sky Server shares annual releases of the observations of the universe to be used as open data by scientists. A release can be processed to extract explicit and implicit meta-data that can describe its content. Such meta-data can be grouped as `releaseView` (Figure 13).

So, following the previous example, this allows our data analyst to classify each data collection according to the moment in which data sets are shared (i.e., released). The police reports in our example, are put online every day at 8 pm.

3.1.3 EXPLORING DATA COLLECTIONS META-DATA

Data collections can contain key insight to the functioning of some systems. But getting that information can be challenging to extract. The large variability and variety of Big Data can make it challenging to extract the structure of the items (extract and understand the role of every attribute in a record) and even more identifying useful patterns in the data for determining whether a data collection can be used for performing a specific type of analysis and thereby obtain an understanding some phenomenon. The job of data exploration is intended to go through the content of data collections to determine whether they are useful to perform some study. In our approach, the notion of `views` will help in this process by providing a standard for in which to identify all the attributes as well as their values variability.

The items within a release are described, in our approach by *attributeDescriptor*'s (Figure 13) which group the meta-data of an attribute particularly distribution and statistical information about the values of a given attribute across the whole release (set of items).

After reorganizing the data, a data analyst is given a far better insight to how the data is usable. She can now see how many times each ring counter has been triggered. By varying the size of the `releaseViews`, she is able to plot the use of each ring counter overtime, this allows her to identify the most used counter and distribute them evenly across the node of her database to ensure optimal load balancing. For example, the police report data collection reveals to have an attribute allowing it to be linked to another report. She chooses to keep related reports in the same physical machine to reduce overhead when queries must be evaluated.

Now that she has reorganized the data into a readable format she will have to choose how to manage the data and use the information. Cloud service models [113] allow for a quick and easy development of software infrastructure and fairly simple solutions to swap out elements, e.g. services, for other more appropriate ones. What is more the scalability and elasticity of cloud computing [114] can make a great combination. This requires dividing tasks into individual services. How we divide the task of each service and type of service will be addressed in the next section.

3.2 DATA CURATION ENVIRONMENT: GENERAL ARCHITECTURE

Figure 14 shows the service-oriented architecture of CURARE the Big Data curation environment that we propose. The services of CURARE are organised into four layers that represent the order in which the major types of operations used in the Big Data: harvesting and cleansing, storage and access, processing and exploration.

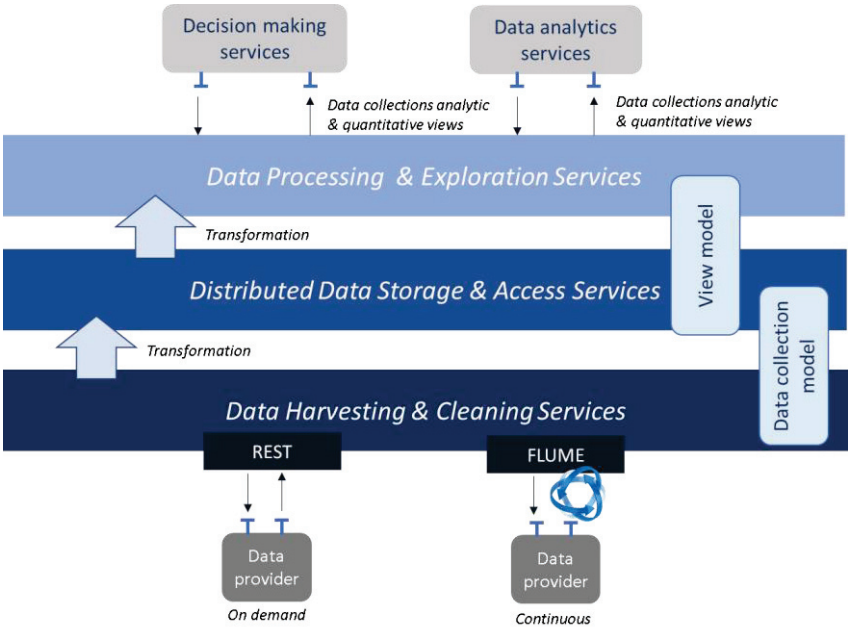

Figure 14: Curare service's layered architecture

3.2.1 DATA HARVESTING AND CLEANING SERVICES

Data harvesting and cleansing services' job is to collect data from various on demand or stream sources. At this level, CURARE relies on hardware (disk), infrastructure and services (acquisition, cleansing, information extraction) that collect the data from the various sources using tools adapted to the sources characteristics. Sources come in many forms and origins: Web services collecting crowd sourced data, databases, data stream. These sources use different technologies and access logic. Thus, CURARE uses underlying infrastructure services for interacting with these different types of sources for harvesting data.

In our example, the police logs would be collected using an http request eventually through a REST architecture. Instead, using Twitter[39] more advanced tools like Flume[40] may be required for implementing streams consumption and authentication protocols for interacting with this service. Extracting and discovering meta-data and grouping them in `attributdescriptors`, `releases` and `views`, help the data analyst choose her pre-processing and cleaning services provided in the second layer of CURARE. Depending on the degree of rawness of harvested data, different pre-processing and cleaning tools may be required. For example, well documented police reports would require little to no clean up before being stored but data from a near Infrared sensors may require more work removing baselines, noise and extracting the actual information in the spectrums, this would require a short chain of service to process the data before being stored.

3.2.2 DISTRIBUTED DATA STORAGE AND ACCESS SERVICES

For most of the history of computer science, Moore's law [115] has been observed to double the processing power every 2 years. Now, it seems we are reaching the limits of silicon processing power. What is more with the development of the internet and in particular internet of thing, the amount of data collected is growing at an exponential rate. The growth in the volume out passes the growth in processing speed computers. Added on top of that, the processing complexity required in dealing with the velocity, variability and variety of data available in Big Data makes the storage and processing of the data in a reasonable time simply impossible on a single machine. The solution is to use more than one computer. Multiple data stores have been designed for distributed processing and storage over a network. But these

[39] https://twitter.com
[40] https://flume.apache.org

bring extra design challenges to optimise the usage of the data as networking is still one of the slowest processes in IT thus traditional processing method like relational systems or some statistical methods relying on analysing the data as a whole become particularly inefficient.

Another challenge with Big Data is more data often means more confusion. Meta-data, is vital in understanding any piece of data but even with good meta-data a large matrix of numbers remains challenging to understand. Unfortunately, this matrix of data often contains key information to vital in some of the strategies involved in Big Data. In fact, even the objective can be unclear without exploring the data. Thus, an abstract representation of the data is needed to help make decision on services and strategies to use for storing and exploiting them.

As said before, to provide an abstract representation of a data collection, we model it as consisting of a set of releases, and releases consisting of a set of items. A release would be for example a set of CSV files containing the number of cars populating specific zones of a city at specific hours of the day. CURARE provides tools to export raw data (CSV, JSON, text, records) packaged into several releases. Every release is in general a zip file containing a dataset with an explicit or implicit structure. The meta-data associated to data collections and their releases is collected from the provider that normally tags them with some information (release date, size, producer). Meta-data can be given manually by a data scientist curating the collection (e.g., provenance) and it can be extracted by analysing the collection (e.g., the structure of the records composing the releases, their type or format).

Data collections pre-processing can help a data scientist decide how to archive and organize them in a storage support. Given the data collection releases volume, storage and processing cannot be done on a single machine. Multiple data stores have been designed for distributed

processing and storage. Therefore, CURARE provides services to process data releases to provide quantitative and analytical description to decide whether to partition and duplicate them completely or partially according to consumption patterns and storage space availability. For example, if a document shows strong usable relationships, it would be smart to adapt your data distribution strategies to keep those pieces of data together. CURARE services help by showing the data analyst where to partition the data in such a way it answers effectively the data analyst queries and maintains a balanced set of data nodes by insuring they all have the same amount of data.

As an example, the reports provided by the police department are linked to other reports when a causal link can be established. The existence of the relationship makes storing these documents on the same machine/shard judicious to maintain efficiency when making queries. Similarly, if it is established that queries made on the ring counter have tendency to focus an area of the town, it would once again be judicious to make sure every document from the ring counter of a predefined area be stored in the same machine/shard. Defining the objective of storage is one thing but ensuring an efficient and balanced database while respecting these constraints is another. The `attributDescriptor`'s of the items of a release in a data collection provide a great deal of help by providing a visual aide on how the data is distributed allowing the data analyst to make an efficient model fairly quickly to distribute the data efficiently whilst following the previous constraints.

3.2.3 DATA PROCESSING AND EXPLORATION SERVICES

There are multiple reasons why one cannot easily exploit raw data collections. The data may be incomplete, uncertain or unclean. This is hard to know without further exploration of the data, but more importantly the events one is trying to detect probably will not stand out but

be the consequence of complex set value scattered within the data. Solutions for these issues [116] include: data analysis techniques [61] and data curation techniques [59]. However, these techniques (as discussed in Chapter **Erreur ! Source du renvoi introuvable.**) rely on having an environment like CURARE to explore and understand the data. Exploring and understanding data can be long and resource intensive. A quantitative view of the content of `releases` is necessary to provide data analysts with aggregated views of the content of a dataset. For example, if we were to investigate document describing road event in the city. Let us consider a collection with 200'000 documents and between 50 and 200 different attributes. Simply trying to understand each attribute requires hours going over each document to understand what is that attribute talking about and that before we even investigate usable relationships between attributes. By investigating a `view` of the collection (i.e., quantitative and semantic meta-data of the collection), as proposed in our approach, we can quickly identify all the existing attributes in the data collection, perform queries to help identify useful ones and the statistical information provides information on what type of algorithms will have to be run to first pre-process then analyse the data.

3.2.4 BIG DATA ANALYTICS AND DECISION SUPPORT SERVICES

The whole point of Big Data is to identify and extract meaningful information. Predictive tools can be developed to anticipate the future or model and understand phenomena. The role of the Big Data analytics and decision support services in our infrastructure is to provide data analytics solutions for predicting events, trends or for decision making tasks. For example, regularly observing an increase in the population in one place and traffic jams 30 minutes later we can deduce cause and effect situations and intervene in future events, so the taxis avoid and evacuate that area.

This service-oriented model allows for quick deployment of services and quick replacement since it takes advantage of the elasticity and agility of cloud computing. Each service could be described by one of the many service description model, MADONA for example [113]. This is appropriate for companies since simply the capacity of elastically testing architecture is on its own a big lure for companies towards cloud computing, as stated by Amazon Web Service at Lyon the 31 of May 2016. On the other hand, as opposed to production service for which the criteria and objective are clearly defined, Big Data applications are dependent on obscured information in the data to the point that even the use of the data may not be clear. The data curation model we propose enable the description of data set to help identify the service required to run the application. We are now going to look into deploying a service-oriented architecture with the objective of running Big Data applications.

3.3 DEPLOYING CURARE ON A TARGET ARCHITECTURE

Given that the architecture of CURARE consists of services of different types organized into layers, we chose an adapted architecture for deploying the system: the cloud. The following sections explain how to deploy CURARE services on top of cloud services that serve as underlying infrastructure for the data curation process implemented by CURARE.

3.3.1 CURARE SERVICES AND UNDERLYING DATA SCIENCE VIRTUAL MACHINE

As shown in Figure 15 CURARE services are deployed on top of a Data Science Virtual Machine which is a cloud image, pre-installed, configured and tested with several popular tools that are commonly used for data analytics, machine learning and AI training. The goal of the DSVM is to provide data professionals at all skill levels and in all roles with a friction-free, pre-configured, and fully-integrated data science environment. CURARE services run on top of the services provided by the DSVM to perform the data curation tasks. Depending on the

characteristics of the data collections and the exploration and decision-making requirements some services are used rather than others. As said before, cloud services [113] ease the development of software infrastructure and fairly simple solutions to swap out elements, e.g. services, for other more appropriate ones. The CURARE services can be configured for adapting the use the underlying data science tools.

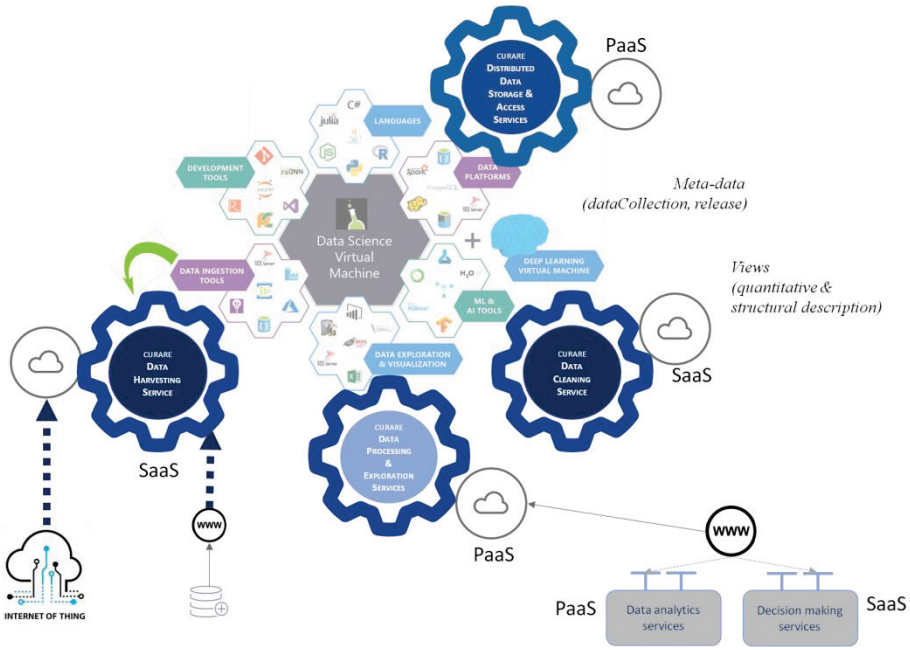

Figure 15: Curare service cloud architecture

When deploying a service-oriented architecture on the cloud there is three major service layers available in such an architecture: infrastructure as a service (IaaS), platform as a services (PaaS) and software as a service (SaaS). As shown in Figure 15, CURARE services according to their function within the three layers of its architecture are weaved within the services of the different levels of the DSVM: distributed data storage & access services and data processing & exploration services are supported by PaaS; data harvesting and data cleaning services are

supported by SaaS. The whole DSVM gives access to IaaS required for accessing to disk, CPU and RAM resources for executing the jobs performed by services.

Infrastructure as a Service (IaaS) cloud layer

The IaaS consists of services designed to deploy computing, storage and memory resources on the cloud. These services are run by the cloud provider and have for sole role to provide and manage resources for the user. CURARE's layers are deployed on top of IaaS services that provide storage, memory and computing resources necessary for its services to process data for extracting the meta-data according to our approach. Depending on the type of methods and algorithms used for extracting structural and statistical meta-data, and on the characteristics of the data collections, CURARE services might need more or less resources. The elasticity of the cloud adding more resources whenever required is an important feature exploited by CURARE. For example, let us say streams production by a source suddenly increases its rate CURARE would immediately instantiate a new on sight data curation and split the load between the 2 services. The IaaS follows its requirement making decisions on whether to add more resources that can let these instances perform well.

Platform as a Service (Paas) cloud layer

The PaaS consists of tools designed to take full advantage of the cloud environment scaling but require a substantial amount of development to use. Database and framework qualify as PaaS. Distributed data storage and access services are weaved with PaaS. They assist data architects to make decisions on the resources to allocate disk for managing data and meta-data persistence and for allocating CPU cycles for processing data collections and extracting meta-data using greedy analysis and processing tasks. Since the DSVM provides different tools for performing similar jobs the CURARE service provides an integration layer that enables

choosing specific tools according to the characteristics of the data collections to store and process and of the analysis programs complexity. Data analysis and processing services are PaaS services giving access to data analytics platforms like Spark and Hadoop.

On the PaaS level, the most predominant CURARE service is the distributed data storage and access service providing the storage and processing power to manipulate the data for other service to use. It is the service providing the bulk of the resources used to process data for extracting meta-data and performing data curation tasks. This layer runs tools that allow other service to query, manipulate, store data across multiple machines. For example, distributed NoSQL databases like MongoDB that we chose, exploiting its MapReduce functionalities and built-in query language; and like HDFS for distributing archived data harvested by services in the first layer of CURARE, on multiple machines both for robustness and for horizontal scaling of resources. Whether distributed across many machine or only a few, CURARE services deployed in the PaaS layer are the most resource intensive as it needs large amount disk space to store the large amounts of data as well as both RAM and CPU since this service will be running all the queries and data manipulation.

Following the previous example, the choice of distributed data storage and access services is dependent on the type of data. For example, the variability in structure of police reports makes a document store like MongoDB a prime candidate in which by applying smart sharding techniques, she can ensure related document remain in the same shard. On the other hand, the loop counters provide a fairly simple and standard data structure making an extended record database interesting.

Software as a Service (SaaS) cloud layer

SaaS in the cloud perform specific tasks and are expected to propose fairly simple configuration to run. The most known service services would be tools like Wordpress[41] or Mediawiki[42]. In CURARE data harvesting, data processing and exploration, data analytics and decision-making services are SaaS.

- *Data Harvesting Service* uses tools like flume to continuously collect streams, and then send them on to the storage service. Other data harvesting services adapted to interact with on demand data providers on the Web are used. The data harvesting service provides an integrated interface that can be adapted according to the data sources used for harvesting data collections. For enabling stream harvesting, this service is instantiated with cache memory that acts as buffer in the case data arrive to quickly for later services to process.
- *Data Cleaning Service* is responsible for pre-processing data before it is stored. This includes information extraction, data cleaning and on sight curating services. The machines running these services require added processing power in the form of computer processing units to run computationally costly processes for cleaning data. For example, near infrared gas analyser used by environmental control has very awkward spectrums to interpret, often requiring a specialist to interpret but can determined using specialized chemiometry techniques. As a consequence, after collecting that data, it would be forwarded to a service capable of identifying all of the compounds measured by the machines.

[41] http://wordpress.com
[42] https://www.mediawiki.org/

- *Data processing and exploration services* interface with the data storage and access service to provide more advanced data processing taking whole collection into account. These services communicate with de data storage service via specialised database protocols, and they use PaaS services like Hadoop for executing processes.

- *Data analytics services* run PaaS services perform analysis in parallel settings. The analysis tasks can use high level languages like HiveQL or Pig Latin designed to interface with the data storage service that puts data into HDFS for performing data analytics processes. The analytics services can also use low level languages for implementing programs and using data mining and Artiificial Intelligence libraries that can run under the MapReduce programming model and then executing them on the suited execution environments (e.g. Hadoop, Spark).

3.3.2 CURARE DATA CURATION LIFE CYCLE

There is two major parts of the life cycle of the data curation, before data collections being stored and after being stored. Before being stored, the data collection goes throw the process of being collected, cleaned and stored (see Figure 16).

The first step is the data harvesting. Here, there are two types of data collections to be harvested:

- Stream data are continuously harvested using a protocol `subscribe(); receive(); stop()` as shown in the upper part of Figure 16. When the service is instantiated it subscribes to the data stream, it then gets regularly data from a channel until the client stops the connection.

– On demand data are obtained establishing a classic protocol `connect(); get()` and `closeConnexion()` as shown in the lower part of Figure 16. The protocol is performed periodically to request new data collection releases from the data source.

In both cases, the collection service sends the batch of items to the CURARE cleaning service using the corresponding method `post()`.

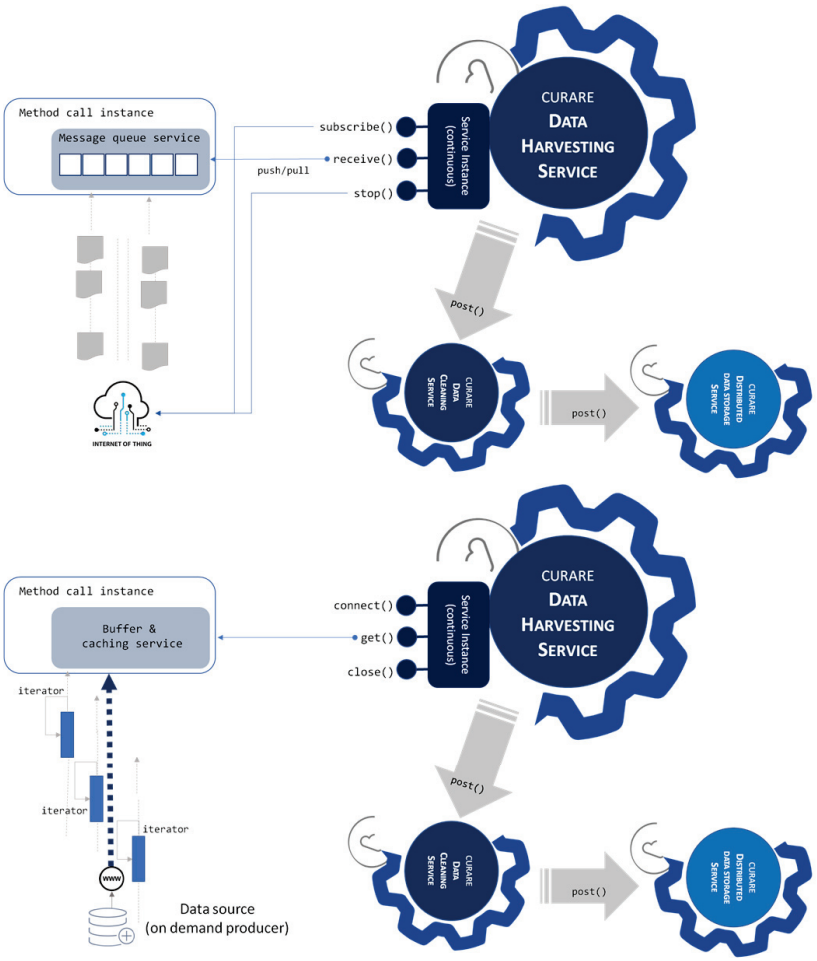

Figure 16: Data harvesting and Pre-storage life cycle using CURARE services

The CURARE cleaning services run different logics designed to clean or extract the information from the data. The new version of cleaned data collections is sent to the storage services. If the collection exists new data can be appended to an existing release or a new release is created. Otherwise, a new data collection is created, and the data are associated to a new release. Data collections and releases are stored using the distributed storage services of CURARE. The storage services use NoSQL stores that shard and distribute the data across different disks provided by the cloud according to strategies chosen by a data scientist responsible of curating data collections. Therefore, the `views` created by the processing services help her to choose well-suited strategies for choosing the best storage strategy.

As shown in Figure 17 the core of CURARE are services devoted for maintaining persistent data collections (raw data) and associated structural meta-data describing the collection, its releases and items and statistical meta-data grouped as `views`. Data collections are processed by CURARE analytics services for generating all these meta-data therefore they have to be distributed across local stores and memory using a distributed file system, in the case of CURARE, HDFS. The curation process supported by CURARE storage and analytics services is recurrent and it is mainly centred in a data exploration workflow with three main activities:

- *harvesting and cleaning data* ensured by CURARE services weaved with underlying analytics and storage cloud services deployed on top of customized IaaS as shown in Figure 15.
- *curation and storing data collections* ensured by analytics and storage CURARE services weaved with underlying data processing services able to exploit the computation power provided by the cloud shown in Figure 15. To perform this, the processing service interacts with the data storage service to process the data into a form that bring up information of the meta-data in the data collection. This process creates and

updates a handful of collections updating meta-data and the releases of these data collections.

– *data collections exploration* (filtering, querying and visualization) ensured by data analysis CURARE services running on top of data science services provided by the cloud. CURARE data processing and exploration services can work on raw data and on `views` to help data analyst have a consolidated vision of the data collections that can be used by data analytics and decision-making services on top of which a data architect can build target solutions (see Figure 17).

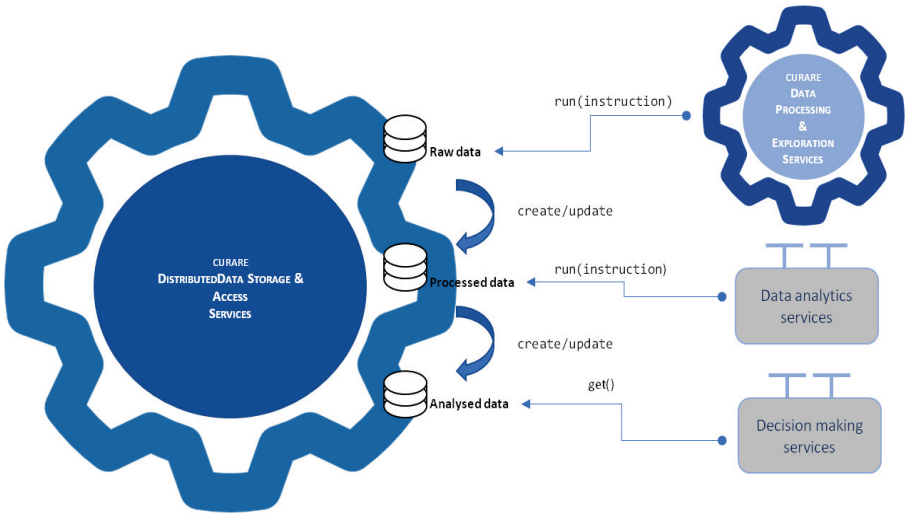

Figure 17: Post processed data cycle

3.4 CONCLUSION

This chapter presented CURARE, a data curation environment based on the notion of data collection view. The contribution of our work adheres to novel approaches for dealing with data collections by adopting an exploration philosophy. CURARE implements a *data collection model* providing concepts for representing data collections as sets of releases of raw data,

where each release consists of a set of items (e.g. records). The *data collections model* is used by the data harvesting and cleansing services for representing structural and context meta-data related to collections (*provider, objective, URL, item structure*). Thereby, CURARE provides abstract `view` of the releases related to a data collection and gives the possibility of exploring the releases without having to zoom in item per item.

Cloud computing and large scale Big Data processing and management make an interesting duo as the elasticity in resource of cloud combines well with the need of scalability in Big Data. The chapter has introduced the service-oriented architecture of CURARE and how it can be deployed on the cloud. The service-oriented architecture of CURARE provides a very agile ecosystem of services allowing some services to be replaced with only minor changes to other services.

In conclusion, CURARE provides tools for supporting data scientists to explore the content of raw data collections and make technical decisions or integrate datasets by combining these collections to be used for data centric sciences experiments.

4 DATA CURATION AS A SERVICE FOR DATA COLLECTIONS

This chapter introduces one of the contributions of our work, a View Data Model that provides concepts defined as data types for supporting a data curation. Accordingly, the chapter is organized as follows. Section 4.1 introduces the general principle adopted for modelling data collections. Section 4.2 provides general definitions, particularly a types system, required for defining our `view` model. Sections 4.3 introduces the `view` model consisting of data types for modelling (1) data collections consisting of releases and items and (2) meta-data defining `views`, `releaseViews` and `attributedescriptors`. Section **Erreur ! Source du renvoi introuvable.** defines data collections and `views` definition functions. Section 4.5 defines functions for manipulating `views`. Finally, Section 4.6 concludes the chapter discussing final remarks.

4.1 MODELLING DATA COLLECTIONS: GENERAL PRINCIPLE

In order to illustrate the role of data curation we use the following example. Let us consider that we want to organize a data collection in the Grand Lyon portal[43] related to the traffic status in the city boulevards. The data collection can grow bigger as new releases are produced, and this means that it is not possible to store it in one local disk but across farms of storage (see Figure 18). A balanced and smooth fragmentation, meaning that fragments should be balanced, related fragments must be collocated, and fragmentation should not

[43] https://www.grandlyon.com/

damage availability. In order to achieve these requirements here are some of the question to be answered: Which attribute can be used to shard the collection? Is there critical data with particular availability requirements? Should some fragments be collocated?

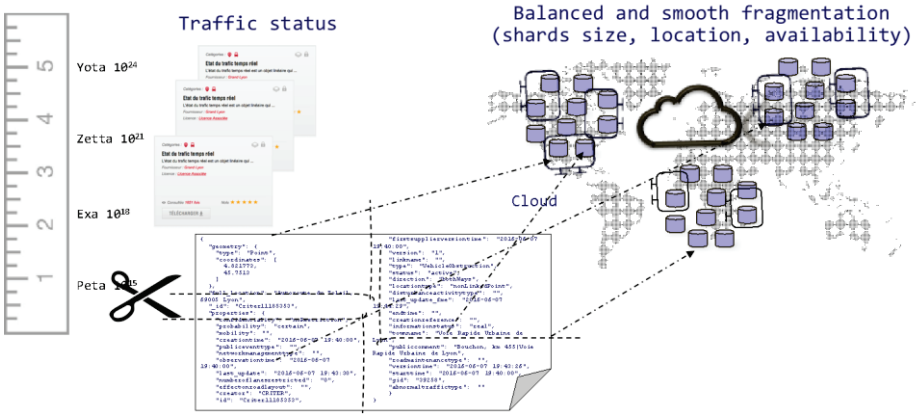

Figure 18: Data sharding example

The decision made for these questions depends on the structure of the data collection items, on the distribution of values and on the relations between attributes. This information is not readily available and has to be discovered in the data collection to create data that describe the data, i.e., meta-data. Since meta-data is the back bone of the curation process, we proposed a `view` model giving information that can be used to curate them and support data analyst decision making. For this purpose, our approach is to propose concepts which describe the content characteristics of a data collection release:

- `Collection` is a data structure that divides data collections into releases. This means we can track the evolution of the data collection overtime.
- `View` is a data structure that provides an abstract description of the content of data collections (including their different releases) by grouping meta-data of different types (Figure 19).

A collection would be for example a folder of CSV files storing data on the number of cars populating specific zones of the city. The collection maintains information on the context of data production and storage and a list of releases. A release would be for example a CSV file from the previous folder containing the number of cars populating specific zones of the city at specific hours of the day. The release being uploaded in the Grand Lyon portal at some date and time. The data collection abstract representation provides meta-data regarding the frequency in which releases are updated, whereas a release concept would represent the conditions in which data in a release were produced, like data production rates and the size of the release. An item represents the structure of every element of the release, for example a line of a CSV file.

We consider that the meta-data representing a data collection, a release and its elements can be retrieved from the meta-data contained in the files, from the provider, manually or automatically extracted by processing the data. From this we produce an analogue structure providing a quantitative and analytical description of the data collection. A `View` provides information on how the `view` is produced, a `releaseView` essentially stores the description of the data in releases, and finally `attributeDescriptors` provide statistical a contextual information extracted from the attribute in the items of the data collections. Thus, `attributeDescriptors` provide statistical and quantitative data, which is important to technical questions; `releaseViews` allow to track this information over time.

These concepts complete the representation of the data collection and maintain information and knowledge of its content without having to scan it and explore it item per item, which can be costly in terms of time, and of computing resources.

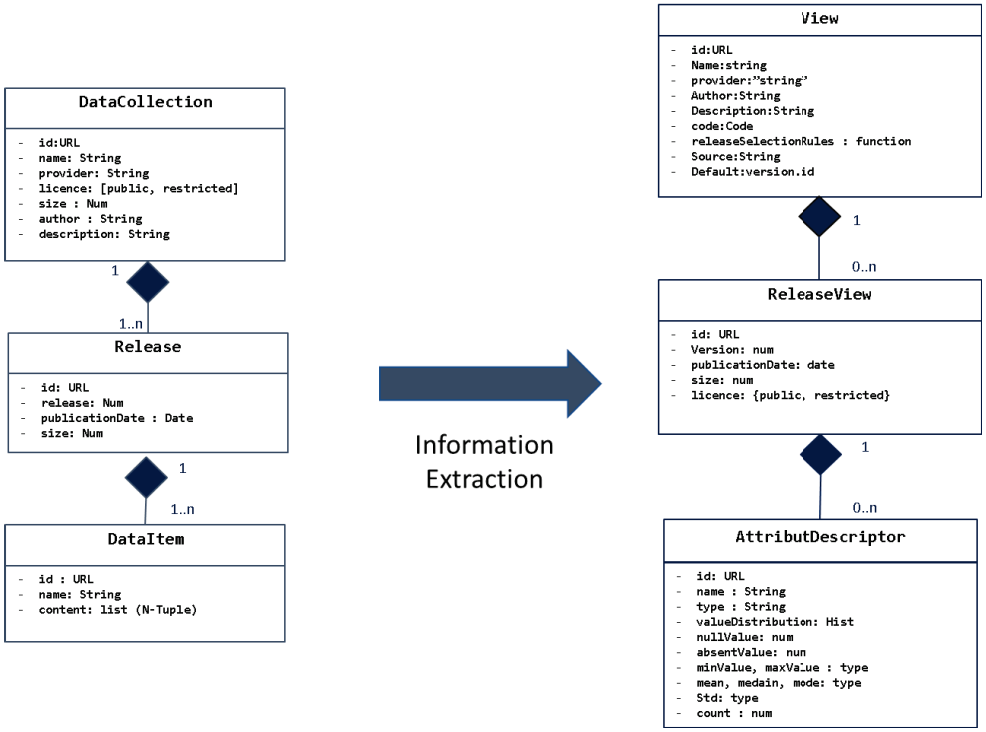

Figure 19: View collection

4.2 PRELIMINARIES: DATA TYPES

To describe our data `view` model, we will use an object-oriented model to define concepts as classes. Data types are the fundamental building blocks of data. Whilst at the lowest level near the metal it is represented as a series of 1 and 0, most programming languages provide higher abstraction to manage data and, in particular, data types which can be used to model pieces of data.

We use atomic and complex data types to define the concepts of our model and we represent them as classes.

4.2.1 ATOMIC AND COMPLEX DATA TYPES

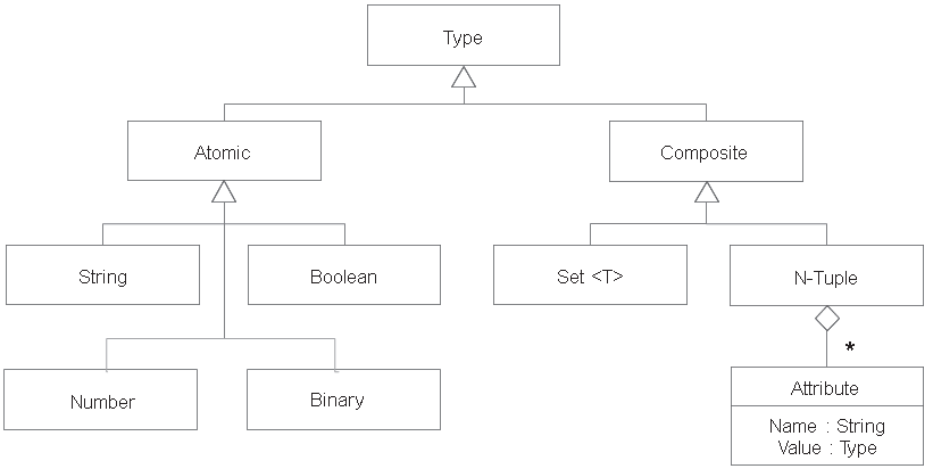

Figure 20: Basic Data Types

There are two families of data types, atomic and composite (Figure 20):

Atomic types represent instances that cannot be further divided in other values. The *atomic types* considered in the model are String, Number and Boolean. The type *String* represents any *chain of characters* (e.g., “www”, “James Bond”, “pwd123”). The type *Number* represents any *real* or *natural* numbers (e.g., -1, 0, 1, 3.1416). The type *Boolean* represents values *true* and *false* (i.e., *truth* values).

Composite types represent instances that are composed of other values (i.e. instances of other types). The *composite types* considered in the model are the types Set, N-Tuple and Attribute. These types conform to the following rules:

- If $A_1 \ldots A_n$ are different *attribute names* (i.e., $A_i \neq A_j, \forall_{i,j} \in [1\ldots n]$), and $T_1 \ldots T_n$ are *type names*, the expression $<A_1: T_1, \ldots, A_n : T_n>$ represents a type N-Tuple. For example, the expression:

< *name*: String, *value*: Type >

represents the type Variable as a binary tuple composed of (i) a *name* attribute of type *String* representing the *variable' name* and (ii) a *value* attribute (of any type) representing the *variable value*.

- If T is a *type name* then the expression Set <T> represents a *typed collection* where all the elements in the collections are of type T. For example, *Set <String>* represents a collection containing only elements of type *String*.

To further develop this model, we need to look at what produces and manipulate the data namely functions and relations. The following section defines function types that can be used for defining functions adapted to manipulate data collections according to their characteristics.

4.2.2 FUNCTION TYPES

Functions are modelled using an N-tuple to represent the input and output data, the logic that implements it (Listing 1).

```
Function: < name: string,
  input: Set[Attribute],
      output: Tuple,
       code: Binary>
```

Listing 1: function type

A `function` is modelled as a tuple with four attributes:

- *name* represents the name of the function.
- *input* represents the set of variables that a function receives.
- *output* represents the *type* of the *value* produced by the execution of the function.

– *code* represents the binary code that implements the function.

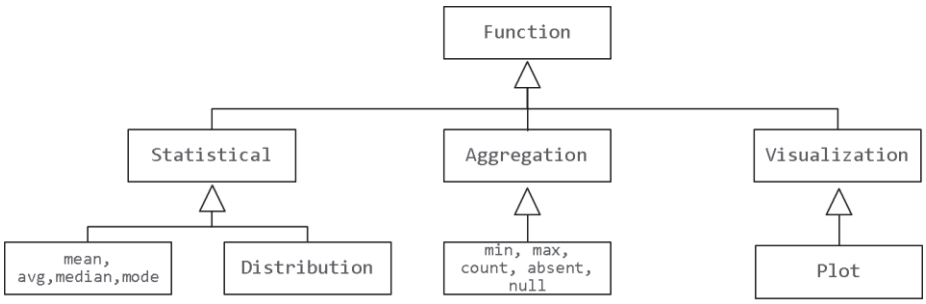

Figure 21: Function Type

We specialize the type `Function` into 3 function types representing statistical, aggregation and visualisation operations (Figure 21):

– `Statistical`: this type represents functions used to compute statistical values and value distribution of an attribute extension.
– `Aggregation`: this type represents functions used to aggregate information for a set of attributes, i.e. get information linked to the data aggregate.
– `Visualisation`: represents functions used to visualize data according to given graphic metaphors (e.g., histograms, bubble charts, etc).

4.2.3 RELATION TYPES

The type relation represents a functional or semantic dependency between two attributes. The type `Relation` is modelled as a N-tuple (Listing 2) with “id”, has a set of “input” attribute which can be put into the “code” to associate each value of the attribute to the value of the “output” attribute.

```
Relation: <    id: string,
   type: relationType,
   input: Set[values],
```

```
                output: Set[Values],
                code: Binary>
```

Listing 2: relation type

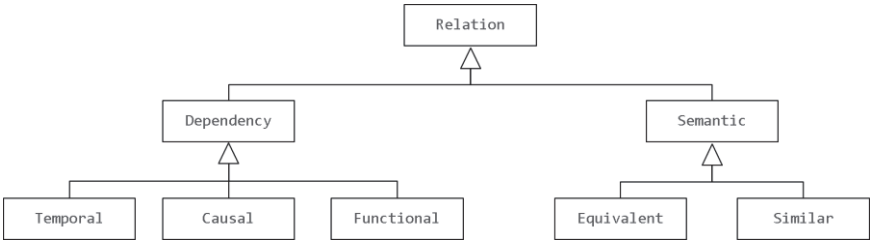

Figure 22: Relation Type

We have identified two families of relation types (Figure 22):

- `Dependency` relations modelling functional, temporal and casual dependencies between two attributes.
- `Semantic` relations defining similarity, equivalence relations between two attributes.

4.3 VIEW MODEL

Figure 23 shows the UML diagram of our `view` model that defines eight classes based on the data types defined in the previous section. The model defines two key types represented respectively by the class

- `dataCollection` that models a data collection consisting in releases produced at a given time.
- `View` models statistical and relational meta-data of the releases of a data collection.

In this section, we are first going to look at the classes involved in defining a `dataCollection`, then we will look at the classes involved in the definition of a *`view`*.

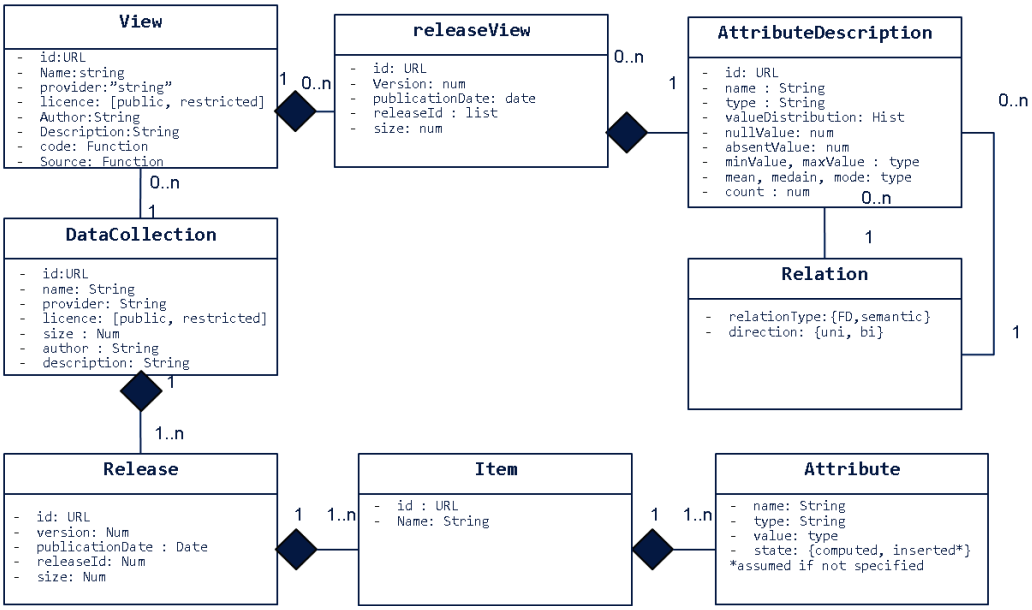

Figure 23: Our curation model

4.3.1 DATA COLLECTION

The class `dataCollection` models a data collection organized into releases together with the context in which it is produced (Figure 24). The class `Release` models the content of a data collection consisting in data items produced at a production time and having a given size. A data item is represented by the class `DataItem`.

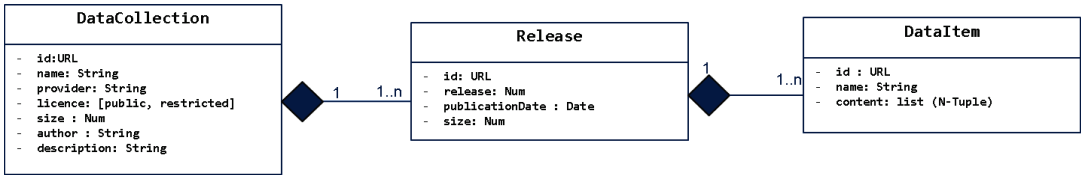

Figure 24: DataCollection Model

As shown in the UML diagram in Figure 24, the class `dataCollection` models a data collection as a composition of `releases` and by:

- "id" which uses the url to represent a unique value identifying the dataCollection;
- "name" represents the logic name used to identify a dataCollection;
- "owner" represents the name of the entity which collects and stores the dataCollection and thus defines the rules of term of use;
- "author" represents the name of the person who setup the collection and that has some expertise on producing the data;
- "description" represents a text giving the context to how, why and what data is contained in the collection;
- "licence" represents the licencing information associated to the dataCollection, for example, whether it contains open data or not;
- "size" represents the size of the collection in terms of number of items and bytes.

The class Release models a set of data belonging to a data collection produced at a specific time and sharing some characteristics. A release is described by:

- "id" represents a unique identifier corresponding to the access address described by the URL given by the dataCollection and the release number;
- "releaseNum" represents a unique number to the dataCollection corresponding the number of the release given by the service at the time of the creation of this release;
- "publicationDate" represents the date and time at which this data is put online;
- "size" represents the number of items in a release, note the size of the collection corresponds to the sum of the sizes of its release.

A *release* consists of *items*. An *item* is a unit of data comparable to tuple in relational databases, or document in a document-oriented database. The class dataItem defines an item with the following attributes:

- "id" represents a unique identifier specified by an URL combined with the release ID to which it belongs and the item name;
- "name" represents a logic name identifying a set of values (sensor group, event, person);
- "attributes" represents the set of attributes that describe the structure of an item.

The class AttributeDescriptor models an attribute that defines (with other attributes) the structure of an item as:

- "name" chosen by the owner often corresponding to the name of the sensor or the type of data being collected;
- "type" deduced from the value of the attribute, it can be an atomic or complex data type;
- "value" is the value of the attribute of type Type;
- "state" is the way the data was produced, i.e. inserted by a sensor or computed.

Producing these views requires the use of functions to generate and manage a dataCollection. We define such functions in the following sections.

4.3.1.1 CREATION OF A DATACOLLECTION

The function creation() of the class dataCollection **Erreur ! Source du renvoi introuvable.**is specified in Listing 3. The objective is to collect all the elements of a raw data set released at a given date/time and update the items describing the content of the data set. This is done in 4 steps:

(1) Collect meta-data from the data collection specified by the provider and assign them to an instance of the class dataCollection.

```
createDataCollection (url: URL, name: String, provider: String,
                      licence: {public, restricted}, size: Float,
                      author: String, description: String) → d1:DataColection where
```

```
        d1. url = url+name, d1. name = name,
        d1. provider = provider,
        d1.licence = licence in {public, restricted},
        d1. size = 0, d1. author = author,
        d1. description = description
```

Listing 3: Create a DataCollection

(2) Create a data `release` (Listing 4).

```
createRelease (d1: DataCollection, pDate: Date,rID:String) → r1: Release where

        r1.url = d1.url+rID, r1.releaseID = rID,
        r1.publicationDate = pDate, r1.size = 0,
        r1.dataItems = set<item>
```

Listing 4: Create a release

(3) Insert data into the `release` (Listing 4), update the size of the `release`.

```
addItem (i1: Item, r1: Release) → r1: Release where

        r1.size=r1.size+1
        r1.items += r1.items + i1
```

Listing 5: Add Item

(4) Insert the release into the `dataCollection` (Listing 5), updating the size of the collection.

```
addRelease (d1: DataCollection, r1: Release ) → d1: DataCollection where
        d1.size = d1.size + r1.size
        d1.releases += d1.releases + r1
```

Listing 6: Add Release

This allows the automated production of `dataCollections` providing all the data over time information. Now we are going to look at the other type of parent object `views`.

4.3.2 VIEW

The class View models a statistical description of a data collection its releases and their data items. As shown in Figure 25 the class View represents a data collection with:

- “id” presented as its access URL given by the service;
- “name” chosen by the owner should include the source and the method;
- “owner” the entity who collects and stores the data and defines the term of use policy;
- “author” person to contact on the view if the description is not sufficient, is generally a data analyst who setup the view;
- “description” generally written by the author, gives context to how, why and what data is produced;
- “source” corresponds to the dataCollection analysed by this view;
- “code” is the code given by the data analysts to analyses the data. This is important to track how the data has been modified and to both reuse the view but also to allow other analysts to understand the details of how the view was produced;
- “releaseSelectionRule” is a function that defines the release selected;
- “default” this attribute shows the “live” version of the view i.e. the version considered the most representative of the present day. It is the version which will be returned by default if no further information is provided. This is done to maintain and update the services using these views in their action.

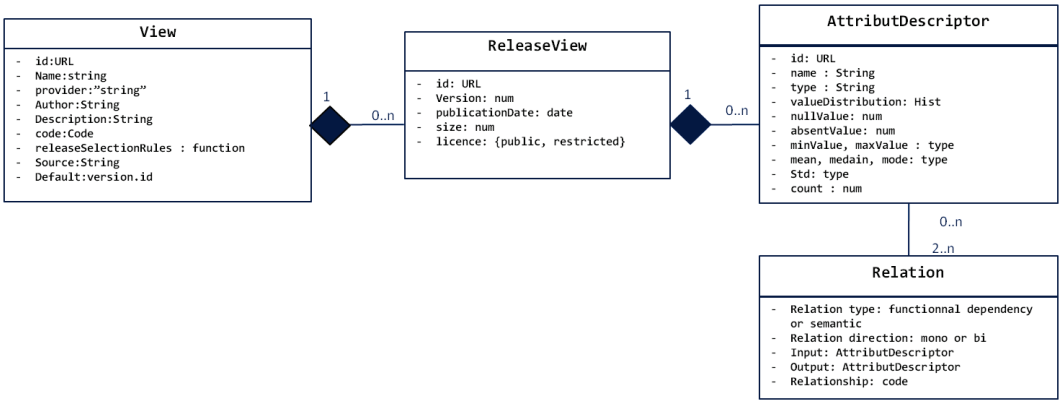

Figure 25: View Model

`View` are composed of `releaseViews`. A `releaseView` is the equivalent to the `release` in the collection model in that it provides infrastructure to track evolution of the `view` over time. The data stored in data `release` correspond to a group of `releases` from a collection. A `releaseView` is described by:

- "id" which corresponds to its access URL given by the service as a combination of the parent view URL and the version number;
- "VersionNum" the number of the `release` given by the service correspond to the number of `release` that existed at the time of creation;
- "publicationDate" date and time at which this data is put online extracted from the `item` time stamps;
- "size" number of `items` in a `release`, note the size of the collection correspond to the sum of the sizes of its `release`;
- "releaseIds" the list of `releases` analysed in the by this version of the `view`.

This has a more limited use than it counterpart as it is essentially a repository for `attributDescriptor`. I can, however, give interesting insights if the size attribute varies substantially between `releaseViews`.

`ReleaseViews` are composed of `attributDescriptor`. An `attributDescriptor` provides a statistical representation of an attribute produced by the code applied to the collection. The simplest form would be to do a statistical analysis of each attribute of a `release`. In this an `attributeDescriptor` corresponds will correspond to a statistical analysis an attribute of the `items` in the `release`. This makes a basic statistical presentation of the predicate with min, max, mean, median, mode, standard deviations, value distribution and consistency information with the number of null values and absent values. It is described by:

- "id" which corresponds to its access URL given by the service as a combination of the parent version URL and the `AttributeDescriptor` name and type;
- "name" given by the code processor used to analyse the data;
- "type" given by the code used to analyse the data;
- "valueDestibution": value distribution represents the number of time a value has been produced in the creation of the attributedescriptor;
- "nullValue" extracted from the data, it represents the number `items` for which the value of an attribute is known to not exist;
- "absentValue" extracted from the data, it represents the number `items` for which the attribute does not appear. An example of deference between absent and null is, in the case of null, attribute will still appear in the `item` with a value of "null" or "", absent correspond to the attribute not appearing in the `item`.

- “count” calculated from the histogram, is the number of non null and absent values;
- “minValue” calculated from the histogram, is the smallest value for this attribute;
- “maxValue” calculated from the histogram, is the largest value for this attribute;
- “mean” calculated from the histogram, is the average value for this attribute;
- “medain” calculated from the histogram, is the middle value of the attribute (count/2);
- “mode” calculated from the histogram, is the attribute with the largest number of time with the same value;
- Having all three mode, median and mean is useful to identify if an attribute respects a normal distribution;
- “std” calculated from histogram, is the average deviation from the average.

This is the document which gives the most information. The information provides condensed information on many of an attribute’s characteristics. Namely its distribution and reliability.

Finally, an important aspect of data in Big Data is in the relation there is between the data pieces, specifically attributes. `Relations` can be represented in two families, functional relations in which you can essentially from on attribute predict the value of another attribute, and semantic relations where despite of having different names the attribute talks about same topic. Thus, relations represent relations between attribute. It is described by:

- “Input”: corresponding to the input attribute;
- “output”: corresponding to the output attribute;
- “relationType” classifies the relation as either a semantic type, e.g. attribute has the same values, or functional dependency, e.g. correlation;
- “direction” classifies the relation as either unidirectional (no twin relation) or by directional (twin relation);

– “relationship” if possible gives the function that transforms one attribute into the other.

This maintains a structure by which data analysts are informed of existing usable relations for their application. The creation of these objects is an important aspect we will now investigate.

4.3.2.1 CREATING A VIEW

As seen in Section 4.2.2 there are many types of collections. The following lines describe the ones we considered in our work.

Computed by statistic functions performed on the content of a given release

Statistical data represents an aggregation of the data in a `view` into a smaller more informative piece of data. The creation of the `view` is very similar to the creation of a `dataCollection` but is done in 3 steps:

(1) We start of by assigning data collected from the data analyst and data collection analysed (Listing 7).

```
createView (   r1: Collection, version: Int,
             licence: {public, restricted}) → v1: View where
       v1.url = r1.url, v1.dataCollectionName = r1.name,
       v1.provider = r1.provider,
       v1.author = r1.author,
       v1.description = r1.description,
       v1.releasesViews =+ r1,
       v1.releasesViews =+ createReleaseView(r1, version, licence)
```

Listing 7: Create a View

(2) The `releaseview`’s are generated from each `release` in the data collection (Listing 8).

```
createReleaseView (r1: Set(Release), version: Int,
                   licence: String, v1:View) → rv1: ReleaseView where

       rv1.releaseId = r1.id, rv1.version = version,
       rv1.publicationDate = r1.publicationDate,
```

```
        rv1.size = 0,
        dataItems= Set(items),
        for each release in r1:
                dataItems = dataItems+release.dataItems
                rv1.releaseMeta-data = createReleaseMeta-data(dataItems,v1)
```

Listing 8: Create a Release View

(3) The value of each attribute in the `release` is collected and stored to extract the meta-data (Listing 9). Ounce this process is done, the `attributeDescriptor` is generated from the value stored previously. It is done by applying a series of functions used to extract the statistical information like averages and median and deduce information like `nullValues` and `absentValues`.

```
createReleaseMeta-data (dataItems: Set[DataItems], view: View)→
                        adesc1:Set[AttributeDescriptor] where
        y = 0;
        for each item in dataItems:
                x = 0;
                for each att in item.Content[x]:
                        attributes = attributes ++ item.Content[x];
                        x++;
        for each attr in attributes:
                adesc1[y].state = attr.state;
                adesc1[y].attribute = attr.att;
                adesc1[y].valueDistribution = Distribution(dataItems, attr);
                adesc1[y].minvalue = MinFunction(dataItems, attr);
                adesc1[y].maxvalue = MaxFunction(dataItems, attr);
                adesc1[y].mean = meanFunction(dataItems, attr);
                adesc1[y].modev = modeFunction(dataItems, attr);
                adesc1[y].count = countFunction(dataItems, attr); y++;
```

Listing 9: Create a release meta-data

Deduced through data analytics processes

Deduced data is found by cross referencing data from multiple sources. The simplest form is the search for null and absent values (Listing 10).

```
discoverNullAbsent(rv1: ReleaseView, dataItems: Set[DataItem]) →
                    adesc1:AttributeDescriptor where
```

```
	for each item in dataItems:
		for each attr in item:
			adesc1.nullValue = discoverNullValue(dataItems, attr)
			adesc1.absentValue = discoverAbsenceValue(dataItems, attr)
```

Listing 10: Discover Null and Absent values

The search for null data is a simple matter of keeping track of the attributes valued as null in an `item` (Listing 11).

```
DiscoverNull(dataItems: Set[DataItems], attr:Attribut) → ditem1 : int where

		for each item in dataItems:
		  if(item[attr]==null):
			ditem=ditem+1
```

Listing 11: Discover Null values

Absent values correspond to attributes not appearing in some `items`. Thus, to work out the number of absent value you need to know the size of the `release` and the amount of data collected for that attribute (Listing 12).

```
DiscoverAbsent(dataItems: Set[DataItems],attr:Attribut) → ditem1 : dataItem where
	count=0
	tot=0
	for each item in dataItems:
		if exist(item[attr]):
			count=count+1
		tot=tot+1
	ditem=tot-count
```

Listing 12: Discover Absent values

Distribution function converts a list of values into a table tracking the number of time a specific value has appeared (Listing 13**Erreur ! Source du renvoi introuvable.**)

```
DistributionFunction (dataItems:Set[DataItem] ,attr:attribute)→
			valueDistribution:Set[<value:Float>]

	valueDistribution = computeDistributionFunction(dataItems)
```

Listing 13: Function distribution

Relations represent a common theme in between attributes either through a capacity to predict an attribute from another, or through a common topic (Listing 14**Erreur ! Source du renvoi introuvable.**).

```
computeAttributeRelations(rv1: ReleaseView,rv2: ReleaseView)→
                        rela1:Set[Relation] where

       rela1 = Set[relation]
       for each attr1 in rv1.attributs:
              for each attr2 in rv2.attribut:
                     relation = findRelation(attr1, attr2)
                     if(relation !=null):
                            rela1.append(relations)
```

Listing 14: Compute Attribute Relation

Views would be exploited in the decision making **Erreur ! Source du renvoi introuvable.**process by data scientist to answer their questions relative to the exploitation of their data. The method would go produce a new version of a view from a group of data releases, use the information in the version to make a decision, test that decision, if it succeeds decision made else go back to the version.

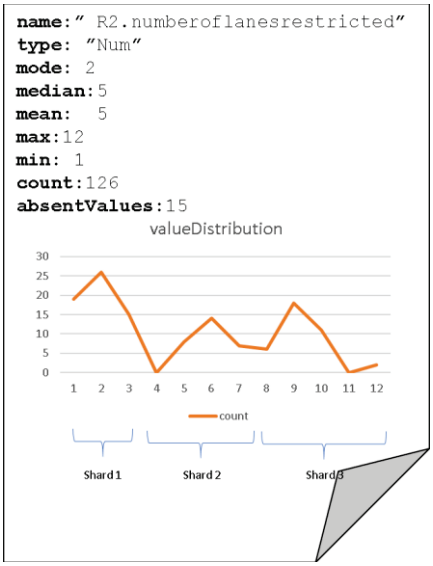

Figure 26 : Sharding example

So, in the scenario presented in section 3.1 the user wishes to shard the data set, the data analyst would start of by producing the collection release model, then produce a `view` model. Based on the `attributeDescriptors`, the data analyst can understand how the data is distributed across values the attributes of the collection.

4.4 MANIPULATING VIEWS

Exploiting `view` turns around comparing a combining element of same type. Operations applied between `releaseViews` produce new `releaseView` as shown in

Table 5.

Table 5: view manipulation operator

Operation	Input/Output	Description	Symbol
similarity	$a_1 \sim a_2 = a_3$	Computes the similarity between two `attributeDescriptors`	~
union	$a_1 \cup a_2 = a_3$	Creates an `attributeDescriptor` corresponding to the union of the values of two other releases.	U
intersection	$a_1 \cap a_2 = a_3$	Creates an `attributeDescriptor` corresponding to the intersection of the values of two other releases.	∩
difference	$a_1 \Delta a_2 = a_3$	Creates an `attributeDescriptor` corresponding to the	Δ

		difference of the values of two other releases.	
product	$r_1 * r_2 = r_3$	Create a new release with a set of items corresponding to the fusion of items between the two releases.	*

4.4.1 SIMILARITY

The function `similarity()` (Listing 15**Erreur ! Source du renvoi introuvable.**) generates an `attributeDescriptor` which expresses the relative similarity between two `attributeDescriptor`, i.e. difference over the range of difference. Given a_1 and a_2 two attributes of type `attributeDescriptors` and two `releases` r_1 and r_2 of type `Release` where:

- a_1<id_{11}, $type_{12}$, $mean_{13}$, $median_{14}$, $mode_{15}$, max_{16}, min_{17}, $count_{18}$, $absentvalues_{19}$, $nullValues_{110}$, $valueDistribution_{111}$>,
- a_2<id_{21}, $type_{22}$, $mean_{23}$, $median_{24}$, $mode_{25}$, max_{26}, min_{27}, $count_{28}$, $absentvalues_{29}$, $nullValues_{210}$, $valueDistribution_{211}$>.

$$a_1 \sim a_2 = a_3$$

Where a_3<id_{31}, $type_{32}$, $mean_{33}$, $median_{34}$, $mode_{35}$, max_{36}, min_{37}, $count_{38}$, $absentvalues_{39}$, $nullValues_{310}$, $valueDistribution_{311}$ > has values for -1 to 1 representing the relative similarity between a_1 and a_2:

$$a_3.attr = (a_1.attr - a_2.attr)/(a_1.attr + a_2.attr)$$

The following code implements the operation.

```
sim(elem1:numeric,elem2:numeric)→ sim:float
```

```
       sim=(elem1-elem2)/(elem1+elem2)

computeSimilarity (attrdesc1:attributeDescriptors ,
                   attrdesc2:attributeDescriptors )
                   → attrSim : attributeDescriptors

       attrSim.state = calculated
       for each elem in attrdesc1.valueDistribution :
              attrSim.valueDistribution[elem] =
       sim(attrdesc1.valueDistribution[elem],attrdesc.valueDistribution[elem] );
              attrSim.minvalue = sim(attrdesc2.minvalue, attrdesc1.minvalue)
              attrSim.maxvalue = sim(attrdesc2.maxvalue, attrdesc1.maxvalue)
              attrSim.mean = sim(attrdesc2.mean, attrdesc1.mean)
              attrSim.modev = sim(attrdesc2.modev, attrdesc1.modev)
              attrSim.count = sim(attrdesc2.count, attrdesc1.count)
```

Listing 15: Compute Similarity

4.4.2 UNION

The function `union` (Listing 16) generates an `attributeDescriptor` corresponding to the set theory definition of union of the values between two `attributeDescriptor`. Given a_1 and a_2 two attributes of type `attributeDescriptors` and two `releases` r_1 and r_2 of type `Release` where:

- a_1<id_{11}, $type_{12}$, $mean_{13}$, $median_{14}$, $mode_{15}$, max_{16}, min_{17}, $count_{18}$, $absentvalues_{19}$, $nullValues_{110}$, $valueDistribution_{111}$>,
- a_2<id_{21}, $type_{22}$, $mean_{23}$, $median_{24}$, $mode_{25}$, max_{26}, min_{27}, $count_{28}$, $absentvalues_{29}$, $nullValues_{210}$, $valueDistribution_{211}$>.

$$a_1 \cup a_2 = a_3$$

With a_3<id_{31}, $type_{32}$, $mean_{33}$, $median_{34}$, $mode_{35}$, max_{36}, min_{37}, $count_{38}$, $absentvalues_{39}$, $nullValues_{310}$, $valueDistribution_{311}$ > an `attributeDescriptors` of the union of the of a_1 and a_2 generated as if the `attributeDescriptor` was generated from a `release` r3 built from the union of two `release` r_1 and r_2, i.e. grouping all the item of two releases into one.

First, we combine the valuesDistributions of a_1 and a_2 by adding the counts for every values in both `attributeDescriptor` then recalculating the statistical values.

```
computeUnion (attrdesc1:attributeDescriptors , attrdesc2:attributeDescriptors )
              → attrUnion : attributeDescriptors

       if(attrdesc1.relation.output == attrdesc2 and
         attrdesc1.relation.type == ”semantic”):

              attrUnion.state = “computed”;
              attrUnion.valueDistribution =+ attrdesc1.valueDistribution
              attrUnion.valueDistribution =+ attrdesc2.valueDistribution
              attrUnion.minvalue = MinFunction(attrUnion.valueDistribution, atr);
              attrUnion.maxvalue = MaxFunction(attrUnion.valueDistribution, atr);
              attrUnion.mean = meanFunction(attrUnion.valueDistribution, atr);
              attrUnion.mode = modeFunction(attrUnion.valueDistribution, atr);
              attrUnion.count = countFunction(attrUnion.valueDistribution, atr);
              attrUnion.missing = attrdesc1.missing + attrdesc2.missing
attrUnion.missing = attrdesc1.null + attrdesc2.null
```

Listing 16: Compute the Union

4.4.3 INTERSECTION

Intersection function (Listing 17**Erreur ! Source du renvoi introuvable.**) generates an `attributeDescriptor` corresponding to the Intersection of the values between two `attributeDescriptor`. Given a_1 and a_2 two attributes of type `attributeDescriptors` and two `releases` r_1 and r_2 of type `Release` where:

- a_1<id_{11}, $type_{12}$, $mean_{13}$, $median_{14}$, $mode_{15}$, max_{16}, min_{17}, $count_{18}$, $absentvalues_{19}$, $nullValues_{110}$, $valueDistribution_{111}$>,
- a_2<id_{21}, $type_{22}$, $mean_{23}$, $median_{24}$, $mode_{25}$, max_{26}, min_{27}, $count_{28}$, $absentvalues_{29}$, $nullValues_{210}$, $valueDistribution_{211}$>.

$$a_1 \cap a_2 = a_3$$

With a_3<id_{31}, $type_{32}$, $mean_{33}$, $median_{34}$, $mode_{35}$, max_{36}, min_{37}, $count_{38}$, $absentvalues_{39}$, $nullValues_{310}$, $valueDistribution_{311}$ > an `attributeDescriptors` of the intersection of a_1 and a_2 generated as if a_3 was generated from a `release` r_3 built from the intersection of two `release` r_1 and r_2 along a specific attribute, i.e. by creating a release with the common elements of two other releases. This is done by computing a valueDistribution for a_3 such as it takes the values of a_1 and a_2 with the lowest count then recalculating the statistical values from this new `valueDistribution`.

```
computeIntersection(attrdesc1:attributeDescriptors ,
                    attrdesc2:attributeDescriptors )
                                    → attrIntersection : attributeDescriptors
       for elem in attrdesc1.valueDistribution:
              attrIntersection[elem]= min(attrdesc1.valueDistribution[elem],
                                          attrdesc2.valueDistribution[elem])
              attrDifference = computeStats(attrIntersection)
```

Listing 17: Compute the Intersection

4.4.4 DIFFERENCE

Difference function (Listing 18**Erreur ! Source du renvoi introuvable.**) generates an `attributeDescriptor` corresponding to the set theory definition of difference of the values between two `attributeDescriptor`. Given a_1 and a_2 two attributes of type `attributeDescriptors` and two `releases` r_1 and r_2 of type `Release` where:

- a_1<id_{11}, $type_{12}$, $mean_{13}$, $median_{14}$, $mode_{15}$, max_{16}, min_{17}, $count_{18}$, $absentvalues_{19}$, $nullValues_{110}$, $valueDistribution_{111}$>,
- a_2<id_{21}, $type_{22}$, $mean_{23}$, $median_{24}$, $mode_{25}$, max_{26}, min_{27}, $count_{28}$, $absentvalues_{29}$, $nullValues_{210}$, $valueDistribution_{211}$>.

```
a1 Δ a2 = a3
```

With a_3<id_{31}, $type_{32}$, $mean_{33}$, $median_{34}$, $mode_{35}$, max_{36}, min_{37}, $count_{38}$, $absentvalues_{39}$, $nullValues_{310}$, $valueDistribution_{311}$ > an `attributeDescriptors` of the diffrence of a_1 and a_2 generated as if a_3 was generated from a `release` r_3 built from the diffrence of two `release` r1 and r2 along a specific attribute, i.e. by creating a release with the noncommon elements of two other releases. This is done by computing a valueDistribution for a_3 such as it take the absolute value of the count of a_1 minus a_2, for every value their valueDistribution then recalculating the statistical values from this new valueDistribution.

```
computeDifference(attrdesc1:attributeDescriptors , attrdesc2:attributeDescriptors )
                  → attrDifference : attributeDescriptors

       for elem in attrdesc1.valueDistribution:
                     attrDifference[elem]=abs( attrdesc1.valueDistribution[elem] –
                                               attrdesc2.valueDistribution[elem])
       attrDifference = computeStats(attrDifference)
```

Listing 18: Compute the Difference

4.4.5 PRODUCT

The production function (Listing 19) produces a new `release` by combining `items` with related attribute together and storing it into a new `release`. Given two releases r_1 and r_2:

- r_1<id, releaseNum, publicationDate, dataitems>
- r_2 <id, releaseNum, publicationDate, dataitems>

$$r_1 * r_2 = r_3$$

With r_3 <id, releaseNum, publicationDate, dataitems> being of type `Release` where r_3`.dataitems` is a set corresponding to the fusion of two r_1`.dataitems` and r_2`.dataitems` such that each element of the set r_1`.dataitems` is combined with all the elements of the set r_2`.dataitems` into a single item:

r_3.item[l] = [r_1.dataitems [n], r_2.dataitems [m]] such as

r1.dataitems[n] and r_2.dataItems[m] are found like this:

a_1.relation[‘$a_1.a_2$’].code(r_1.item[n])=r_2.item[m]

The following shows the implementation of the Cartesian product between to data items sets.

```
computeProduct (attrdesc1:attributeDescriptors,
                attrdesc2:attributeDescriptors,
                R1: Release, R2: Release) → RVJ : Release

       if(attrdesc1.relation.type==”functional”) and
         attrdesc1.relation.output=attrdesc2):

                      for (elem1 in R1):

                             for (elem2 in R2):

                                    if(elem1[attrdesc1.name]==
                                   attrdesc1.relation.listing(attrdesc2))
                                           elem = elem1+elem2
                                           RJV += elm
```

Listing 19: Compute the Cartesian Product

4.5 MAINTAINING VIEWS

The structure of collection and the associated `view` is fairly complexes but crucial in providing information on the data. Whilst on large `attributeDescriptors` the addition or subtraction of an `item` would produce only minor changes to the information, this cannot be said for smaller `attributeDescriptor` and the cumuli of minor changes, if not accounted for, risks producing disinformation. Thus, we need operations (Table 6) designed to manipulating `items` in `release` and updating the corresponding `view`.

Table 6: Modifying views operators

Operation	Input/Output	Description

insert	release, item, releaseView	Adds an item to a release and updates the associated releaseView.
modify	release, item, releaseView	replaces an item in a release and updates the associated releaseView.
delete	release, itemID, releaseView	Removes an item from a release and updates the associated releaseView.

4.5.1 INSERT

The function insertItem(release:R1, item:i1, releaseView:RV1) (Listing 20) is used to insert and update both the collection model and the view model. Given a Release R1 and item I1. The insert operator will add the item to the release and update the corresponding releaseViews RV1.

```
insertItem(release:R1, item:i1, releaseView:RV1): → release:R1, releaseView:RV1
       R =+ i1
       for( attr in i1):
              if(attr==null):
                     RV1.attr[attr].null++
              else:
              RV1.attr[attr] = insertToAttributDescriptor( i1[attr],RV1.attr[attr])

       for(attr in RV1 not in i1):
               RV1.attr[attr].absent++

insertToAttributDescriptor(attribut:A1, attributDescriptor:AD1)
                           → attributDescriptor:AD1

       AD1.valueDistribution[A1] ++
       AD1.mean = (AD1.mean* AD1.count+A1)/(AD1.count + 1)
       AD1.count ++
       AD1.median = median(AD1.valueDistribution)
       AD1.mode = mode(AD1.mode, AD1.valueDistribution[A1])
       AD1.min = min(AD1.min, A1)
       AD1.max = max(AD1.max, A1)
```

Listing 20 : Operator Insert

4.5.2 MODIFY

The function `modifyItem(release:R1, item:i1, releaseView:RV1)` (Listing 21) is used to find and replace an `item` in a collection and update both the collection model and the `view` model. Given a `Release` R1 and an `item` I1, the function update() will replace the `item` in the `release` and update the corresponding `releaseViews` RV1.

```
modifyItem(release:R1, item:i1, releaseView:RV1): → release:R1, releaseView:RV1
       it= R.items[i1.id]
       R.items[i1.id] = i1
       for( attr in i1):
              if(attr==null):
                     RV1.attr[attr].null++
              else:
              RV1.attr[attr] = updateToAttributDescriptor( i1[attr], it[attr],
                                                           RV1.attr[attr])
       for(attr in it not in i1):
               RV1.attr[attr].absent++

updateToAttributDescriptor(attribut:A1,attribut:AT,
                           attributDescriptor:AD1) → attributDescriptor:AD1

       AD1.valueDistribution[A1] ++
       AD1.valueDistribution[At] --
       AD1.mean = (AD1.mean*count+(A1-AT))/(count)
       AD1.median = median(AD1.valueDistribution)
       AD1.mode = mode(AD1.mode, AD1.valueDistribution[A1])
       AD1.min = min(AD1.valueDistribution)
       AD1.max = max(AD1.valueDistribution)
```

Listing 21: Operator Modify

4.5.3 DELETE

The function `deleteItem(release:R1, item.id:ID1, releaseView:RV1)` (Listing 22) is used to find and delete an `item` in a collection and update both the collection model and the `view` model. Given a `Release` R1 and an `item` id ID1, the function `deleteItem(release:R1, item.id:ID1, releaseView:RV1)` will remove the `item` from the `release` and update the corresponding `releaseViews` RV1.

```
deleteItem(release:R1, item.id:ID1,
           releaseView:RV1): → release:R1, releaseView:RV1
       it = R.items[ID1]
       R.items[ID1]=-
       for( attr in it):
              if(attr==null):
                     RV1.attr[attr].null++
              else:
             RV1.attr[attr] = deleteToAttributDescriptor( it[attr],
                                                          RV1.attr[attr])
       for(attr in it not in i1):
               RV1.attr[attr].absent++

deleteToAttributDescriptor(attribut:AT,
                           attributDescriptor:AD1)→ attributDescriptor:AD1
       AD1.valueDistribution[AT] --
       AD1.mean = (AD1.mean*count-AT)/(count-1)
       AD1.count--
       AD1.median = median(AD1.valueDistribution)
       AD1.mode = mode(AD1.mode, AD1.valueDistribution[A1])
       AD1.min = min(AD1.valueDistribution)
       AD1.max = max(AD1.valueDistribution)
```

Listing 22: Operator Delete

4.6 DISCUSSION AND FINAL REMARKS

Since meta-data is the backbone of the curation process, we propose a model to assist data analysts and managers in their decision making. Fundamentally data curation intervenes after the data analysis step, as a form of data visualisation for IT professional to explore in their decision support tools. Whilst this is technically true for CURARE as well, our model intervenes at every step of Big Data analytics as a tool assisting data analysts in the decision involved at each step.

In the data collection and cleaning phase, data analysts have to answer questions related to storage strategies and data pre-processing. CURARE assists by providing the range, distribution and variability of the data. This would be used to get clues on the characteristics of the data continuity useful for data pre-processing for example. The data distribution would assist the data analysts in finding optimal data storing strategies.

CURARE is a data curation model designed for data analyst with inbuilt operations comparable to those in database management systems allow to manipulate update and aggregate data from multiple collections.

5 IMPLEMENTING THE VIEWS MODEL AND EXPERIMENTING CURARE

In this chapter we look at the implementation of the data curation, the experiments conducted to evaluate the cost of generating `views` and a decision-making use case for validating the use of `views`. Accordingly, the chapter is organized as follows. First, section 5.1 introduces the implementation of the collection `view` model on top of a document-oriented data model. Then Section 5.2 looks into the manipulation of `views`. It describes how to update and manipulate them. Section 5.2.2 introduces the experiments we conducted for estimating the cost of generating `views` from raw data collections with different characteristics. It describes how `views` can be used for making decisions on how to shard data collections across different stores. Finally, Section 5.4 concludes the chapter and discusses lessons learned.

5.1 IMPLEMENTATION OF THE VIEW MODEL

We choose to build CURARE around the MongoDB [44] database, a document-oriented database. We use its service manager and the Webpy[45] framework that provides a user interface. We focused our implementation on the lower levels of the architecture in particular the information extraction and aggregation and integration layers of CURARE.

[44] https://www.MongoDB .com
[45] http://webpy.org

From technical perspective, MongoDB manipulates data in structures known as `collection` and `document`. A `collection` is a named entity which is a list of documents that serves a persistence root. `Document's` are stored as binary versions of a JSON [46] data format (JavaScript Object Notation) thus uses a JavaScript based query language known as BSON[47], it uses the attribute "`_id`" to identify the document. Thus, the logic for manipulating data in MongoDB was written using JavaScript.

As specified in the previous chapter, `dataCollections` in our model are data structures supporting 3 tiers of document (`dataCollection`, releases and items) contained in each other. The first step in development was adapting a 3-tier data (Figure 27) structure to a document database.

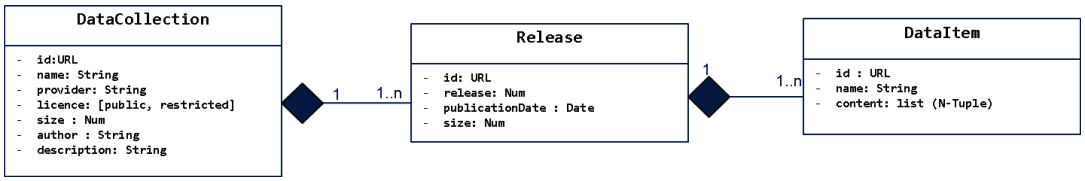

Figure 27: DataCollection Model

[46] https://www.json.org
[47] http://bsonspec.org

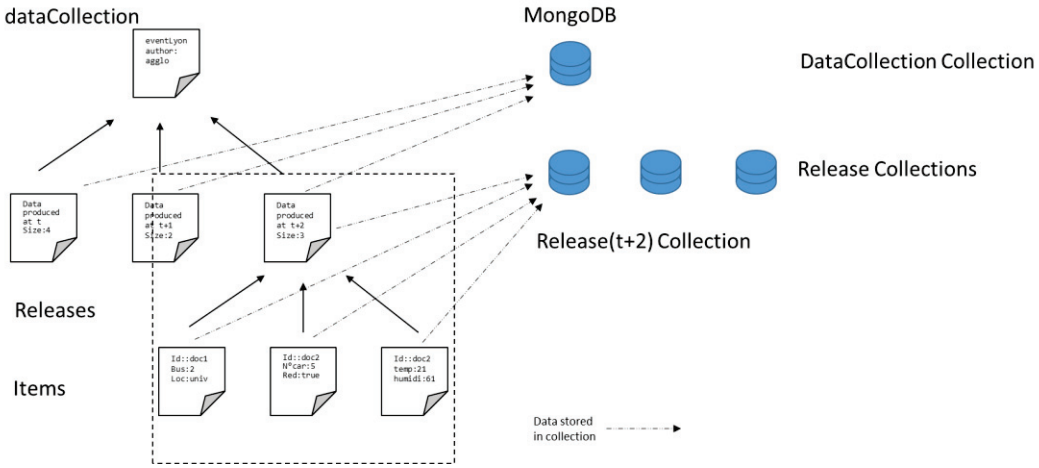

Figure 28: dataCollection to MongoDB

2 types of document in the model we propose, documents containing other documents, namely `dataCollections` and R
-alone documents, namely `Items`. These entities have to be modelled considering that the Mongodb data model provides
llection and Document. For modelling a document containing other documents in the case of the Mongodb data model, w
hen we create a Mongodb collection containing the parent and children documents. Thus, for the implementation, we have
collections (Figure 28): a unique collection `dataCollection`, which contain the `dataCollection` document and all the
ents; and a number of `release` collections. These `release` collections are named after the _id of the `release`. Each `re`
on stores the `release` document and the associated `item` documents. To associate the `release` to the `dataCollecti`
`llection` will have an attribute storing the list of _id's of all it `releases`, rather than the actual documents. Similarly, a d

Release has an attribute storing list _id's of Item documents. Similarly, views are implemented by 3 tiers (

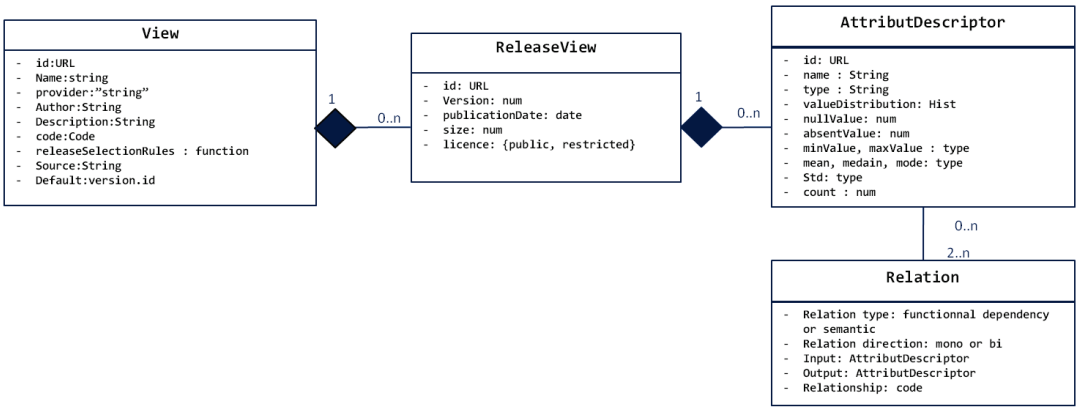

Figure 29) of document collections (views, releaseViews and attributeDescriptors). The model is implemented in the MongoDB data model as follows (Figure 30):

- a unique collection view, which contain the view document and all the releaseView documents; and a number of releaseView collections. These releaseView collections are named after the _id of the releaseView. Each releaseView collection store the releaseView document and the associated attributDescriptor documents. To associate the releaseView to the view, the view will have an attribute storing the list of _id's of all it releaseViews, rather than the actual documents. Similarly, a document releaseView has an attribute storing list _id's of attributDescriptor documents.

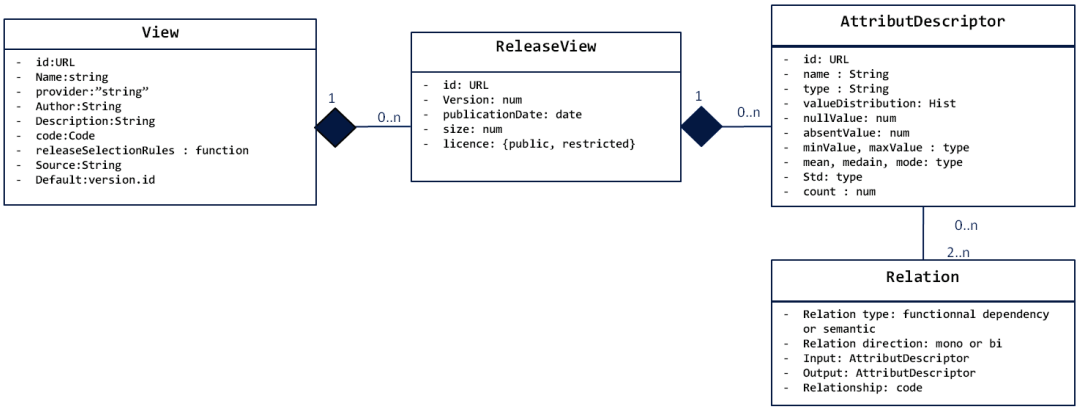

Figure 29: View Model

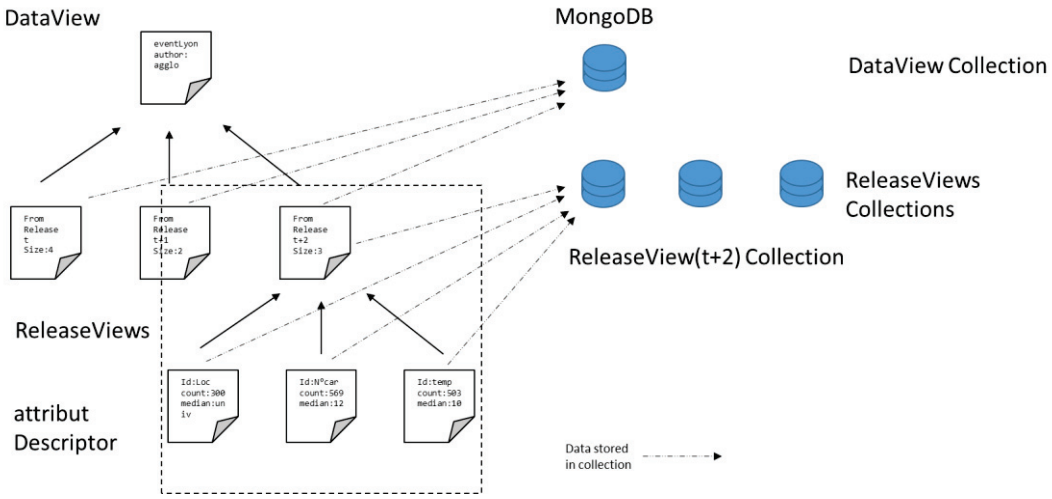

Figure 30:viewCollection to MongoDB

5.1.1 CREATING A DATA COLLECTION

To create a `dataCollection` from ready available datasets we identify meta-data in order to create a MongoDB data collection of type `dataCollection` and an associated `release`. We use 4 functions (Figure 31) coordinated by a general function that builds a new `dataCollection` out of a raw dataset:

(1)`createCollection(url, name, provider, licence, author: string, description)` creates a `dataCollection` and a document `dataCollection` and inserts it to the collection.

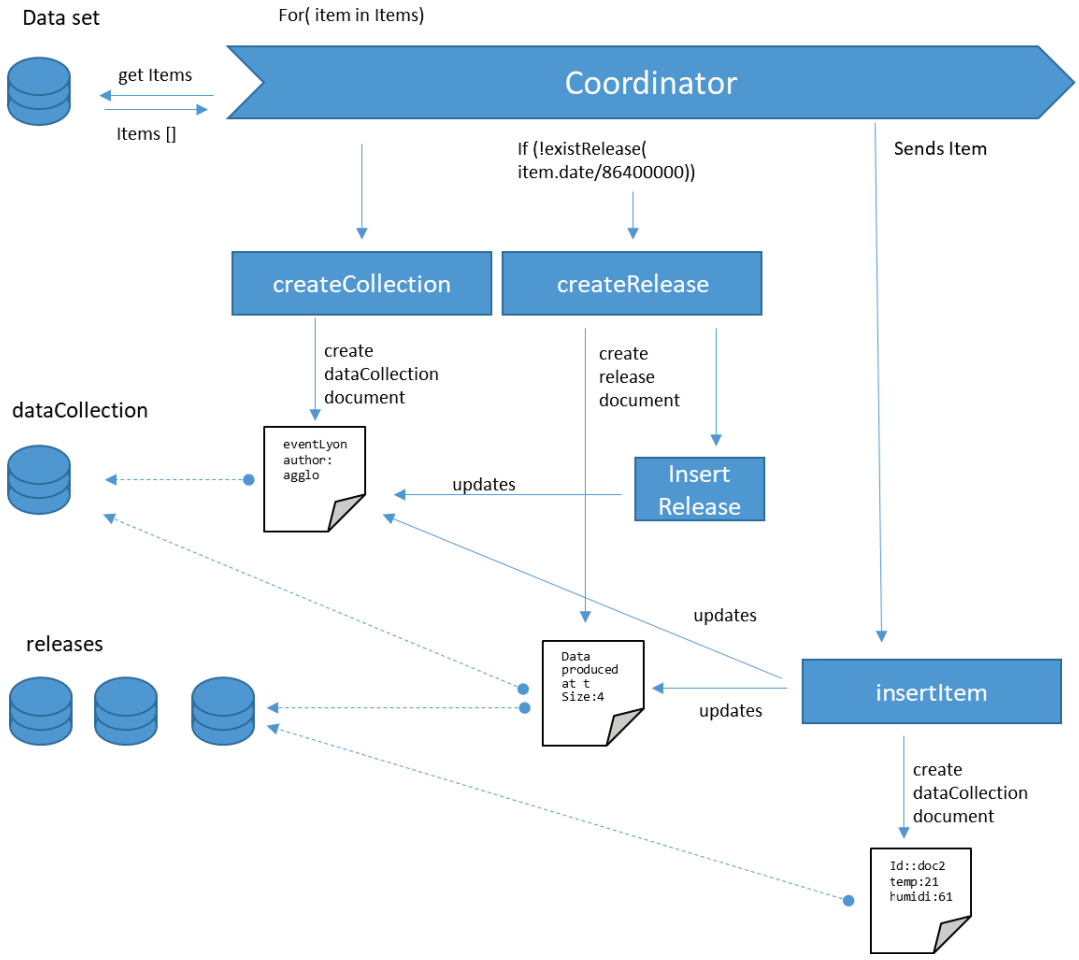

Figure 31: Creating a dataCollection

(2)createRelease(collection, id, date)creates a document of type release.

(3)`insertRelease(release, collection)` inserts a document of type `release` into the corresponding `dataCollection` mongo collection and the corresponding `release`

mongo collection. It then adds the `release _id` to the `dataCollection` release attribute.

(4) `insertData(item,release, collection, name)` inserts each data items into its corresponding release mongo collection, update the release document item list and size in both the collection `dataCollection` and the release data collection, and update the size of the collection document.

The coordinator function collects (fetches) and analyses each document from the initial data set and extracts the attributes date and time and converts them in epoch time. The release number is computing by dividing the date by 86400000, a.k.a 24*60*60*1000 on the number of milliseconds in a day to produce an integer used as the release number. If the release already exists it uses the function `insertData(item, release, collection, n:name)` Otherwise the function `createRelease(collection, id, date)` creates a new release and inserts its `id` to the corresponding collection.

5.1.2 CREATING A VIEW

Figure 32 shows the workflow implemented for creating a view. Given a `dataCollection` this operation will create the `view` and all the associated `releaseViews` and `attributDescriptors` using the following functions:

(1) createView(url, name, provider, licence, author, description, code, source),builds a MongoDB collection for the `view` and a `view` document with the meta-data describing the `view`.

(2) createReleaseView(collection, number, date), builds a mongo collection for a `releaseView`, creates a `releaseView`document.

(3) `insertReleaseView(Release, View)`, inserts the `releaseView` into the `view` MongoDB collection and the release MongoDB collection, then adds the `releaseView` _id to the `view` `releaseViews` attribute.

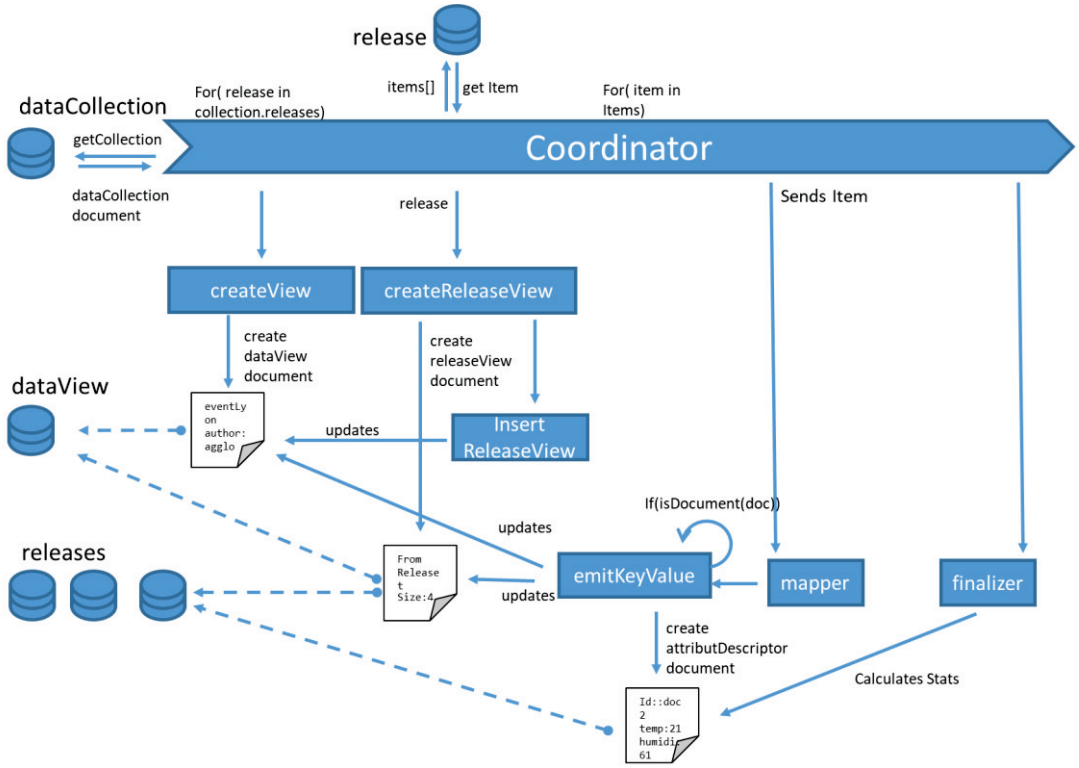

Figure 32: view creation

(4) `mapper(Item,ReleaseView)` creates and distributes the attributes to their corresponding `attributeDescriptor`. Depending on the structure of the input the process can be more or less complex. Processing the input item is done using the function `emitKeyValue(document, route).` Given an object as input, the item and the name of the `releaseView`, scan through each attribute, if this attribute is an object then it executes “emitkeyvalue”. If it is not an object then we identify the type of the data and add the type it to the end of the name, if the `attributDescriptor` does not exist we

create the attribute descriptor, update the array of `attributDescriptor` and size in the `releaseView` document in both the `view` data collection and the `releaseVeiw` data collection. Then finally we add the value to a list of values.

`emitKeyValue(document, route)` this function is designed to identify all the attributes of a document, even the embed attributes. To do this, we get all the attributes of a document and identify the type of each attribute. The attributes of document type are run through emitKeyValue and so one till there is no document attribute left. The other attributes are inserted into an `attributDescriptor` document with the same name and type of this attribute. To track the full name of embedded attributes `emitKeyValue` has a route variable which will take the name of the parent attribute and the name of the daughter attribute separated by a dot. The coordinator function that implements the workflow illustrated in Figure 31, First it executes the function `createView()` then it fetches the collection document to get the names of the releases. Then one by one, it collects the items from each release and runs them through the mapper. Once this task has been accomplished, it get the names of the `releaseView` collections and runs each `attributeDescriptor` through the `finalizer(ad:attributdescriptor).`

The mapper is by far the slowest process in the `view` creation. This can be accelerated implementing the creation of `views` under a MapReduce programming model to parallelise the process. In this case, the map function implements uses the same logic as the previous the mapper but send the key/value to the reducer rather than to a `releaseView` collection. The key is the name of the attribute and the value is the value of the associated attribute. The reducer function concatenates all the values from an attribute into a list of List. The finalizer will then calculate all the statistics in the same way the previous finalizer did.

5.2 MANIPULATING VIEWS

Data can be modified, added or removed from the releases of data collections. `Views` must be updated to reflect the changes in the data collection. We have defined the functions insert, remove, replace for manipulating the items of the `views`. The following sections define such functions.

5.2.1 INSERTING, REMOVING AND REPLACING AN ITEM

Given an Item and a collection the operation `insertItem(collection, Item)` inserts a new item to a collection then updates corresponding `views` using the following functions:

(1) `insertItem(collection, Item)` inserts the item to the corresponding release and update the information in the release document in both the release collection and the dataCollection collection. This function then triggers the function `insertAttribute(attribute)`

(2) `insertAttribute(attribute)` given an item as input parameter it searches all the attributes item and adds them and updates the corresponding `attributDescriptor`. Once done this, we trigger the function finalizer to recalculate all the statistical values.

(3) `finalizer(attributDescriptor)` updates all the calculated values in the attribute descriptor.

The operation `removeItem(itemID, Release)` deletes an item using 3 functions:

(1) `removeItem(itemID, Release)` removes the item from a `release` and updates the information in both the corresponding `release collection` and `dataCollection`. This function then triggers in cascade the function `removeAttribute`.

(2) `removeAttribute(item)` take the item and recursively searches all the attributes of that item and removes and removes ounce the value from the attributDescriptor them and updates the corresponding `releaseviews`. Ounce done this triggers the finalizer function

(3) finalizer(attributeDescriptor) updates all the calculated values in an attributeDescriptor.

An `Item` is replaced from its collection in the following steps:

(1) itemReplace(Release, Item, ItemID) removes and replaces the item in the input Release r; then executes the functions removeAttribute(id: itemID, r: Release) and insertAttribut(c: collection, i: Item) to replace the item.

5.2.2 COMPARING AND COMBINING VIEWS

Part of the objective of the `view` is to be able to compare attributes between each other of the same or different `Views`. The intuition is to determine how similar or different are the releases of a `dataCollection` and how different are two collections. For this, we defined operations:

- to compute the union of two `attributDescriptor`,
- to determine in which degree the `attributDescriptor` of a `view` are similar among each other,
- to compute the degree in which two `attributeDescriptors` are similar builds a set of `attributeDescriptors` that appear in two releases of the same or different `views` (`common`) or that are unique within each set (`uncommon`).

We define these functions in the next lines.

ad:<attributDescriptor> $_1$ and R_2<releases:{attributeDescriptors}>

Given two attributDescriptor ad_1:<attributDescriptor> and ad_2:<attributDescriptor>

(2) The operation `Union(`ad_1`, `ad_2`)` combines the data of 2 `attributeDescriptors` ad1 and ad_2 into one ad_u. For this, each value of the `valueDistirbutions`, the absent values, null values and the size of the `attributeDescriptor` are added together. For the values distribution, this is done in two steps: first we create an object and insert all the values of the first `attributeDescriptor`, then for each value in common we add their count, for those who don't exist we insert the vales from the second attribute descriptor. The calculated values are then recalculated with the function `finalizer()`.

(3) The operation `common(`ad_1`, `ad_2`)` computes a set of releases ad_C:{ `attributeDescriptors` } where every element ad_C appears both in ad_1`.valueDistirbutions` and ad_2`.valueDistirbutions` with `elements` with the same values. For this, for each value in the `valueDistribution`, `absentValue` and `nullValue` we take the smallest of the 2 attributes descriptor. The size is recounted before execution the function `finalizer()`.

(4) The operation `noncommon(`ad_1`, `ad_2`)` does the opposite of the common operation. For this, for each value in the `valueDistribution`, `absentValue` and `nullValue` we substract their values of the 2 attributes descriptor. For the values distribution, this is done in two steps:

   a. Create an object and insert all the values of the first `attributeDescriptor`,
   b. For each value in common we subtract their count, if the count is negative we simply multiply by -1, for those who do not exist we insert the values from the

second attribute descriptor. The calculated values are then recalculated with the function `finaliser()`.

The similarity operation determines how similar the `attributeDescriptor` of two releases are. As stated previously, the similarity operation produces an `attributeDescriptor` like object. What we mean is for each calculated value in an `atttributeDescriptor` is not calculated from the `valueDistribution` but is computed based on the similarity of the values. We defined similarity as:

$$Sim(v_i,v_2)=(v_1-v_2)/(v_1+v_2)$$

The operation functions simply by running this equation on every numbered value in the `attributeDescriptor`.

5.3 EXPERIMENTS AND USE CASE

We implemented the lowest layer of our service-oriented architecture devoted to data collection harvesting and storage, which is the basis for providing data curation services. Then, we conducted experiments to estimate the cost of generating `views` and we developed a use case showing how `views` can be used for making decisions about the best way of sharding and storing data collections.

Figure 33 shows the setting of our experiment consisting of three layers. The experiment workflow starts with data harvesting tasks from providers of post mortem data collections and streams. Our data providers are accessible on the Web either as services like social network ones dealing with urban computing issues like traffic status and static data collections available in portals such as the Grand Lyon.

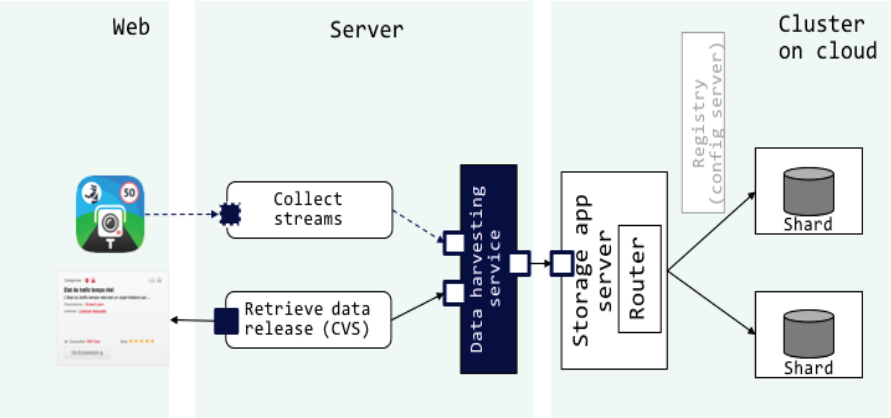

Figure 33 Experiment setting for data sharding

Data harvesting tasks are integrated behind a data harvesting service that interacts with storage services installed in a cluster on the cloud that gives access to a distributed persistence layer. This layer is used by applying sharding strategies chosen under a decision-making process based on the structural characteristics of the data and their content shown be `views`. In the following section we describe our strategy for estimating the cost of generating `views` from data collections. We also describe the use of `views` for making decisions on how to best shard a data collection across cluster elements.

5.3.1 ESTIMATING THE COST OF CREATING DATA COLLECTIONS & VIEWS

As part of our experiments, we performed a cost analysis on the production of our model using two datasets `grandLyonEvent` and `Twitter` (Table 7). As shown in Table 7 we varied the number of releases and therefore the total number of documents. Note that the documents from each collection vary greatly, in the number of attributes: `grandLyonEvent` having on average 36 attributes and Tweets 98 attributes.

Table 7: Data collections description

		Nb. of documents	Nb. Of releases
Data collections	grandLyonEvent	2095 (2.5 Mo)	86
	Twitter	736242 (2.5 GO)	125

We performed our experiments first in one machine, assuming that data scientists at the beginning of their experiments do not necessarily use complex computing systems to perform them. Then we migrated our experiments into a cluster setting. The cluster-based setting runs on a total 16 machines. Each machine runs on an Openstack cloud IaaS as an Openstack `m1.xlarge`. consisting of 4 VCPU of 2.5 Ghz, 8192 Mo RAM and 80 Go of disk memory.

We therefore prepared the following logic architecture to test the cost of distributing the data into a distributed master slave architecture. The architecture copes with the MongoDB sharding solution, the NoSQL system that we used as storage support. The logic architecture used by MongoDB consists of a Mongo server (`config server`) that serves as master and access point to a set of shards stored in other Mongo servers. A router server serves as registry to host an index that will maintain information about the distribution of data across shards. This logic architecture was deployed on one virtual machine ready to run experiments: creation of `dataCollection`s and `view`s from the initial datasets described in Table 7.

The 16 machines of the cluster-based setting “extend” the initial logic architecture of used by MongoDB as follows: 3 “`config`” `servers`, 4 `routers` and 9 `data shard servers` that replicate datasets distributed into 3 shards.

The experiment we conducted consisted in programming scripts and statistical operators to process data collections and generate objects instantiating our model. The objective was to measure execution time to profile the cost of extracting meta-data and computing statistical

measures to compute quantitative meta-data. Time obtained for the creation of `dataCollection` and `views` is shown in Figure 34. We observe that the production time varies greatly with the size of the collection, ranging from about 2 min for a small collection of 2000 documents to 36 hours for the 700000 documents collection.

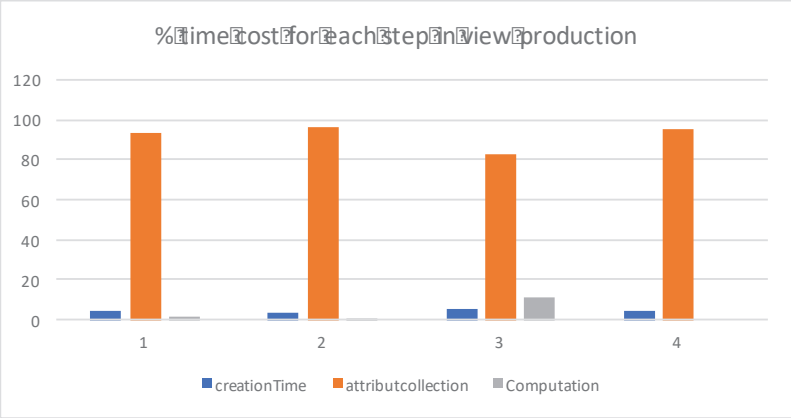

Figure 34: Relative time cost of producing views

Table 8 Map-reduce view creation time in ms

View creation MR		
		Total
1 machine	grandLyonEvent	6559
	Twitter	3145687
Cluster	grandLyonEvent	10679

After the observations, we made on the first experiment, we decided to perform an experiment varying each variable independently. The variables were the average number of attributes per document, number of release, and number of documents. This was performed on an Openstack virtual machine with Ubuntu 14.04.3 with a 2Ghz quad core VCPU and 8GO of RAM. The times given are in milliseconds. The collections are created as sample of the

grandLyonEvent store and the Tweets store with the following conditions (Table 9). Note the smaller the division of time of more release are produced since there are more time windows to use.

Table 9: experiments

n°	*Store*	*mean nb Attribut*	*time divisions in h*
36 : 24	grandLyonEvent	36	24
36 : 1	grandLyonEvent	36	1
98 : 24	twitter	98	24
98 : 1	twitter	98	1

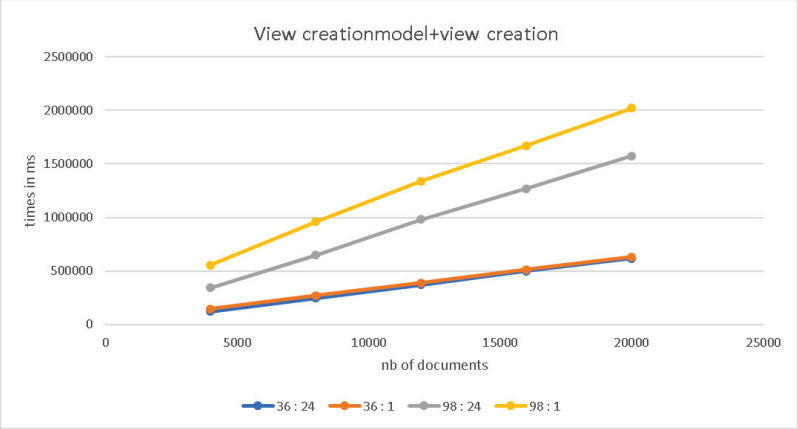

Figure 35: View + Collection creation time

First, as shown in Figure 35, what we observe is that the production time varies linearly with the size of the data store both with the number of documents and the number attributes within these documents. There is small effect on the time from the number of releases which we will see later is due to the computation time.

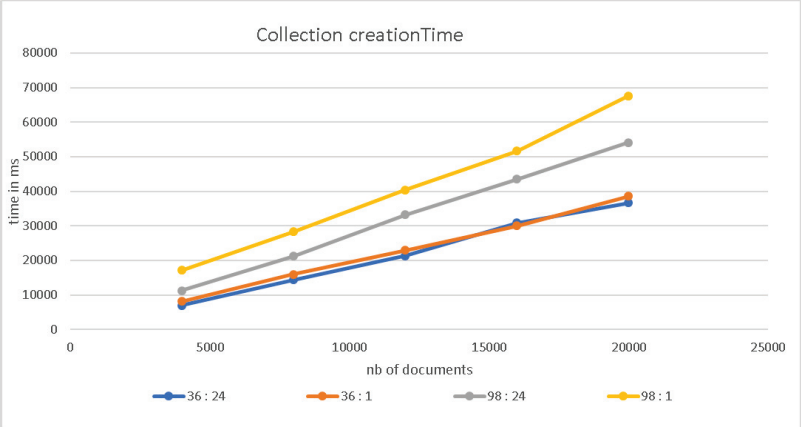

Figure 36: Collection creation time

The creation of a `dataCollection`, as shown in Figure 36, is mainly affected by the number of documents which increases linearly with the number of documents. It is also affected to a smaller degree by the number of attribute. This makes sense first, because this step essentially processes the document structure to assign them a release, secondly more attributes mean more memory usage when saving the document to disk in a new release. This still remains fairly insignificant versus the attribute collection. This process contributes for 3-6% of the total creation time.

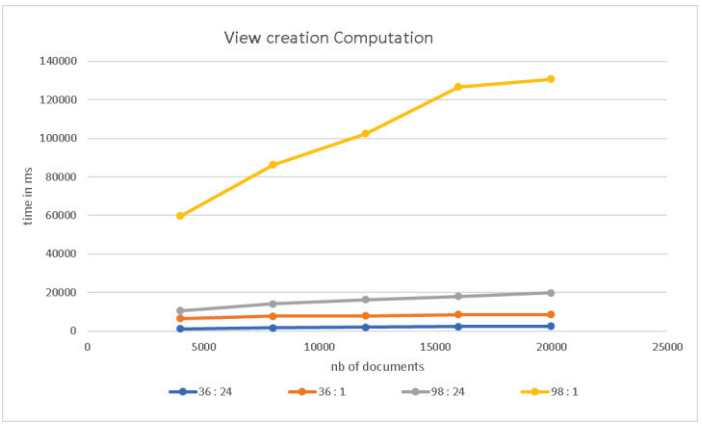

Figure 37: computation time

The computation stage, shown in Figure 37, is affected by the number of attributes and the number of releases as these affect the overall number of documents that have to computed and stored. The contribution of the number of document is fairly minor as the computation time only doubles when the number of document increases 5 times. This makes the time spent in this step far more significant for small collections with a small number of attributes than a large number of releases. This step contributes for 0.5-10% of the production time.

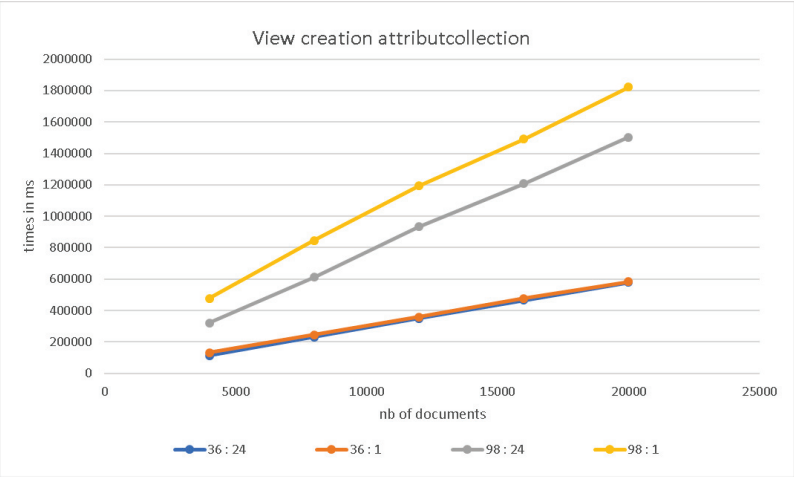

Figure 38: Attribute collection

As shown in Figure 38 the `attributeCollection` stage time increases linearly with the number of attributes and the number of documents. The increase with the number of attributes and documents make sense since they are both the main contributor with the number of database requests. This stage is by far the slowest contributing for 90-95% of the overall production time, since it has to find all the attributes in each document and the insert the data into a new document resulting in tens of thousands of database requests. On the other hand, it can be greatly improved through the use of parallel computing as seen in Figure 39 running between 4 times in the case of small number of document and high number of

releases and 16 for high number of collections and low number of releases. This will further increase with the size of the collection and it can be noted that a `view` of 200000 documents produced in parallel (using the map-reduce programming model) runs faster than 20000 documents run in a sequential execution.

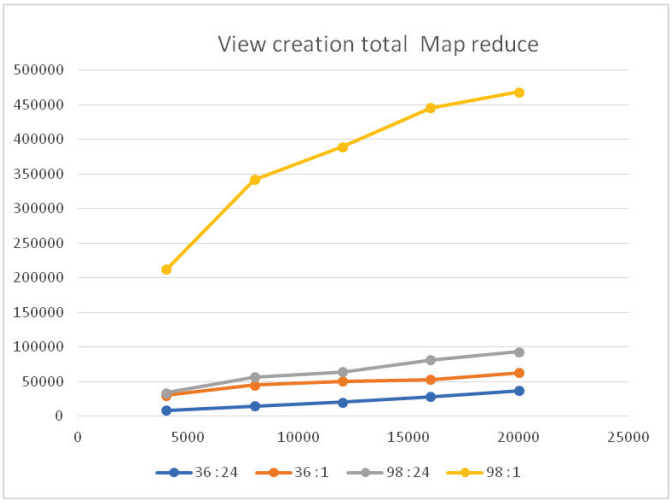

Figure 39: view creation Time total

The sequential execution could in theory be further improved by the use threading at last in the collection creation phase and the computation phase but that was unfortunately not available for JavaScript scripts.

5.3.2 MAKING DECISIONS FOR STORING DATA COLLECTIONS

Services of the layer harvesting and storage data collections are implemented in the cluster architecture that we have previously described. The cluster architecture is also exploited so that we cope with a logical architecture of stores that are prepared to host shards of fragmented data collections. The architecture is a master slave cluster consisting of a storage application server that interacts with the data providers to store data or with other services

for retrieving data. The slaves are running in an Openstack `m1.xlarge`. cloud virtual machines consisting of 4 VCPU of 2.5 Ghz, 8192 Mo RAM and 80 Go of disk memory that provide storage support for shards. The slaves are indexed by a registry that maintains information about the location of every fragment and item of the data collection within the cluster. For experimenting this configuration, we used MongoDB as reference and performed a decision-making scenario according to the three sharding strategies it proposes: hashed, interval and semantic based.

We did 3 experiments to analyse how the selection of a shard key and corresponding strategy affect the performance and distribution of the database (see Figure 40). So, from a collection of urban data tweets (10 Go), we did two experiments on the `user.location` attribute, one ranged with imposed rules to how the data must be distributed. We choose the ranges: `[MinKey → "Lyon, FRANCE"]`, `["Lyon, FRANCE" → "Lyon, Rhône-Alpes"]`, `["Lyon, Rhône-Alpes" → MaxKey]`. Then, we did another experiment with the hashed strategy on the same attribute. In this last case, the data store (MongoDB) tries to balance shards. We used both experiments to observe the behaviour of the store in the presence of queries (reads). We also did another experiment using "`_id`" attribute to observe the effect of sharding on a randomly generated unique value.

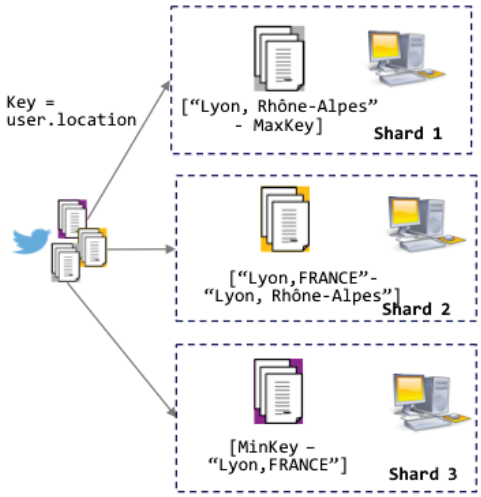

Figure 40 Experimenting data sharding strategies

These strategies are chosen under a decision-making process done by a data analyst according to the data to be sharded. We ran experiments to determine what were the technical, quantitative and structural criteria that were key for deciding whether to use one of these sharding strategies. We observed as expected that the most balanced was ranged "`_id`" then ranged "`user.location`" since the database attempts to optimize the distribution and optimizing the distribution of set of random unique value is easier than sets with multiple times the same value. The ranged strategy was less effective with 4 times more data in one shards than the others in spite having chosen ranges dividing the collection into 3 equal sizes (Figure 40). We then did 6 queries to observe the effect of different types of queries:

Two simple filter queries we used:

- {user.location: «Lyon»} user.location = Lyon
- {user.location:null} user.location = null

Two of the queries using regular expressions:

- {user.location:/^lyon$/i} user.location = Lyon ignoring case
- {user.location:/lyon/i} user.location containing Lyon ignoring case

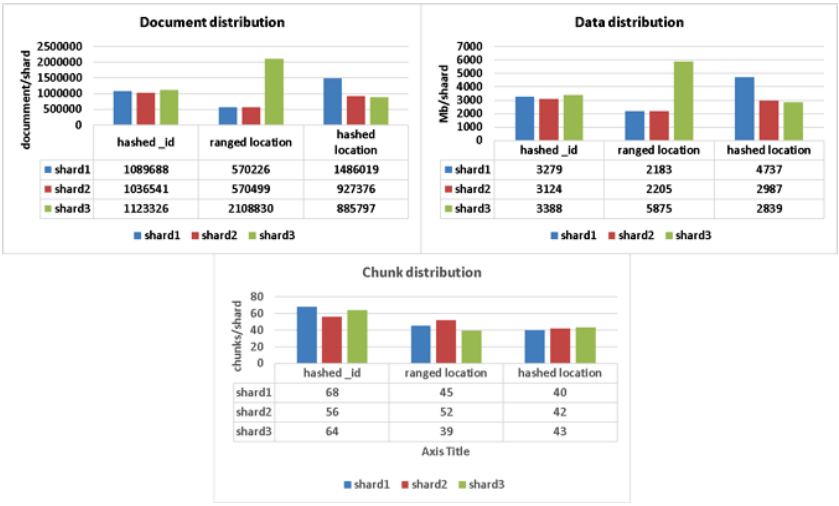

Figure 41: Data distribution according to different sharding strategies

Two other queries on another attribute corresponding to a copy of the attribute `user.location` to observe the effect on non-indexed attribute:

- Search for the document with the `locArray` equal to Lyon ignoring case: `{locArray:/^lyon$/i}`
- Search for the document with the `locArray` equal to `Lyon: {locArray: «Lyon»}`

We observed that the ranged sharding provides the best query time, very slightly better than the hashed location, on the index equal queries (see Figure 41). On the other hand, ranged hashed `_id` provided the most consistent times. In our case the hashed location is probably the better choice since it provides both quick simple queries, but the other are still efficient.

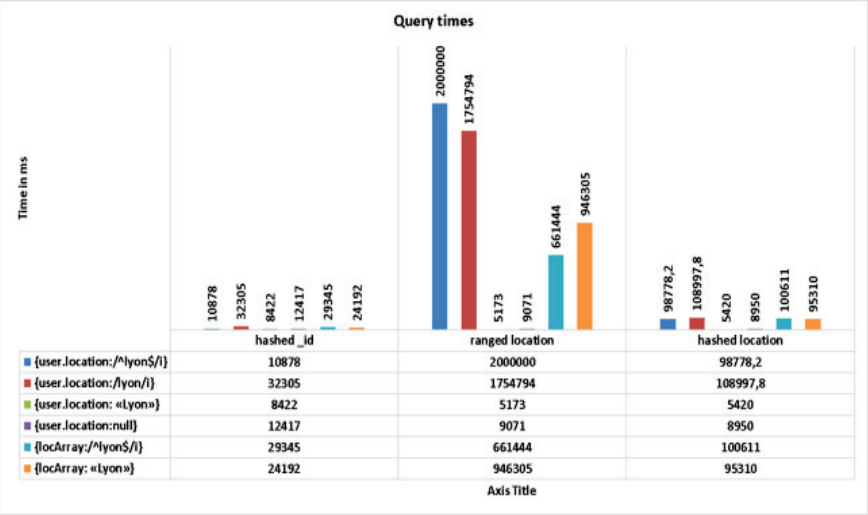

Figure 42: Query evaluation performance on a sharded database

So, we conclude that the distribution of the values of the attribute chosen as key for sharding the collection, together with the type of queries that can potentially be asked, are two key criteria to be considered for deciding how to shard a collection and maintaining it within a curation process.

This use case shows decisions making is done for Big Data (see Figure 42). For example, in the case of choosing the attribute that will provide the best sharding key across a cluster according to different strategies. It is important to note that finding the ranges to use is not a simple and straightforward process. It implies identifying the attributes which could be used as key for sharding the collection, an awkward process, when the attributes change in presence and number. For example, tweets have on average 74 attributes but in reality we have found up to 800 different attributes for tweets.

Once an attribute has been chosen as sharding key, for an interval-oriented sharing strategy it is necessary to identify the ranges of values which would cleanly divide the data collection

into thirds. This mean looking through thousands of documents and trying to identify values which will a value which will divide the collection into thirds. Looking through the values allow for at best an educated guess. In fact, I ended writing a script returning me the value of the 2 documents on the boundaries on if each thirds. This took several days of investigation. Surely there must be a more efficient way.

5.3.3 MAKING DECISIONS USING VIEWS

In this section, we look at how to shard a database using `views`. In this scenario we are seeking to shard effectively a tweets database of 3190382 documents, containing 10 GB of tweets having on average 74 attributes, optimized to answer geographical queries. After producing the `views`, sub divided with respect to the posting date, we had 125 `ReleaseViews` containing between 620 and 887 documents representing attribute descriptors (Figure 43). This reveals that a large number of attributes is not present in most documents, and second this is beyond what is practical to investigate every attribute individually, in fact identifying all existing attributes would be a time-consuming challenge.

```
{
        "_id" : "tweets_3190382_viewMapReduce_24_16999.entities.hashtags.0.text.number",
        "value" : {
                "data" : [
                        1,
                        5,
                        6,
                        7,
                        7,
                        7,
                        7,
                        92
                ],
                "median" : 7,
                "count" : 8,
                "valueDistribution" : {
                        "1" : 4,
                        "5" : 4,
                        "6" : ,
                        "7" : 16,
                        "92" : 4
                },
                "type" : "number",
                "mode" : {
                        "value" : "7",
                        "count" : 4
                },
                "missing" : 21491,
                "nulls" : null
        }
}
```


Figure 43: attributDescriptor example

On the other hand, since `view` transfer the attributes of each document to a value in an `attributeDescriptor` under the "_id" attribute, a query using a regular expression can be used to identify useful `attributDescriptors`. Having chosen to look into geography optimisation for this experiment, we looked for `attributDescriptors` containing "`loca`", "`zone`", "`area`", "`geo`" and "`coor`" in their "`_id`" since these seemed the best terms used the reveal location. This means we can immediately reduce the number of

candidates for 887 attributes to a total of 13 attributes usable for identifying the location of the user producing the tweet. These attributes are:

- "tweets.quoted_status.user.location.string"
- "tweets.user.location.string"
- "tweets.user.time_zone.string"
- "tweets.place.bounding_box.coordinates.0.0.0.number"
- "tweets.place.bounding_box.coordinates.0.0.1.number"
- "tweets.place.bounding_box.coordinates.0.1.0.number"
- "tweets.place.bounding_box.coordinates.0.1.1.number"
- "tweets.place.bounding_box.coordinates.0.2.0.number"
- "tweets.place.bounding_box.coordinates.0.2.1.number"
- "tweets.place.bounding_box.coordinates.0.3.0.number"
- "tweets.place.bounding_box.coordinates.0.3.1.number"
- "tweets. quoted_status.user.geo_enabled.string"
- "tweets.user.geo_enabled.string"

A quick investigation of each of these attributeDescriptors (Table 10) will immediately eliminate most of these options. The objective is to find an attribute which allows to share the collection into 3 separate and balanced shards:

- "tweets.place.bounding_box.coordinates" contains only a small number values with one value appearing in almost 60% of the documents making it a poor candidate for a shared key;
- "tweets.quoted_status.user.location.string" has on average more than 95% of it values missing;
- "tweets.user.geo_enabled.string" only contains Boolean values;

- "tweets.quoted_status.user.geo_enabled.string" only contains Boolean values and has more than 95% missing values
- "tweets.user.location.string" and "tweets.user.time_zone.string" has a more homogeneous distribution of the number of different values and most documents contain them.

Table 10: View Interesting values

	nb values	missing	max count	min count
tweets.quoted_status.user.location.string	8436	3008392	16820	4
tweets.user.location.string	11041	571962	269506	4
tweets.user.time_zone.string : 149	149	860539	870263	4
tweets.place.bounding_box.coordinates.0.0.0.number : 3	3	0	1983882	19950
tweets.place.bounding_box.coordinates.0.0.1.number : 3	3	0	1968043	35784
tweets.place.bounding_box.coordinates.0.1.0.number : 3	3	0	1983882	19950
tweets.place.bounding_box.coordinates.0.1.1.number : 3	3	0	1969602	34231
tweets.place.bounding_box.coordinates.0.2.0.number : 5	5	0	1898817	19950
tweets.place.bounding_box.coordinates.0.2.1.number : 3	3	0	1969602	34231
tweets.place.bounding_box.coordinates.0.3.0.number : 5	3	0	1898817	19950
tweets.place.bounding_box.coordinates.0.3.1.number : 3	3	0	1968043	25784
tweets.quoted_status.user.geo_enabled.string : 2	2	2938908	155678	95665
tweets.user.geo_enabled.string : 1	1	0	3190337	3190337

Let us start with "tweets.user.time_zone.string" since it is a simpler attribute to investigate. The objective of sharding is to achieve better response times for certain types of queries whilst maintaining a balanced amount of data between the shards. Using the attribute

valueDistribution of the view, we can generate a histogram of the data as shown in Figure 44. From this we can identify a number of key values:

- “Paris”: 870308 and the “missing”: 860580 values,
- “Amsterdam”: 325538 values,
- “Athens”: 362878 values,
- “Pacific Time (US & Canada)”: 357869 values hidden under Paris
- “Greenland”: 83694 values and “Ljubljana”: 76427 values.

With a quick bit of arithmetic reveals this distribution:

- Shard 1 will have values “Paris”, “Greenland” and ”Ljubljana” for a total of 1030429 documents or 32.3% of the documents
- Shard 2 will have values “Amsterdam”, “Athens” and “Pacific Time (US & Canada)” for a total of 1046285 or 32.8% of the documents
- Shard 3 will have the rest for a total of 1113668 document or 34.9% of the documents

Now let’s look at the more complex attributes like “tweets.user.location.string”. Using the valueDistribution, we can quickly generate a histogram of the data as shown in Figure 45. Immediately, we see a few values that are sticking out namely: “France”,”Lyon”, “Lyon, France” and “missing”. But after further investigation we notice a large number of similar values following themselves as revealed by this cumulative distribution (Figure 46).

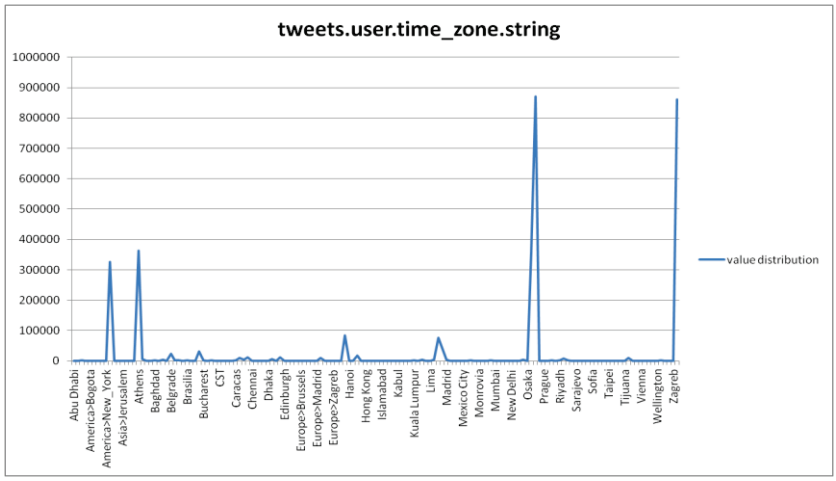

Figure 44: Value distribution of the attribute tweets.user.time_zone.string

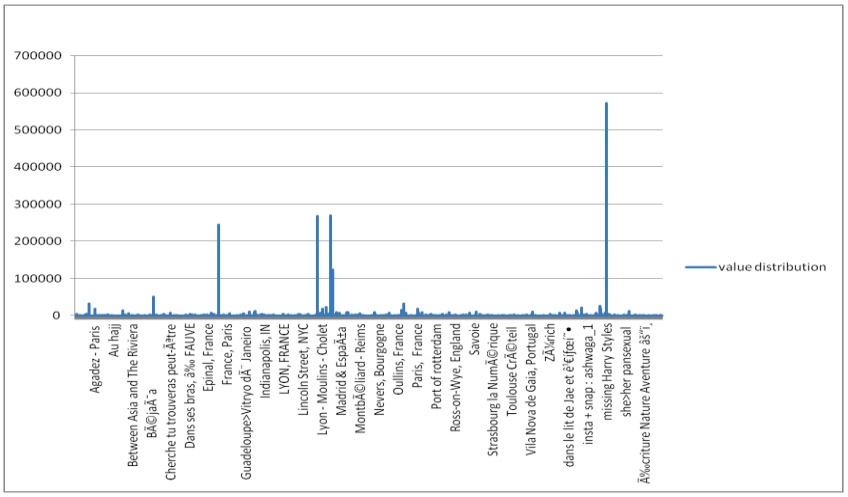

Figure 45: Value distribution of the attribute tweets.user.location.string

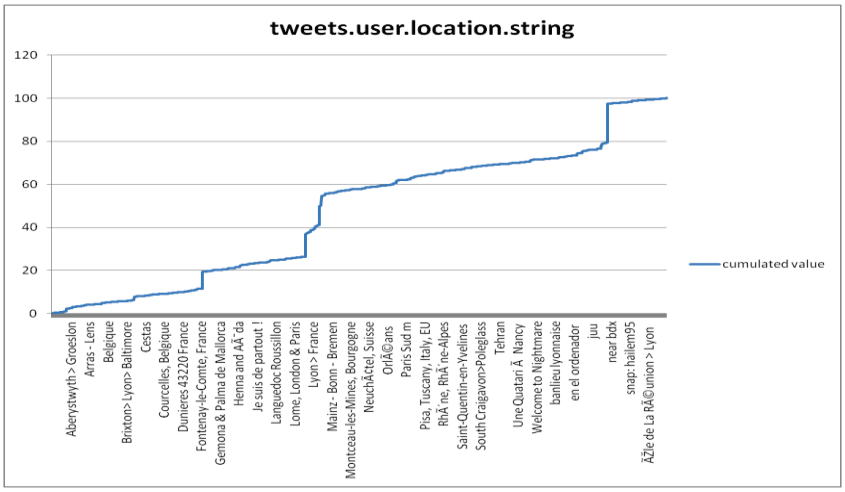

Figure 46: Cumulated distribution of the attribute tweets.user.location.string

Using this graph allows us to find a range of values to use for sharding. We choose to find ranges which would distribute the data most evenly, but which also allowed to maintain shard with focused on certain regions. This lead us to estimate the sharding ranges as follows by first exploring the major leaps in value around “Lyon”, then “France”, then “missing”:

- Shard 1: has all the data related to Lyon i.e. “L69n” → ”Lyonâ˜”ï¸“ containing 1012601 document or 31.8% of the documents
- Shard 2: has all the data related to France i.e. “Franceâ™•â„¢” → “Fr“ and from missing to maximum value i.e. “missing” → KeyMax, this contains 925877 document or 29.0% of the document
- Shard 3: has all the rest corresponding to 1251813 documents or 39.2% of the documents

In this case, the attribute “tweets.user.time_zone.string” provides a better distribution as well a more wieldy values to use when producing queries.

This process demonstrates the effectiveness of views. First it reduced the number of attributes to investigate from 887 to 13 then 2, second it provides a quick and effective way

of finding an effective sharding solution. Finally, it provides a light weight easy to manipulate and easily explainable data to explain choices within the decision-making process. So, beyond the cost, an important aspect to keep in mind is that having collections' `views` reduces the effort of exploring manually or in artisanal manner data collections, for understanding their content and making decisions on how to organize and use them for target requirements

5.4 ANALYSIS OF EXPERIMENTS

CURARE can help a data analyst decision making providing services and strategies regarding raw data collection harvesting and storage processes. CURARE gives information that can be used to determine the best way to manage and exploit data: storage strategies and distribution, data pre-processing according to the status of the content (missing, null values), and data preparation required to apply specific data analytics algorithms.

We observed from our experiments that the creation of this curation model is quite costly for mid to large collections with complex data structures (many attributes) when using sequential computing. The CURARE services implementing our data collections view model can use different algorithms and strategies to implement creation operations. This is necessary depending on the content of the data collections. For example, computing the distribution of the values of attributes when they correspond to multimedia data, or to data types that need to be pre-processed to compute values statistics.

We also observed that making decisions on Big Data sets is challenging. But that these decisions have a massive impact on the efficiency of the database. As we can see it took me several days to identify shard key ranges and resulted in providing an answer which is passable at best. On the other hand, when using views, it allowed me (1) identify much more candidate attribute by querying the views, (2) to investigate much faster the potential of the candidate

and (3) to identify ranges much faster and much more effectively. Essentially rather than taking day to find a sharding solution which is only passable, it took me only half a day to give a far more nuanced and good solution.

6 CONCLUSIONS AND PERSPECTIVES

The initial objective of this thesis was to propose a data curation model that can model meta-data describing the structure, content and conditions in which data collections are produced; and a service-oriented data curation environment for harvesting, cleaning, processing data collections for computing discovering and deducing meta-data and storing data for supporting the design of data centric experiments though exploration operations.

Accordingly, we proposed a meta-data model, the view model, and operations to explore this information and enable data curation processes within an integrated environment named CURARE.

6.1 SUMMARY OF THE WORK AND CONTRIBUTION

This thesis has explored of service-oriented architecture on the cloud and data curation and exploration in the context of Big Data. From the state of the art we determined that using Big Data relies on making many decisions related to the data. Most data curation models are designed for a specific field. Techniques used to perform the data curation are either limited in scope context or rely on a substantial amount of person power. Other methods have taken the angle of providing information to data analysts through the extraction of meta-data. These methods lack quantitative meta-data (distribution of the values for a given attribute) necessary to evaluate the state and quality of data collections. Data analysts have a lot of decisions to make in the hope to produce effective and potentially efficient tools and services to choose which data collections are best for performing specific analysis, and also to choose the best strategies to maintain and share them.

Our work focused on the extraction, computation and deduction of quantitative and qualitative meta-data that ease the data curation process. Thus, we proposed data curation model and tools implemented by CURARE addressing the difficulty of semi-manual data analysts' tasks by providing quantitative information on the data they are curating.

The CURARE service-oriented architecture defines types of services based on their tasks: harvesting, pre-processing, storage, processing, analysis and decision support. We described the type of resources these service use and the general interfaces they use to communicate and the Data Science Virtual Machine (DSVM) services they use to implement their functions.

For our data curation model, we focused on providing visual and quantitative meta-data so the data analyst can investigate more rapidly their data and make strategic decision. We implemented the data collection model which is designed to track the sources and maintain track of how the collections evolves over time through the use of release, set of data produced from a source at a particular time. We also implemented the data view model which is designed to explore each attribute of a data set through the use of attribute descriptor. We also implemented a full suite of operators to update and combine data views and data collections.

We finally experimented with Big Data decision making and demonstrated the usefulness of data views. In the experiment, we ran two sharding experiments. The first to evaluate the importance of sharding, we also observed the time it took to perform the decision. And the second we attempted to shard a data set using view. We observed from these 2 experiments that sharding can have a lot of effect on the efficiency of the database. We also observed that making decisions without tools to assist is difficult, inaccurate and long whereas when using views, we were able to make sharding decisions much more accurately and quickly. We also

ran an experiment to evaluate the cost of creating this model. We observed whilst this model is heavy for linear processing, using parallel processing makes it well manageable.

6.2 FUTURE WORK AND PERSPECTIVES

Future work will tackle operators' experiments for easing the construction and preparation of data collections and adding visualisation tools that can help to compare different releases of the same data collections or different data collections. We are currently using CURARE to explore newspapers collections and political campaigns data collections in the context of e-social sciences projects [117]. CURARE is also being experimented for exploring heterogeneous datasets that compare neurosciences experiments [118].

This work and of course the state of the art have shown that the expectations of a tremendous productions of data announced in the last years due to the evolution of technology and the progressive consolidation of the Internet of Things is not science fiction. Yet, from our perspective, the real challenge is not introduced by the volume and the velocity of data collections, nor by its variety but about the conditions in which such data collections will be processed for creating novel applications.

Which are the kinds of applications that will be ready to efficiently consume data while ensuring a smart vision and exploitation of data collections? Do we need well adapted machines for addressing this problem? Who is going to get value out of data collections

processing? Are current data harvesting and storage techniques adapted for brontobytes[48] and Geopbytes[49] of data? The objective will be to design and develop novel ways of curating, exploring and exploiting data collections and propose alternative ways of dealing with the "Big Data reloaded".

6.2.1 COMPOSING AD-HOC DATA CURATION AND EXPLORATION ENVIRONMENTS

The first perspective of our work concerns the way data curation services can be delivered and personalized in CURARE according to data collections characteristics. The current CURARE architecture does not provide the possibility of defining explicit curation workflows that can compose one or several services for providing a solution. This would allow data analysts to choose services according to both their data requirements their objectives and other functional and non-functional qualities criteria. For example, choosing reliant services, choosing the metrics and dependencies of these services according to data collections. The objective would be to provide data curation and exploration workflows for data collections composing services according to QoS criteria. Workflows could be shared with other data analysts as cases to be applied and reused for other tasks.

6.2.2 HUMAN IN THE LOOP BASED DATA EXPLORATION

Data curation and exploration is an important step to integrate data collections and provide users with a unified view. However, data collections integration cannot be completely addressed by purely automated methods [119]. Therefore, it is demanding to develop

[48] A Brontobyte is a unit of data that represent a very large number of bytes. It is often compared to approximately 1000 Yottabytes; the specific number being 10^{27} bytes.
[49] After the brontobyte comes "geopbyte" (a thousand brontobytes)10^{30} bytes.

effective techniques and systems that integrate the intervention of humans to serve the data integration problem.

Recent technology trends (such as touch screens, motion detection, and voice recognition) are widening the possibilities for users to interact with systems, and many information-provision industries are shifting to personalized processing to better target their services to the users' wishes. The issue with data collections curation is that it is important for some applications in electronic social sciences, in digital humanities, in neurosciences to track the operations applied to curate and explore them. Since data collections are potentially used by potentially millions of users and processed by millions of processes, it is important to have automatic and fine control on the operations applied on them. The emergence of blockchain approaches [120], [121], [122] that keep track of operations in an anonymous way can be a novel approach for having a decentralized highly distributed and collaborative data curation model and thereby provide new alternatives to store, disseminate and exploit data collections.

6.2.3 DATA COLLECTIONS AND BIG DATA SERVICE MARKET

In the information era users' have to make decisions and generate knowledge by dealing with huge amounts of data. Putting order to data collections takes time and people and organizations invest energy finding, retrieving and, organizing data of different provenances and qualities. Effort in creating these data collections (links collections, images, documents) and then in summarizing and generating information out of them is rarely shared. These collections of curated data often remain in the logs and histories of personal computers. A good number of person/hours are somehow wasted or at least they are not capitalized once the data has been used or not. Consider instead a scenario where data tagged with comments

on consumers' experiences and opinions about their quality and usefulness, are exported as a data market with an associated cost model.

Our vision is that it is necessary to see data collections curation that goes beyond accessing timely and costly ready to use data sources. It should be seen as an effort that implies economically capitalizing the effort of going out for hunting data sources and services of different qualities and stemming from different processing processes, curating and delivering them. We shall call this environment a data market because we will assume that data brokers have an associated cost model. These brokers can be then contacted for accessing to data collections that can be processed for building new data collections that can be then made available in the data market.

The key issues here are: (i) associate a cost model for the data market, i.e., associate a cost to raw data and to processed data according on the amount of data and processing resources used for treating it, for instance; (ii) combine these cost model and the consumer expectations (service level agreement) with processing resources cost required by data processing; (iii) extend CURARE like data management and brokering processing mechanisms under ad hoc business models.

The objective will be to propose solutions for curating and exploring data collections according to a cost model and a business model that can provide a strategy guaranteeing given quality of service criteria ranging from privacy, economic cost, provenance, reputation, trust.

BIBLIOGRAPHY

[1] D. Howe *et al.*, "Big data: The future of biocuration," *Nature*, vol. 455, no. 7209, pp. 47–50, Sep. 2008.

[2] Andrew Cave, "What Will We Do When The World's Data Hits 163 Zettabytes In 2025?," *forbes*. [Online]. Available: https://www.forbes.com/sites/andrewcave/2017/04/13/what-will-we-do-when-the-worlds-data-hits-163-zettabytes-in-2025/#6fa6e658349a. [Accessed: 12-Jun-2018].

[3] Gartner, "'Dirty Data' is a Business Problem, Not an IT Problem, Says Gartner," 2007. [Online]. Available: http://www.gartner.com/newsroom/id/501733. [Accessed: 07-Mar-2017].

[4] I. Terrizzano, P. Schwarz, M. Roth, and J. E. Colino, "Data Wrangling: The Challenging Journey from the Wild to the Lake," in *7th Biennial Conference on Innovative Data Systems Research (CIDR '15) January*, 2015.

[5] T. Becker, E. Curry, A. Jentzsch, and W. Palmetshofer, *New horizons for a data-driven economy: Roadmaps and action plans for technology, businesses, policy, and society*. 2016.

[6] NIST Big Data Public Working Group, "NIST Special Publication 1500-1 - NIST Big Data Interoperability Framework: Volume 1, Definitions," *NIST Spec. Publ.*, vol. 1, p. 32, 2015.

[7] "Facing the threat: Big Data and crime prevention - Internet of Things blog." [Online]. Available: https://www.ibm.com/blogs/internet-of-things/big-data-crime-prevention/. [Accessed: 05-Jul-2018].

[8] R. Y. Wang and D. M. Strong, "Beyond Accuracy: What Data Quality Means to Data Consumers," *Source J. Manag. Inf. Syst.*, vol. 12, no. 4, pp. 5–33, 1996.

[9] S. Idreos, O. Papaemmanouil, and S. Chaudhuri, "Overview of Data Exploration Techniques," *Proc. ACM SIGMOD Int. Conf. Manag. Data, Tutor.*, pp. 277–281, 2015.

[10] P. M. and T. Grance, "The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology," 2008.

[11] V. Gulisano, R. Jiménez-Peris, M. Patiño-Martnez, C. Soriente, and P. Valduriez, "StreamCloud: An elastic and scalable data streaming system," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, pp. 2351–2365, 2012.

[12] H. V. Jagadish *et al.*, "Big Data and Its Technical Challenges," *communication of the acm*, vol. 57, no. 7, pp. 86–94, 2014.

[13] Z. Zheng, J. Zhu, and M. R. Lyu, "Service-generated big data and big data-as-a-service: An overview," *Proc. - 2013 IEEE Int. Congr. Big Data, BigData 2013*, pp. 403–410, 2013.

[14] Brandtzæg Petter Bae, "Big Data, for Better or Worse: 90% of World's Data Generated Over Last Two Years - SINTEF," *SINTEF*, 2013. [Online]. Available: https://www.sintef.no/en/publications/publication/?pubid=CRIStin+1031676. [Accessed: 07-Jun-2018].

[15] M. Winans *et al.*, "10 Key Marketing Trends for 2017 Customer Expectations and Ideas for Exceeding Customer Expectations," *IBM Offer. Inf.*, p. 18, 2016.

[16] D. Laney, "3D data management: Controlling data volume, velocity, and variety," *Appl. Deliv. Strateg.*, vol. 949, no. February 2001, p. 4, 2001.

[17] Robert Hillard, "Information Development » Blog Archive » It's time for a new definition of big data," 2012. [Online]. Available: http://mike2.openmethodology.org/blogs/information-development/2012/03/18/its-time-for-a-new-definition-of-big-data/. [Accessed: 07-Jun-2018].

[18] Mike Loukides, "What is data science?," *O'Reilly Media*, 2010. [Online]. Available: https://www.oreilly.com/ideas/what-is-data-science. [Accessed: 07-Jun-2018].

[19] Jacobs and Adam, "The Pathologies of Big Data," *Queue*, vol. 7, no. 6, p. 10, 2009.

[20] Networked European Software and Services Initiative (NESSI), "Big Data: A New World of Opportunities," *Big Data A New World Oppor.*, no. December, p. 25, 2012.

[21] Michael Stonebraker, "What Does 'big Data' Mean? | blog@CACM | Communications of the ACM," 2012. [Online]. Available: https://cacm.acm.org/blogs/blog-cacm/155468-what-does-big-data-mean/fulltext. [Accessed: 07-Jun-2018].

[22] "The 10 Vs of Big Data | Transforming Data with Intelligence." [Online]. Available: https://tdwi.org/articles/2017/02/08/10-vs-of-big-data.aspx. [Accessed: 30-Jun-2018].

[23] “The 42 V’s of Big Data and Data Science.” [Online]. Available: https://www.elderresearch.com/blog/42-v-of-big-data. [Accessed: 30-Jun-2018].

[24] B. Stein and A. Morrison, “The enterprise data lake: Better integration and deeper analytics,” *Technol. Forecast Rethink. Integr. Issue*, vol. 1, 2014.

[25] I. Terrizzano, P. Schwarz, M. Roth, and J. E. Colino, “Data Wrangling: The Challenging Journey from the Wild to the Lake,” in *7th Biennial Conference on Innovative Data Systems Research (CIDR ’15) January*, 2015.

[26] K. Abuosba, “Formalizing big data processing lifecycles: Acquisition, serialization, aggregation, analysis, mining, knowledge representation, and information dissemination,” in *2015 International Conference and Workshop on Computing and Communication (IEMCON)*, 2015, pp. 1–4.

[27] G. Kousiouris, G. Vafiadis, and T. Varvarigou, “A front-end, Hadoop-based data management service for efficient federated clouds,” *Proc. - 2011 3rd IEEE Int. Conf. Cloud Comput. Technol. Sci. CloudCom 2011*, pp. 511–516, 2011.

[28] M. Inc., “Sharding and MongoDB,” pp. 1–80, 2014.

[29] D. A. Keim, “Visual exploration of large data sets,” *Commun. ACM*, vol. 44, no. 8, pp. 38–44, 2001.

[30] A. F. Zuur, E. N. Ieno, and C. S. Elphick, “A protocol for data exploration to avoid common statistical problems,” *Methods Ecol. Evol.*, vol. 1, no. 1, pp. 3–14, Mar. 2010.

[31] M. H. Cragin, P. B. Heidorn, C. L. Palmer, and L. C. Smith, “An Educational Program on Data Curation,” p. 2006, 2007.

[32] E. Curry, A. Freitas, and S. O’Riáin, “The Role of Community-Driven Data Curation for Enterprises,” in *Linking Enterprise Data*, Boston, MA: Springer US, 2010, pp. 25–47.

[33] M. Pennock, “Digital Curation: A Life-Cycle Approach to Managing and Preserving Usable Digital Information,” *Libr. Arch. J.*, vol. 1, no. January, pp. 1–3, 2007.

[34] E. Curry and A. Freitas, “Coping with the Long Tail of Data Variety,” 2005.

[35] F. C. Bernstein *et al.*, “The protein data bank: A computer-based archival file for macromolecular structures,” *J. Mol. Biol.*, vol. 112, no. 3, pp. 535–542, May 1977.

[36] H. E. Pence and A. Williams, “ChemSpider: An Online Chemical Information Resource,” *J. Chem. Educ.*, vol. 87, no. 11, pp. 1123–1124, Nov. 2010.

[37] F. Khatib *et al.*, “Crystal structure of a monomeric retroviral protease solved by protein folding game players,” *Nat. Struct. Mol. Biol.*, vol. 18, no. 10, pp. 1175–1177, Oct. 2011.

[38] AQMP Specification, “AMQP v1.0 07,” *ReVision*, vol. 2011, no. revision 0, 2011.

[39] “Apache Kafka.” [Online]. Available: https://kafka.apache.org/. [Accessed: 29-Jun-2018].

[40] “Welcome to Apache Flume — Apache Flume.” [Online]. Available: https://flume.apache.org/. [Accessed: 29-Jun-2018].

[41] R. Rana, C. T. Chou, N. Bulusu, S. Kanhere, and W. Hu, “Ear-Phone: A context-aware noise mapping using smart phones,” *Pervasive Mob. Comput.*, vol. 17, pp. 1–22, Feb. 2015.

[42] M. Haklay and P. Weber, “OpenStreetMap: User-Generated Street Maps,” *IEEE Pervasive Comput.*, vol. 7, no. 4, pp. 12–18, Oct. 2008.

[43] A. Artikis, M. Weidlich, A. Gal, V. Kalogeraki, and D. Gunopulos, “Self-Adaptive Event Recognition for Intelligent Transport Management,” pp. 319–325, 2013.

[44] N. Lathia and L. Capra, “Mining mobility data to minimise travellers’ spending on public transport,” in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD ’11*, 2011, p. 1181.

[45] P. Borgnat, E. Fleury, C. Robardet, and A. Scherrer, “Spatial analysis of dynamic movements of V{é}lo’v, Lyon’s shared bicycle program,” *Eccs*, 2009.

[46] J. Candia, M. C. González, P. Wang, T. Schoenharl, G. Madey, and A.-L. Barabási, “Uncovering individual and collective human dynamics from mobile phone records,” *J. Phys. A Math. Theor*, vol. 41, pp. 224015–11, 2008.

[47] J. Bao, Y. Zheng, and M. F. Mokbel, “Location-based and preference-aware recommendation using sparse geo-social networking data,” in *Proceedings of the 20th International Conference on Advances in Geographic Information Systems - SIGSPATIAL ’12*, 2012, p. 199.

[48] J. Yuan, Y. Zheng, X. Xie, and G. Sun, “Driving with knowledge from the physical world,” *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min. (KDD ’11)*, no. 5, pp. 316–324, 2011.

[49] F. Zhang, D. Wilkie, Y. Zheng, and X. Xie, “Sensing the pulse of urban refueling behavior,” *Proc. 2013 ACM Int. Jt. Conf. Pervasive ubiquitous Comput. - UbiComp ’13*, vol. 1, no. 4, p. 13, 2013.

[50] Y. Chen, K. Jiang, Y. Zheng, C. Li, and N. Yu, “Trajectory Simplification Method for Location-Based Social Networking Services.”

[51] Gabriel Grant, “PyVideo.org · Storm: the Hadoop of Realtime Stream Processing,” *PyConUs*, 2012. [Online]. Available: http://pyvideo.org/pycon-us-2012/storm-the-hadoop-of-realtime-stream-processing.html. [Accessed: 12-Jun-2018].

[52] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, “S4: Distributed stream computing platform,” *Proc. - IEEE Int. Conf. Data Mining, ICDM*, pp. 170–177, 2010.

[53] “Welcome to Apache™ Hadoop®!” [Online]. Available: http://hadoop.apache.org/. [Accessed: 14-Jun-2018].

[54] V. Abramova and J. Bernardino, “NoSQL databases: a step to database scalability in web environment,” *Proc. Int. C* Conf. Comput. Sci. Softw. Eng. - C3S2E ’13*, no. July, pp. 14–22, 2013.

[55] R. Cattell, “Scalable SQL and NoSQL data stores,” *ACM SIGMOD Rec.*, vol. 39, no. 4, p. 12, 2011.

[56] A. B. M. Moniruzzaman and S. A. Hossain, “Nosql database: New era of databases for big data analytics-classification, characteristics and comparison,” *arXiv Prepr. arXiv1307.0191*, vol. 6, no. 4, pp. 1–14, 2013.

[57] A. Halevy *et al.*, “Goods: Organizing Google’s Datasets,” in *Proceedings of the 2016 International Conference on Management of Data - SIGMOD ’16*, 2016, pp. 795–806.

[58] M. Stonebraker *et al.*, “Data Curation at Scale: The Data Tamer System,” *CIDR 2013*.

[59] I. Terrizzano, P. Schwarz, M. Roth, and J. E. Colino, “Data Wrangling: The Challenging Journey from the Wild to the Lake.”

[60] R. Hai, S. Geisler, and C. Quix, “Constance,” in *Proceedings of the 2016 International Conference on Management of Data - SIGMOD ’16*, 2016, pp. 2097–2100.

[61] D. Feldman, M. Schmidt, and C. Sohler, “Turning Big data into tiny data : Constant-size coresets for k -means , PCA and projective clustering,” *Proc. Twenty-Fourth Annu. ACM-SIAM Symp. Discret. Algorithms*, pp. 1434–1453, 2013.

[62] S. Chawla, Y. Zheng, and J. Hu, “Inferring the Root Cause in Road Traffic Anomalies,” in *2012 IEEE 12th International Conference on Data Mining*, 2012, pp. 141–150.

[63] T. Tsuda *et al.*, “Advanced Semiconductor Manufacturing Using Big Data,” *IEEE Trans. Semicond. Manuf.*, vol. 28, no. 3, pp. 229–235, 2015.

[64] M. Kersten, S. Idreos, S. Manegold, and E. Liarou, “The Researcher’s Guide to the Data Deluge: Querying a Scientific Database in Just a Few Seconds,” in *Proceedings of the VLDB Endowment,* 2011, p. 1474.

[65] H. Weatherspoon, B. Chun, C. W. So, and J. Kubiatowicz, “Long-Term Data Maintenance in Wide-Area Storage Systems: A Quantitative Approach,” *Science (80-.).*, no. July, 2005.

[66] A. Bhardwaj *et al.*, “DataHub: Collaborative Data Science & Dataset Version Management at Scale,” *CIDR*, 2015.

[67] H. D. Morris and D. Vesset, “Managing Master Data for Business Performance Management: The Issues and Hyperion’s Solution,” *IDC white Pap.*, 2005.

[68] A. Doan, R. Ramakrishnan, and A. Y. Halevy, “Crowdsourcing systems on the World-Wide Web,” *Commun. ACM*, vol. 54, no. 4, p. 86, Apr. 2011.

[69] Paramita Ghosh, “Big Data as a Service: What Can it Do for Your Enterprise? - DATAVERSITY,” 2017. [Online]. Available: http://www.dataversity.net/big-data-service-can-enterprise/. [Accessed: 28-Jan-2018].

[70] “Définition - Services | Insee.” [Online]. Available: https://www.insee.fr/en/metadonnees/definition/c1161. [Accessed: 21-Feb-2018].

[71] “OpenStack Introduction.” [Online]. Available: https://www.slideshare.net/openstackindia/openstack-introduction-14761434. [Accessed: 04-Jul-2018].

[72] EMC Solution Group, “Big Data-as-a-Service. A market and technology perspective,” no. July 2012, pp. 1–16, 2012.

[73] J. Horey, E. Begoli, R. Gunasekaran, S.-H. Lim, and J. Nutaro, “Big data platforms as a service: challenges and approach,” *Proc. 4th USENIX Conf. Hot Top. Cloud Ccomputing*, p. 16, 2012.

[74] E. E. Schadt, M. D. Linderman, J. Sorenson, L. Lee, and G. P. Nolan, “Cloud and heterogeneous computing solutions exist today for the emerging big data problems in biology.,” *Nat. Rev. Genet.*, vol. 12, no. 3, p. 224, Mar. 2011.

[75] “BDAS, the Berkeley Data Analytics Stack.” [Online]. Available: https://amplab.cs.berkeley.edu/software/. [Accessed: 14-Jun-2018].

[76] A. Zimmermann, M. Pretz, G. Zimmermann, D. G. Firesmith, I. Petrov, and E. El-Sheikh, “Towards service-oriented enterprise architectures for big data applications in the cloud,” *Proc. - IEEE Int. Enterp. Distrib. Object Comput. Work. EDOC*, pp. 130–135, 2013.

[77] “Apache Hive TM.” [Online]. Available: https://hive.apache.org/. [Accessed: 04-Jul-2018].

[78] “Welcome to Apache Pig!” [Online]. Available: http://pig.apache.org/. [Accessed: 17-Mar-2015].

[79] V. R. Borkar, M. J. Carey, and C. Li, “Big data platforms,” *XRDS Crossroads, ACM Mag. Students*, vol. 19, no. 1, p. 44, 2012.

[80] H. Demirkan and D. Delen, “Leveraging the capabilities of service-oriented decision support systems: Putting analytics and big data in cloud,” *Decis. Support Syst.*, vol. 55, no. 1, pp. 412–421, 2013.

[81] Z. Li *et al.*, “Enabling big geoscience data analytics with a cloud-based, MapReduce-enabled and service-oriented workflow framework.,” *PLoS One*, vol. 10, no. 3, p. e0116781, Jan. 2015.

[82] “Understanding Hadoop-as-a-Service Offerings | Data Center Knowledge,” 2014. [Online]. Available: http://www.datacenterknowledge.com/archives/2014/05/14/understanding-hadoop-service-offerings. [Accessed: 28-Feb-2018].

[83] P. A. Prakashbhai and H. M. Pandey, “Inference patterns from Big Data using aggregation, filtering and tagging- A survey,” in *2014 5th International Conference - Confluence The Next Generation Information Technology Summit (Confluence)*, 2014, pp. 66–71.

[84] “Top 16 Companies in the Hadoop-as-a-Service (HDaaS) Market - Technavio.” [Online]. Available: https://www.technavio.com/blog/top-16-companies-in-the-hadoop-as-a-service-hdaas-market. [Accessed: 01-Mar-2018].

[85] Amazon, “elastic map reduce,” 2015. [Online]. Available: http://aws.amazon.com/elasticmapreduce/.

[86] “Artemis EMC^2 - Artemis EMC^2.” [Online]. Available: https://www.artemis-emc2.eu/. [Accessed: 14-Jun-2018].

[87] “Understanding InfoSphere BigInsights.” [Online]. Available: https://www.ibm.com/developerworks/data/library/techarticle/dm-1110biginsightsintro/index.html. [Accessed: 14-Jun-2018].

[88] “IBM Knowledge Center - Overview of BigSheets.” [Online]. Available: https://www.ibm.com/support/knowledgecenter/SSPT3X_4.2.0/com.ibm.swg.im.infosphere.biginsights.analyze.doc/doc/c0057518.html. [Accessed: 14-Jun-2018].

[89] “Altiscale | Crunchbase.” [Online]. Available: https://www.crunchbase.com/organization/altiscale. [Accessed: 14-Jun-2018].

[90] “Cask - Big Data Applications on Hadoop.” [Online]. Available: http://cask.co/. [Accessed: 14-Jun-2018].

[91] “CDH | Open Source | Hadoop Stack | Cloudera.” [Online]. Available: https://www.cloudera.com/products/open-source/apache-hadoop/key-cdh-components.html. [Accessed: 14-Jun-2018].

[92] “FICO® | FICO Decisions.” [Online]. Available: http://www.fico.com/. [Accessed: 14-Jun-2018].

[93] “BigQuery - Analytics Data Warehouse | Google Cloud.” [Online]. Available: https://cloud.google.com/bigquery/. [Accessed: 14-Jun-2018].

[94] “Manage Data-at-Rest and Deliver Big Data Analytics with Hortonworks Data Platform

(HDP) | Hortonworks." [Online]. Available: https://hortonworks.com/products/data-platforms/hdp/. [Accessed: 14-Jun-2018].

[95] "Infochimps Big Data Platform as-a-Service Technical Overview | Infochimps." [Online]. Available: http://www.infochimps.com/resources/technical-overview/. [Accessed: 14-Jun-2018].

[96] "The Only Converged Data Platform | MapR." [Online]. Available: https://mapr.com/. [Accessed: 14-Jun-2018].

[97] "Modern monitoring & analytics." [Online]. Available: https://www.datadoghq.com/. [Accessed: 14-Jun-2018].

[98] "Pentaho Marketplace." [Online]. Available: http://www.pentaho.com/marketplace/. [Accessed: 14-Jun-2018].

[99] "Business Analytics, Hybrid Cloud & Consulting | Teradata." [Online]. Available: https://www.teradata.com.au/. [Accessed: 14-Jun-2018].

[100] G. C. Deka and M. Labor, "Feature : Cloud Computing A Survey of Cloud Database Systems," *IT Prof.*, vol. 16, no. 2, 2014.

[101] "Voldemort." [Online]. Available: https://www.project-voldemort.com/voldemort/. [Accessed: 14-Jun-2018].

[102] "Key Value Database | NoSQL Key Value Database | Riak KV | Basho." [Online]. Available: http://basho.com/products/riak-kv/. [Accessed: 14-Jun-2018].

[103] "Redis." [Online]. Available: https://redis.io/. [Accessed: 14-Jun-2018].

[104] "Scalaris." [Online]. Available: http://scalaris.zib.de/. [Accessed: 14-Jun-2018].

[105] "Tokyo Cabinet: a modern implementation of DBM." [Online]. Available: http://fallabs.com/tokyocabinet/. [Accessed: 14-Jun-2018].

[106] "AWS | Amazon SimpleDB – Simple Database Service." [Online]. Available: https://aws.amazon.com/simpledb/. [Accessed: 14-Jun-2018].

[107] "Apache CouchDB." [Online]. Available: http://couchdb.apache.org/. [Accessed: 14-Jun-2018].

[108] “MongoDB for GIANT Ideas | MongoDB.” [Online]. Available: https://www.mongodb.com/. [Accessed: 14-Jun-2018].

[109] “Google Code Archive - Long-term storage for Google Code Project Hosting.” [Online]. Available: https://code.google.com/archive/p/terrastore/. [Accessed: 14-Jun-2018].

[110] “Apache HBase – Apache HBase™ Home.” [Online]. Available: https://hbase.apache.org/. [Accessed: 14-Jun-2018].

[111] “Home | Hypertable - Big Data. Big Performance.” [Online]. Available: http://www.hypertable.org/. [Accessed: 14-Jun-2018].

[112] “Apache Cassandra.” [Online]. Available: http://cassandra.apache.org/. [Accessed: 14-Jun-2018].

[113] H. Benfenatki, C. Ferreira Da Silva, G. Kemp, A. N. Benharkat, P. Ghodous, and Z. Maamar, “MADONA: a method for automated provisioning of cloud-based component-oriented business applications,” *Serv. Oriented Comput. Appl.*, vol. 11, no. 1, pp. 87–100, Mar. 2017.

[114] L. Zhang, C. Wu, Z. Li, C. Guo, M. Chen, and F. C. M. Lau, “Moving Big Data to The Cloud: An Online Cost-Minimizing Approach,” *IEEE J. Sel. Areas Commun.*, vol. 31, no. 12, pp. 2710–2721, Dec. 2013.

[115] R. R. Schaller, “Moore’s law: past, present and future,” *IEEE Spectr.*, vol. 34, no. 6, pp. 52–59, Jun. 1997.

[116] A. Freitas, E. Curry, A. Freitas, and B. E. Curry, “Big Data Curation 6.2 Key Insights for Big Data Curation,” 2016.

[117] G. Vargas-Solar, J.-L. Zechinelli-Martini, and J.-A. Espinosa-Oviedo, “Computing query sets for better exploring raw data collections,” in *Proceedings of the 13th International Workshop on Semantic and Social Media Adaptation and Personalization*, 2018.

[118] E. Arriaga-Varela, J. A. Espinosa-Oviedo, G. Vargas-Solar, and R. Dvila Pérez, “Supporting Real-Time Visual Analytics in Neuroscience,” *B. Abstr. 4th BSC Int. Dr. Symp.*, pp. 71–72, 2017.

[119] G. Li, J. Wang, Y. Zheng, and M. J. Franklin, “Crowdsourced Data Management : A Survey,” *Tkde*, 2015.

[120] J. L. De la Rosa, V. Torres-Padrosa, A. El-Fakdi, D. Gibovic, L. Maicher, and F. Miralles, “A survey of Blockchain Technologies for Open Innovation,” in *White Paper*, 2017, no. November, pp. 1–27.

[121] I.-C. Lin and T.-C. Liao, “A Survey of Blockchain Security Issues and Challenges,” *Int. J. Netw. Secur.*, vol. 1919, no. 55, pp. 653–65901, 2017.

[122] Z. Zheng, S. Xie, H.-N. Dai, and H. Wang, “Blockchain Challenges and Opportunities : A Survey Shaoan Xie Hong-Ning Dai Huaimin Wang,” *Int. J. Web Grid Serv.*, pp. 1–24, 2016.

APPENDIX

```
Distribution(dataItems, attr) → ditem1 : dataItem where
       ditem1.tab=set(Numbers)
       for each item in attr:
              ditem1.tab[item]= ditem1.tab[item]+1
minFunction(dataItems: Set[DataItems],attr:Attribut) → ditem1 : dataItem where
       for each item in dataItems:
              if ditem1==null or ditem>item[attr]:
                     ditem1=item[attr]
maxFunction(dataItems: Set[DataItems],attr:Attribut) → ditem1 : dataItem where
       for each item in attr:
              if ditem1==null or ditem<item[attr]:
                     ditem1=item[attr]
modeFunction(dataItems: Set[DataItems],attr:Attribut) → ditem1 : dataItem where
       tab=set(Numbers)
       for each item in attr:
              tab[item]= ditem1.tab[item]+1
for each value in tab:
if ditem1==null or ditem.count<tab[value]:
                     ditem1.value=value
                     ditem1.count=tab[value]
medianFunction(dataItems: Set[DataItems],attr:Attribut) → ditem1 : dataItem where
       dataItem = dataItems.sort()[dataItems.length/2]
meanfunction(dataItems: Set[DataItems],attr:Attribut) → ditem1 : int where
       count=0
sum=0
       for each item in DataItems:
              sum=sum+item
              count=count+1
       ditem=sum/coun
countFunction(dataItems: Set[DataItems],attr:Attribut) → ditem1 : dataItem where
ditem
       for each item in dataItems:
              ditem=sitem + item[attr].count
```

Listing 23: ComputeStatistics