Towards new methods for the integration and interrogation of geo-referenced sensor data applied to the environmental cloud for the benefit of agriculture (CEBA)

Thi Thu Trang Ngo

THÈSE PRÉSENTÉE

POUR OBTENIR LE GRADE DE

DOCTEUR DE

L'UNIVERSITÉ CLERMONT AUVERGNE

……..

ECOLE DOCTORALE DES SCIENCES POUR L'INGENIEUR

SPÉCIALITÉ : Informatique

Soutenue publiquement le 22/06/2023, par :

Thi Thu Trang NGO

Vers de nouvelles méthodes pour l'intégration et l'interrogation des données de capteurs géoréférencés appliquées au cloud environnemental au bénéfice de l'agriculture (CEBA)

Sous la direction de M. François PINET

M. David SARRAMIA et Mme. Myoung-Ah KANG

Devant le jury composé de :

M. David SARRAMIA	MCF, LPC, Université Clermont Auvergne	Co-encadrant de thèse
M. Didier DONSEZ	PR, LIG, Université Joseph Fourier - Grenoble 1	Rapporteur
M. Dino IENCO	DR, TETIS, INRAE Occitanie-Montpellier	Président
M. François PINET	DR, TSCF, INRAE, Clermont-Auvergne-Rhône-Alpes	Directeur de thèse
M. Jérôme DARMONT	PR, ERIC, Université Lyon 2	Rapporteur
Mme. Myoung-Ah KANG	MCF, LIMOS, Université Clermont Auvergne	Co-encadrant de thèse

Titre : Vers de nouvelles méthodes pour l'intégration et l'interrogation des données de capteurs géoréférencés appliquées au cloud environnemental au bénéfice de l'agriculture (CEBA)

Résumé court : Les capteurs ont révolutionné divers domaines, tels que l'agriculture et le suivi de l'environnement. Ils fonctionnent de manière autonome, collectent et transmettent des données en utilisant des technologies sans fil. Cependant, l'intégration des données de capteurs provenant de différents projets pose le défi de leur analyse et de la gestion des données en raison du débit. Cette thèse propose trois contributions principales. Premièrement, une méthode pour modéliser un entrepôt de données spatiales (SDW en anglais) en utilisant la suite ELK et une architecture pour intégrer en continu les sources de données de capteurs en les chargeant dans Elasticsearch. Deuxièmement, nous proposons un système basé sur des techniques de médiation pour l'analyse en temps réel de flux de données spatiales avec une intégration transparente. Troisièmement, nous proposons un modèle multidimensionnel générique qui cible les données IoT compatibles avec l'API SensorThings.

Mots clés : Données géoréférencées, intégration de données, cloud, surveillance de l'environnement, entrepôt de données, système de médiation, flux de données, ELK, Elasticsearch.

Title: Towards new methods for the integration and interrogation of geo-referenced sensor data applied to the environmental cloud for the benefit of agriculture (CEBA)

Short abstract: The use of sensors has transformed various fields, such as agriculture and the environmental monitoring. Sensors operate independently in remote locations and collect data with wireless technologies. However, the integration of sensor data from different projects poses the challenge of their analysis and data management due to throughput. This thesis proposes three main contributions. First, a method to model a spatial data warehouse (SDW) using the Elastic Stack (ELK) and an architecture, named IAT, as a streaming Extract, Transform, Load (ETL) process to integrate various sensor data sources and load them into Elasticsearch. Second, we propose a system based on mediation technique for the real-time analysis of spatial data streams in combination with static data, with seamless integration. Third, we propose a generic multidimensional model that targets IoT data using the SensorThings API standard.

Keywords: Geo-referenced Data, Data Integration, Cloud, Environmental Monitoring, Data Warehouse, Mediation System, Streaming Data, ELK Stack, Elasticsearch.

Towards new methods for the integration and interrogation of geo-referenced sensor data applied to the environmental cloud for the benefit of agriculture (CEBA)

Thi Thu Trang NGO

Department of Computer Science
Doctoral School of Sciences for Engineering
Clermont Auvergne University, France

Abstract

In recent years, the widespread use of sensors has revolutionized many sectors, particularly in agriculture and environmental applications, with the emergence of the Internet of Things (IoT). Equipped with batteries, sensors can work autonomously in remote locations, without the need for maintenance or repairs. They are typically deployed as clusters, where each sensor collecting data on its surroundings and communicating wirelessly with one or several gateways which are communicating to the cloud. The collected data is analyzed using various techniques and visualizations to extract insights and drive decisions. This entire process is known as the "sensor-to-decision chain". However, data analysis across heterogeneous sources is a complex task, especially when considering data from sensors. One of the difficulties is the interoperability issues due to different vendors in the market of sensors. The Open Geospatial Consortium (OGC) has defined several standards for accessing sensor data to overcome these issues. Additionally, stream processing presents numerous challenges due to the high rate at which sensor data is generated.

In this thesis, we first address the problem of integrating and analyzing geo-referenced data from multiple sources, we propose both a method and an architecture to represent and query a spatial data warehouse (SDW) model with the ELK (Elasticsearch, Logstash, Kibana) stack. We demonstrate how to implement a data warehouse of geo-referenced data in an ELK-based architecture, with the use of a component called IAT (Integration and Aggregation Tool) that operates like a streaming ETL to integrate different sensor data and load it into Elasticsearch. We illustrate the approach with two multidimensional models relevant to environmental sensor data and show the value of the system with some real-world user queries. Additionally, we evaluate the system with a benchmark dataset with respect to several aspects.

In the second part, we address the problem of big and streaming spatial data analysis, we propose a system based on mediation techniques for analyzing spatial stream in real-time with seamless integration. We also address the issue of integrating different data sources under a uniform schema for efficient analysis. We propose an interface and a customized

SQL grammar to express queries with streaming and spatial semantics. The proposed system allows for an administrator to configure the system by designing a mediated schema and defining the mappings between the mediated schema and the data sources. Users can express queries on the mediated schema in a dedicated SQL grammar, and the system rewrites the query into an Apache Spark application. The results are returned to the user continuously. Moreover, we implement into the mediation system, an optimizer that overtakes the query plans computed natively by Spark. Our experiments show that these optimizations improve up to one order of magnitude the queries execution time.

Finally, we address the problem of modeling IoT sensor data. We propose a generic multidimensional model for SensorThings API compatible data sources, which is based on UML profile. Furthermore, we model the ETL process and present our proposal through a case study.

Résumé

Ces dernières années, l'utilisation généralisée de capteurs a révolutionné de nombreux secteurs, notamment dans l'agriculture et les applications pour l'environnement, à la suite de l'émergence de l'Internet des objets (IdO). Équipés de leurs propres batteries, les capteurs peuvent fonctionner de manière autonome dans des endroits éloignés, sans nécessiter de maintenance ou de contrôle. Ils sont généralement déployés en clusters, chaque capteur collectant des données sur son environnement et communiquant sans fil avec une ou plusieurs passerelles communiquant elles-mêmes avec un cloud. Les données collectées sont analysées et visualisées à l'aide de diverses techniques afin d'en extraire des informations et prendre des décisions. L'ensemble de ce processus est connu sous le nom de "sensor-to-decision chain". Cependant, l'analyse de données provenant de sources hétérogènes est une tâche complexe, surtout lorsqu'il s'agit de données de capteurs. L'une des difficultés réside dans les problèmes d'interopérabilité dus aux différents fournisseurs de capteurs. L' « Open Geospatial Consortium » (OGC) a défini plusieurs normes d'accès aux données de capteurs pour surmonter ces problèmes. De plus, le traitement des flux de données présente de nombreux défis en raison du débit élevé auquel les données des capteurs sont générées.

Dans cette thèse, nous abordons d'abord l'analyse de données spatiales avec des systèmes NoSQL. Nous proposons à la fois une méthode et une architecture pour représenter et interroger un modèle d'entrepôt de données spatiales (SDW) avec la pile ELK (Elasticsearch, Logstash, Kibana). Nous démontrons comment mettre en œuvre un entrepôt de données géoréférencées dans une architecture basée sur ELK, avec l'utilisation d'un composant appelé IAT (Integration and Aggregation Tool) qui fonctionne comme un processus ETL (Extract-Transform-Load) en continu pour intégrer différentes données de capteurs et les charger dans Elasticsearch. Nous illustrons l'approche avec deux modèles multidimensionnels pertinents pour les données de capteurs environnementaux et montrons la valeur du système avec quelques requêtes réelles d'utilisateurs. En outre, nous évaluons le système avec un ensemble de données de référence par rapport à plusieurs aspects.

Ensuite, nous présentons notre proposition de système basé sur des techniques de médiation pour l'analyse de données spatiales statiques et temps réel avec une intégration transparente.

Nous abordons la question de l'intégration de différentes sources de données sous un schéma uniforme pour une analyse efficace. Nous proposons une interface et une grammaire SQL personnalisée pour exprimer des requêtes avec une sémantique pour le traitement des données spatiales et des flux. Le système proposé permet à un administrateur de configurer le système en concevant un schéma global et en définissant les correspondances entre le schéma global et les sources de données. Les utilisateurs peuvent exprimer des requêtes sur le schéma global dans une grammaire SQL dédiée, et le système réécrit la requête en une application Apache Spark. Le résultat est renvoyé à l'utilisateur en continu. De plus, nous implémentons dans le système de médiation, un optimiseur qui surpasse les plans de requêtes calculés nativement par Spark. Nos expériences montrent que ces optimisations améliorent le temps d'exécution des requêtes.

Enfin, nous abordons la modélisation des données des capteurs IdO. Nous proposons un modèle multidimensionnel générique pour les sources de données compatibles avec l'API SensorThings, qui est basé sur le profil UML. En outre, nous modélisons le processus ETL et présentons notre proposition au moyen d'une étude de cas.

Acknowledgements

This PhD journey has been enriching and challenging, filled with ups and downs, joy and tears, and most importantly invaluable lessons. Fortunately, I have been surrounded by a supportive community of individuals who have given me the strength to navigate through the difficult times. It is with great pleasure that I express my gratitude to all of them for their positive impact during this period.

First and foremost, I would like to express my sincere appreciation to my supervisors Prof. François Pinet, Prof. David Sarramia, and Prof. Myoung-Ah Kang for granting me the opportunity to participate in this project. I am grateful for their trust, technical expertise, and invaluable guidance in the fields of scientific research, environment, and agriculture domain over three years. Thank you for your support and guidance through the period of research and writing of this thesis.

I would also like to express my gratitude to all the jury members of my defense. Many thanks to Mr. Dino IENCO, Professor at TETIS INRAE Occitanie-Montpellier for accepting to evaluate my work. I am equally grateful to Mr. Jérôme Darmont, Professor at University Lumiere Lyon 2 and Mr. Didier Donsez, Professor at University Grenoble Alpes, for their willingness read and review of my manuscript. I thank each of them in advance for their guidance and valuable feedback.

My appreciation extends to the members of the “comite de these” as well. I am deeply grateful to Mr. Chafik Samir, Professor at University Clermont Auvergne and Mr. Engelbert Mephu Nguifo, Professor at University Clermont Auvergne and Mr. Didier Donsez, Professor at University Grenoble Alpes and Mr. David Hill, Professor at University Clermont Auvergne and Mr. Cyril Ray, Professor at University of Rennes, for their commitment to reviewing and overseeing my thesis. I thank all of them in advance for their guidance and invaluable feedback.

I would like to express my sincere gratitude to colleagues and professors at the LIMOS laboratory. Special mention goes to Mr. Chafik Samir and Mr. Nguyen Viet Hung, as well as

the PhD students Diego, Pedro, Tam, Mateus, Donald, Roxane, Hélène, Weixuan, Hieu, Loan, Thao, Hoang, Valentin, postdoc Anis, and many others who made this journey exceptional. I also extend my thanks to the administrative team at LIMOS, Mrs. Béatrice Bourdieu, Mrs. Martine Caccioppoli, and the technical team at LIMOS, Mr. William Guyot Lenat, for their assistance and support, thank you all.

I would like to acknowledge the French government IDEX-ISITE initiative 16-IDEX-0001 (CAP 20-25) for providing FEDER funding (European Regional Development Fund).

Additionally, I would like to express my gratitude to the ConnecSenS project and the CEBA project for sharing their research materials and the agriculture and environment dataset. Their support has greatly contributed to the evaluation of my contributions.

I would like to thank, particularly Mr. Sofian Maabout, Professor at University Bordeaux and Mr. Fabien Baldacci, Professor at University Bordeaux for their invaluable guidance during my master's internship, which marked the early stages of my research journey. I am sincerely grateful to all my dearest friends: Carole Blanc, Marie Beurton-Aimar, Vu Manh Tu, Le Van Linh, Phuong Thao, Karim, Nabil, Mohamed, Abiodun, Charles, Nhung, Minh Huong, Lam, and many others for their unwavering support and true friendship and love, and their presence during challenging times.

Finally, I would like to express my deepest gratitude to my parents, my sisters, and my beloved for their unwavering encouragement and support throughout this journey.

Contents

List of Figures

List of Tables

List of Abbreviations

API	Application Programming Interface
BSD	Big Spatial Data
BSVD	Big Spatial Vector Data
CDC	Change Data Capture
CEBA	Environmental Cloud for the Benefit of Agriculture
DAG	Directed Acyclic Graph
DBMS	Database Management System
DFS	Distributed File System
DW	Data Warehouse
ELK stack	Elasticsearch - Logstash - Kibana stack
ER	Entity Relationship
ES	Elasticsearch
ETL	Extract-Transform-Load
GAV	Global as View
GeoJSON	Geographic JSON
GIS	Geographic Information System
GISc	Geographic Information Science
GML	Geography Markup Language
HDFS	Hadoop Distributed File System
IAT	Integration and Aggregation Tool
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
JSON	JavaScript Object Notation
LAV	Local as View
LoRaWAN	Long Range Wide-Area Network
LPWAN	Low Power Wide Area Network Technology
MQTT	Message Queuing Telemetry Transport
OGC	Open Geospatial Consortium
OLAP	Online Analytical Processing
RDD	Resilient Distributed Dataset

SDW	Spatial Data Warehouse
SOLAP	Spatial OLAP
SOS	Sensor Observation Service
SPE	Streaming Processing Engine
SPS	Sensor Planning Service
STA	SensorThings API
STADW	SensorThings API Data Warehouse
UML	Unified Modeling Language
WKT	Well-Known Text
XML	Extensible Markup Language

Publications' Notes

Ngo, Thi Thu Trang, David Sarramia, Myoung-Ah Kang, and François Pinet. "An Analytical Tool for Georeferenced Sensor Data based on ELK Stack." In 7th International Conference on Geographical Information Systems Theory, Applications and Management, SCITEPRESS-Science and Technology Publications (Ngo 2021).

Ngo, Thi Thu Trang, David Sarramia, Myoung-Ah Kang, and François Pinet. "A New Approach Based on ELK Stack for the Analysis and Visualisation of Geo-referenced Sensor Data." in the Springer Nature Computer Science journal (Ngo 2023).

Ngo, Thi Thu Trang, François Pinet, David Sarramia, and Myoung-Ah Kang. "A Mediation System for Continuous Spatial Queries on a Unified Schema Using Apache Spark". in revision for the Big Earth Data journal (Taylor & Francis).

Introduction

In recent years, sensors have become widely used for many sectors especially in the context of agriculture and environment Internet of Things (IoT). Billions of sensors are connected to the Internet everyday creating a novel concept called IoT (Da Xu 2014). Sensors are devices that measure physical quantities such as temperature, humidity, pressure, and stream the measures to a central location for processing or storage, e.g., server, database. They are usually equipped with batteries that allow them to work independently in remote places, e.g., space, forest, sea without need of maintenance or repair. Most of the time, sensors are deployed as a cluster of nodes, where each is collecting data about its surroundings and are all communicating through a wireless network. Then, sensor data is analyzed using numerous analytical techniques and with the help of visualization to extract insights and derive decisions. This process pipeline has been named "sensor to decision chain" (Moßgraber 2018).

Analyzing sensor data is challenging due to the difficulties in integrating data from diverse sources. Environmental monitoring, for example, relies on data from multiple sources, including remote sensing, weather stations, or geographic information systems. Integrating these diverse data sources can be complex, as the data can vary in format, frequency, and quality. Poor data integration can result in incomplete or inconsistent analysis results, making it difficult to extract meaningful insights. To address these challenges, researchers have proposed various approaches, such as developing data lakes to integrate and store the data in multiple formats. The Environmental Cloud for the Benefits of Agriculture project (CEBA) presented in (Breton 2019) and (Sarramia 2022), is an example of such systems. The CEBA architecture comprises a data lake and a set of tools and processes for the storage, retrieval, and analysis of geo-referenced sensor data. As depicted in Figure 1, CEBA is composed of various components, including Ingestion, Storage, Indexing, Cataloging, and a User Interface for data sharing. The ELK stack (ELK stack 2022), including Beats and Logstash for data ingestion and ElasticSearch for indexing, is utilized for ingestion and indexing processes. Long-term data storage is facilitated by the relational database PostgreSQL,

while cataloging is achieved through the use of GeoNetwork. Data can be accessed and searched through different endpoints, including APIs and message queue systems. The main data flow in CEBA involves the ingestion of collected sensor data through Logstash or Beats shippers.

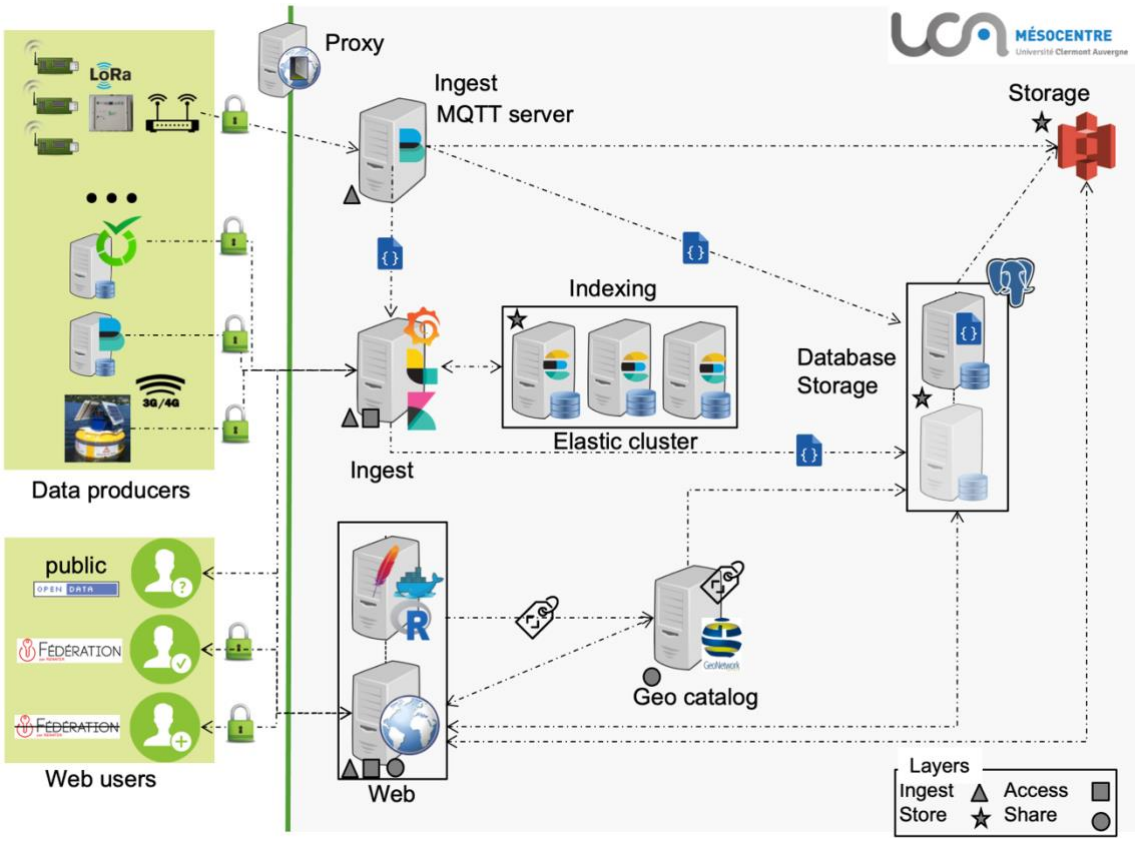


Figure 1 CEBA architecture (Sarramia 2022).

For example, CEBA stores data collected by sensor networks managed by INRAE in the experimental site of Montoldre in Allier. Figure 2 shows a weather station and a wireless communication node to transmit the data using LoRaWAN (Long Range Wide-Area Network) (Augustin 2016). The collected data are measurements such as air temperature, soil moisture, rainfall. They help farmers to monitor the crop state and manage agricultural operations. The detection of plant water stress can require irrigation, while too much water can also lead to plant diseases, depending on other factors (temperature, phenological stages of the plant, etc.). Consequently, all the collected data are important to identify plant problems as soon as possible and to carry out the appropriate agricultural actions.

Figure 2 A weather station and a wireless communication node collecting data in an agricultural plot of INRAE Montoldre.

The purpose of this thesis is to address the issue of data analysis across heterogeneous sources and show a concrete application in the development of CEBA. The process of conducting a thorough analysis of data obtained from various sources can often prove to be a complex task, owing to several contributing factors such as heterogeneity within the data sets. The complexity is further increased when considering sensor data. Next, we present some of these difficulties.

When conducting sensor data analysis, it is often necessary to integrate data from different sensor devices. However, the heterogeneity of sensors and their data management practices can cause interoperability issues, especially among devices from different vendors. For instance, temperature sensors from different providers may measure temperature using either Celsius or Fahrenheit. Also, sensors may support date and time in different formats. To overcome the interoperability issues due to different vendors in the market of sensors, Open Geospatial Consortium (OGC) (OGC 2022), a non-benefit organization, has defined several standards for accessing sensor data. One such standard is the SensorThings API (Liang 2016), which is a comprehensive data model and set of requirements for implementing an API system to access sensor data.

This standard is novel in its simplicity of data modeling, facility of implementation through REST, and support for event data. The analysis of sensor data presents numerous challenges, particularly regarding the streaming nature of such data. Due to the high rate at which sensor data is generated, conventional methods of data processing often prove insufficient. New systems are designed to achieve *exactly-once guarantee* processing with maximum throughput (Isah 2019).

Additionally, stream processing differs fundamentally with batch processing. Stream processing uses the concepts of *window* and *watermark* (Isah 2019). *Window* serve to limit the number of records processed by a query, either in terms of time intervals or a specific quantity. In the context of time interval windows, *watermark* serve to define the acceptable delay for records. These operations play a crucial role in the efficient and effective processing of streaming data.

In the field of data analytics (M. M. Alam 2021), which involves dealing with heterogeneous and numerous data sources, different techniques exist such as data warehousing or mediation. The primary difference between these two techniques lies in their approach to data storage. Let us consider a set of data sources $S_1 \dots S_n$, and a data model G specifically designed for analytics purposes. In data warehousing, a dedicated datastore stores data represented in this analytics data model. Data from data sources $S_1 \dots S_n$ is migrated to the analytics data model following a set of rules and transformations, collectively known as Extract-Transform-Load (ETL). User queries are written in accordance with the analytics data model G, and they are executed on the data warehouse engine. Meanwhile, for mediation, data remains in its local storage. User query q is rewritten into $q_1 \dots q_n$ following a set of mapping rules. Each query q_i is executed on the data source S_i. Answers to the queries are then integrated in the mediation system. To summarize, data warehousing involves transforming the data and loading into the analytics data storage, whereas mediation involves transforming the queries to match the local storage schema.

Data warehousing is a prevalent technique utilized in the business domain, as it provides fast query answering. Nonetheless, due to the need for the Extract-Transform-Load (ETL) process, analyzes are predominantly carried out on historical data. Recent techniques such as Streaming ETL and Change Data Capture (CDC) have been

proposed to circumvent this limitation. On the other hand, mediation is a technique that allows for quicker adaptation to schema changes and enables querying of data as soon as it is available. However, this technique comes at a high cost of data integration and overburden query answering.

In this dissertation, we address the challenges of real-time collecting and analyzing geo-referenced environmental data from heterogeneous sources. We focus on two popular techniques in data integration for our proposals, i.e., data warehouse and mediation. The paragraphs below present more precisely our contributions.

A method for designing data warehouses with ELK stack for near real time analysis and visualization of geo-referenced sensor data.

Data warehouses and Online Analytical tools (OLAP) are efficient solutions to analyze sensor information, as they allow users to visualize, aggregate, and summarize these large sets of geo-referenced data at different scales. Data warehouses are often based on relational data models (Jarke 2002, Inmon 2005, Pinet 2010), but this type of model is not the most efficient for real-time sensor streams. (Bicevska 2017) discussed the NoSQL-based data warehouse solutions and provided some encouraging points. However, they focused on the early stage of this approach and the lack of reporting tools compatible with NoSQL systems.

The ELK stack (ELK stack 2022) (i.e., Elasticsearch, Logstash and Kibana) can be a useful and flexible solution to integrate and store these large heterogeneous data. In this manuscript, we propose a method and a dedicated architecture to represent and query a spatial data warehouse (SDW) model with ELK stack. The purpose of our approach is to combine both advantages of ELK and data warehouse technology. Thus, the manuscript shows how to implement a data warehouse of geo-referenced data in an ELK-based architecture.

We illustrate the approach in two multidimensional models relevant to environmental sensor data. We detail their implementation in Elasticsearch (also called ES), as well as their respective queries. Moreover, we implement and present a component called IAT (Integration and Aggregation Tool) that operates like a streaming ETL (Sabtu 2017) driven by a user configuration, to integrate different sensor data and load it into

Elasticsearch. We show the value of the system with some real-world user queries and evaluate it with a benchmark dataset with respect to several aspects.

A mediation system and architecture for real-time integration and analysis of geo-referenced sensor data.

Big data management systems offer high efficiency in processing data and enabling analytics. Tools such as Apache Spark (Zaharia 2012) and Apache Flink (Carbone 2015), have become the reference tools to process high volumes of data. Moreover, they were also enriched with geospatial capabilities (Pandey 2018, Shaikh 2020).

However, these tools lack of support for real-time integrating different data sources under a uniformed schema. Consider an example in which a physicist wants to analyze IoT sensor data collected all around Europe. The physicist is interested in the quality of air near buildings of industrial areas. Then, he/she must join data from different data sources for his/her queries, e.g., streams of sensors, city buildings, industrial zones. Suppose the physicist uses Apache Sedona (Yu 2015), (i) he/she must handle the heterogeneity of data sources, e.g., documents, streams, relational database, and (ii) he/she must manage each data source schema. The complexity of handling these issues increases considerably with the number of data sources. Moreover, this task is not straightforward for users without knowledge in big and streaming data integration.

In our work, we propose a system and architecture that is based on mediation technique to analyze spatial stream-static data in real time with seamless integration. To that purpose, we propose a customized SQL grammar to express queries with streaming and spatial semantics. Given a set of local data sources and an application requirement: first, an administrator configures the system, i.e., he/she designs a mediated schema and defines the mappings between the mediated schema and the data sources. Second, users express queries on the mediated schema using a dedicated SQL grammar and our system rewrites the query into an Apache Spark application. Finally, the Apache Spark application is submitted to an Apache Spark cluster and the result is returned to the user continuously.

A generic data model for data warehousing IoT data and ETL processes.

OGC proposed SensorThings (Liang 2016) as a new data model to address interoperability issues within the Internet of Things (IoT) systems. The model describes the relationship of different entities: Thing, Sensor, Datastream, ObservedProperty, Observation, and FeatureOfInterest. Furthermore, the standard includes an API for CRUD (Create, Read, Update, Delete) operations. Despite its comprehensive design, SensorThings is intended for transactional purposes rather than analytics. Recognizing this limitation, this work presents a generic data model for warehousing SensorThings IoT data.

Manuscript Organization

We first recall the literature relevant to this thesis in Chapter 1. We recall definitions related to spatial data and summarize work relative to standards for sensor data, IoT data modeling and data integration.

In Chapter 2, we present our proposal of a method and architecture to represent and query a spatial data warehouse (SDW) model with the ELK stack. We demonstrate how to implement a data warehouse of geo-referenced data in an ELK-based architecture, with the use of a component called IAT (Integration and Aggregation Tool) that operates like a streaming ETL to integrate different sensor data and load it into Elasticsearch. We have published the architecture and data integration principle in (Ngo 2021). Experiments and evaluation were published in the Springer Nature Computer Science journal (Ngo 2023).

In Chapter 3, we present our proposal of a system based on mediation techniques for analyzing spatial stream-static data in real-time with seamless integration. This work is in revision in the Big Earth Data journal.

In Chapter 4, we propose a work for a generic multidimensional model for analyzing sensor data from SensorThings compatible data sources. Finally, we conclude the manuscript by providing perspective for future work.

CHAPTER 1 BACKGROUND AND RELATED WORK

In this chapter, we give a general overview of work related to this dissertation. Our research work is related to several research topics. They are mainly, (i) geo-referenced sensor data, (ii) streaming data processing, and (iii) spatial data analytics. We will provide concepts and state of the art of each of the above topics.

1.1 Concepts Related to Spatial Data

In this section, we present and describe the different types of geo-referenced data. Then, we describe the common data formats, the vector data type presentation, and spatial queries.

1.1.1 Spatial Data Definition

The origins of the term *spatial data* refer to the field of geographic information systems (GIS) and the study of geographic information science (GISc). The first academic papers that express the definition of spatial data are in the 1970s and 1980s, as GIS technology and the GISc field were introduced. The paper of (Tomlinson 1974) is one of the early academic research projects that described the definition of spatial data. In this document, the author determined spatial data as "*location-specific data … related to the Earth's surface*".

The encyclopedia of GIS (S. a. Shekhar 2007) determined, "*spatial data is data related to a location. Some examples include the population of a city, the type of soil in a region, and data from remote-sensing satellites. In the first example, the city could be considered as a location and the population is the data or a feature. In the second example, the region is a collection of locations and the type of soil is the feature. Note that a location may have one or more features. For example, it may be useful to*

represent both the population and the average age group of a city". In conclusion, the term spatial data refers to data that is referenced and applies to a specific location on the Earth.

1.1.2 Spatio-temporal Data and Geo-referenced Data

Spatio-temporal data, as defined by (Roshannejad 1995) represents a subcategory of spatial data that include the integration of both spatial and temporal information. Specifically, spatial data refers to the physical location of an object or phenomenon, while temporal data refers to the progression of time. The integration of space and time allows the analysis of the relationship and interactions between the physical location and the movement of time of a particular entity. This wide range of subjects, including data, processes, and events, is extensively employed within various fields, such as computer science, physics, and geography.

Geo-referenced data, as a subcategory of spatial data, is defined in (Roshannejad 1995) as "*a phenomenon that is extended in three directions: spatial extent, temporal extent, and aspatial extent*". Figure 3 illustrates the main components of a geo-referenced object, which includes spatial, temporal, and aspatial (also known as non-spatio-temporal) elements. Among these components, the temporal element holds a special role as it influences the other components. An example of geo-referenced data is the temperature of water at the North Pole in December 2020, which illustrates the intersection of spatial, temporal, and aspatial components.

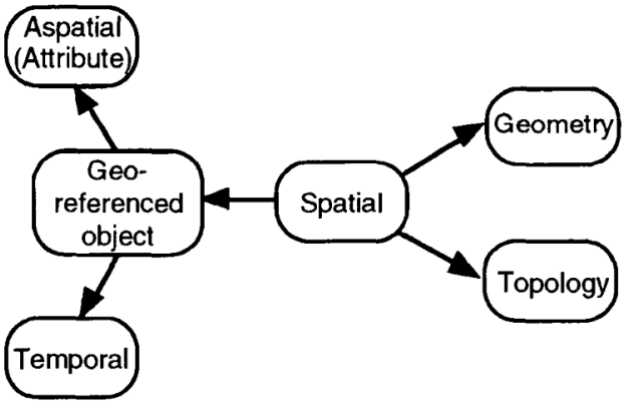


Figure 3 Components of Geo-referenced data (Roshannejad 1995).

1.1.3 Spatial Geometry Objects

Geometry objects are one of the main components of spatial data. According to (Guting 1994): "*the fundamental abstractions are point, line, and region. A point represents (the geometric aspect of) an object for which only its location in space, but not its extent, is relevant. For example, a city may be modeled as a point in a model describing a large geographic area (a large scale map). A line (in this context always understood to mean a curve in space, usually represented by a polyline, a sequence of line segments) is the basic abstraction for facilities for moving through space, or connections in space (e.g., roads, rivers, cables for phone, electricity). A region is the abstraction for something having an extent in 2-D space (e.g., a country, a lake, or a national park). A region may have holes and may also consist of several disjoint pieces*". A region is also called a polygon. Figure 4 presents the fundamental geometries point, line, and polygon.

Figure 4 Three fundamental geometries: point, line, polygon (Guting 1994).

1.1.4 How to Describe Spatial Data in Database Systems?

In general, there are three primary ways to describe spatial data in a traditional spatial database system (S. Shekhar 2012):

- Graph data: often used for representing primarily network-based information, such as transportation networks. For example, a map of the transportation network in Paris can be represented using graph data.
- Raster data: geographical images consisting of a grid of cells. Each cell within a raster has a location and set of attributes (e.g., satellite images providing the remote sensing data of the Earth).

- Vector data: represents the real world features using spatial objects: points, lines, polygons. A point can be represented as a pair of latitude and longitude in 2D or a triad of latitude, longitude and altitude in 3D. Other shapes, such as a lake shape, or a country area are represented as a series of points.

In our work, we consider the vector data in 2D presentation. In the next section we detail the formats used to represent and store vector data.

1.1.5 Vector Data Formats

In this section, we present an overview of the three most popular open standards for spatial vector data. These include the Well-Known Text (WKT) format, the Geography Markup Language (GML) format, and the GeoJSON format. These standards are designed for expressing basic geographical features, in conjunction with their associated non-spatial attributes. Table 1 provides a comparison of these three data formats by presenting the representation of the same object in each.

Well-Known Text (WKT) format (Lott 2015) is a text-based representation of vector geometry objects that is widely used in the field of Geographical Information Systems (GIS) and spatial data management. It has been an official OGC (Open Geospatial Consortium) standard since 2015 and is currently considered as the ISO 19111-3:2019 standard (ISO 2019). WKT is a simple, human-readable representation of vector data, which can be easily understood and edited by both humans and machines. It is commonly used for data storage, data exchange, and data visualization in GIS applications. WKT is also supported by many GIS software and libraries, making it a versatile and widely supported format for vector data.

Geography Markup Language (GML) is a widely adopted open standard for encoding and exchanging geospatial information. It is developed and maintained by the Open Geospatial Consortium (OGC) and is expressed in XML. The GML specification, as outlined in the literature (S. a. Cox 2003), provides a common schema language and predefined properties for describing various types of geospatial features, such as points,

polygons, and observations, as well as coordinate reference systems, units of measure, and other metadata. The format is designed to be extensible, allowing for the description of a wide range of feature types, their properties, and relationships. Additionally, GML has the capability to integrate a variety of forms of vector spatial data, as well as sensor and other non-spatial data, making it a versatile and powerful format for encoding and exchanging geospatial information.

GeoJSON is a lightweight format for encoding geospatial data that is based on JavaScript Object Notation (JSON). It was published as a standard by the Internet Engineering Task Force (IETF) as RFC 7946 (Butler 2016). GeoJSON is designed to express data about geographical features, their properties, and their spatial extents. It supports a variety of geospatial feature types, including Points, Line Strings, Polygons, and multipart collections of these types. As such, it can be used to represent a wide range of geographical features, including addresses, locations, streets, and more.

Data format	**Example**

GeoJSON	{
	"type": "Feature",
	"properties": {
	"name":"LIMOS"},
	"geometry": {
	"type": "Polygon",
	"coordinates": [
	[
	[3.111100, 45.759510],
	[3.110043, 45.759188],
	[3.111142, 45.758361],
	[3.111786, 45.758563],
	[3.111100, 45.759510]
	]
	]
	}
	}

GML	<ogr:FeatureCollection> <gml:boundedBy> <gml:Box> <gml:coord><gml:X>3.110043</gml:X><gml:Y>45.758361</gml:Y></gml:coord>

	<gml:coord><gml:X>3.111786</gml:X><gml:Y>45.75951</gml:Y></gml:coord> </gml:Box> </gml:boundedBy> <gml:featureMember> <ogr:geometryProperty><gml:Polygon srsName="EPSG:4326"><gml:outerBoundaryIs><gml:LinearRing><gml:coordinates>3.1111,45.75951 3.110043,45.759188 3.111142,45.758361 3.111786,45.758563 3.1111,45.75951</gml:coordinates></gml:LinearRing></gml:outerBoundaryIs></gml:Polygon></ogr:geometryProperty> <ogr:name>LIMOS</ogr:name> </gml:featureMember> </org:FeatureCollection>
WKT	POLYGON ((3.1111 45.75951, 3.110043 45.759188, 3.111142 45.758361, 3.111786 45.758563, 3.1111 45.75951))

Table 1 Example of vector data in WKT, GML and GeoJSON formats.

1.1.6 Spatial Query

There are three fundamental spatial data queries and a large range of queries are made of these three (Pandey 2018):

- Spatial range query is to return all objects *s* from a set of geometry objects *S* that are inside the range *R* (e.g., return all museums within 10 km from Eiffel tower).
- Spatial join query is to consider at least two datasets of spatial data *R* and *S*, and apply a join condition (e.g., intersect, contains, within) and return set of all pairs (*r*,*s*) satisfying this condition (e.g., from two datasets restaurant and cinema, return the cinemas that are in the same neighborhood of an Italian restaurant).
- K-Nearest Neighbors query (also called KNN query) is to take a set of objects *S*, a query point *p*, and a number $k >= 1$ as input, and find a subset of *S* of size k that are the nearest to *p* (e.g., return 5 nearest by restaurant).

Additionally, (Clementini 1996) defined topological operations Equals, Disjoint, Intersects, Touches, Crosses, Within, Contains, and Overlaps as the Dimensionally Extended Nine-Intersection Model (DE-9IM). Then, DE-9IM was adopted by OGC. Later, it became OGC-compliant for join predicates and (M. M. Alam 2018) implemented it in two most popular spatial processing systems SpatialHadoop (Eldawy

2015) and GeoSpark (Yu 2015). We recall in Table 2 the OGC-compliant join predicates originally from the DE-9IM model and other spatial analysis functions.

Predicates/ Functions	**Operation**	**Description**
Topological Relations (all pair joins)		
Equals	Polygon equals polygon	Find the polygons that are spatially equal to other polygons.
Equals	Point equals point	Find the points that are spatially equal to other points.
Disjoint	Polygon disjoint polygon	Find the polygons that are spatially disjoint from other polygons.
Intersects	Line intersects polygon	Find the lines in edges merge table that intersect polygons.
Intersects	Point intersects polygon	Find the points in point merge table that intersect polygons.
Intersects	Point intersects line	Find the points in point merge table that intersect lines.
Touches	Polygon touches polygon	Find the polygons that touch polygons.
Touches	Line touches polygon	Find the lines in edges merge table that touch polygons.
Crosses	Line crosses line	Find first 5 lines that crosses other lines.
Crosses	Line crosses polygon	Find the lines in edges merge table that cross polygons.
Overlaps	Polygon overlaps polygon	Find the polygons that overlap other polygons.
Within	Polygon within polygon	Find the polygons that are within other polygons.
Within	Line within polygon	Find the lines in edges merge table that are inside the polygons.
Within	Point within polygon	Find the points in pointlm merge table that are inside the polygons.
Contains	Polygon contains polygon	Find the polygons that contain other polygons.
Spatial analysis		
Distance	Distance search	Find all polygons in arealm merge table that are within 1000 distance units from a given point.
Within	Bounding box search	Find all lines in edges merge table that are inside the bounding box of a given specification.
Dimension	Dimension of polygons	Find the dimension of all polygons.
Envelope	Envelope of lines	Find the envelopes of the first 1000 lines.
Length	Longest line	Find the longest line.
Area	Largest area	Find the largest polygon.
Length	Total line length	Determine the total length of all lines.
Area	Total area	Determine the total area of all polygons.
Buffer	Buffer of polygons	Construct the buffer regions around one mile radius of all polygons.
ConvexHull	Convex hull of polygons	Construct the convex hulls of all polygons.

Table 2 OGC-compliant join predicates and other spatial analysis functions (Ray 2011).

1.2 Sensor Data Access Standards

In this section, we provide an overview of the OGC sensor data access standards. After an introduction on environmental sensor networks, we will also compare the two most widely utilized OGC data access standards in the context of modeling and exchanging data from sensors and Internet of Things.

1.2.1 Environmental Sensors and Sensor Network

Environmental sensors are devices that are used to measure various physical and chemical parameters of the environment, such as temperature, humidity, air quality, light, sound, and other variables. These sensors can be used in a variety of applications, such as weather forecasting, air quality monitoring, agriculture, and building automation. They can be integrated into a range of devices such as smartphones, smart home devices, and industrial control systems. The data collected from these sensors can be very heterogeneous, meaning that it can have different formats, semantics, structures, and levels of complexity. This can make it challenging to process and analyze the data, as well as to integrate it with other data sources. Data may be collected at different intervals or at different resolutions, which can make it difficult to compare or analyze the data. This can lead to further heterogeneity in the data, as different organizations may use different sensors, protocols, and methods for collecting and processing data. To overcome these challenges, techniques such as data standardization, data cleaning, and data integration can be used to make environmental sensor data more usable and actionable. Many organizations developed standards for sensor data access and interoperability.

Wireless communication systems are used for outdoor applications like in agriculture. For example, LoRaWAN (Long Range Wide-Area Network) (Augustin 2016) is a low power wide area network technology (LPWAN) that has been designed specifically for use in IoT sensor environments. This technology is capable of transmitting small messages of data over long distances. LoRaWAN has been extensively studied in the literature. For example, (Haxhibeqiri 2018) highlights the advantages of LoRaWAN

over other LPWANs (e.g., SigFox (Sigfox 2023), NB-IoT (Wang 2017)), specifically in terms of lower deployment costs and the ability to support private networks. Additionally, LoRaWAN networks are known for their low power consumption and simple deployment which helps to reduce the cost and complexity of setting up and maintaining a network. (Terray 2020) leverages the use of LoRaWAN network in terms of private networks compared with public networks in the hard environment such as volcano monitoring. They highlighted that private network infrastructures can be suitable to meet the specific needs and limitations of environment sensors which are collecting data in hard environment conditions, whereas public networks are managed independently by private companies and are not easily customizable. Furthermore, they also highlight that investments in hardware and maintenance for a private network are more cost-effective in the long-term and for a high number of connected sensors. Additionally, the LoRaWAN network has the ability to handle the issue of message loss in the communication between the sensor node and either the gateway or the server. Figure 5 depicts LoRaWAN architecture for connections between sensor devices and servers.

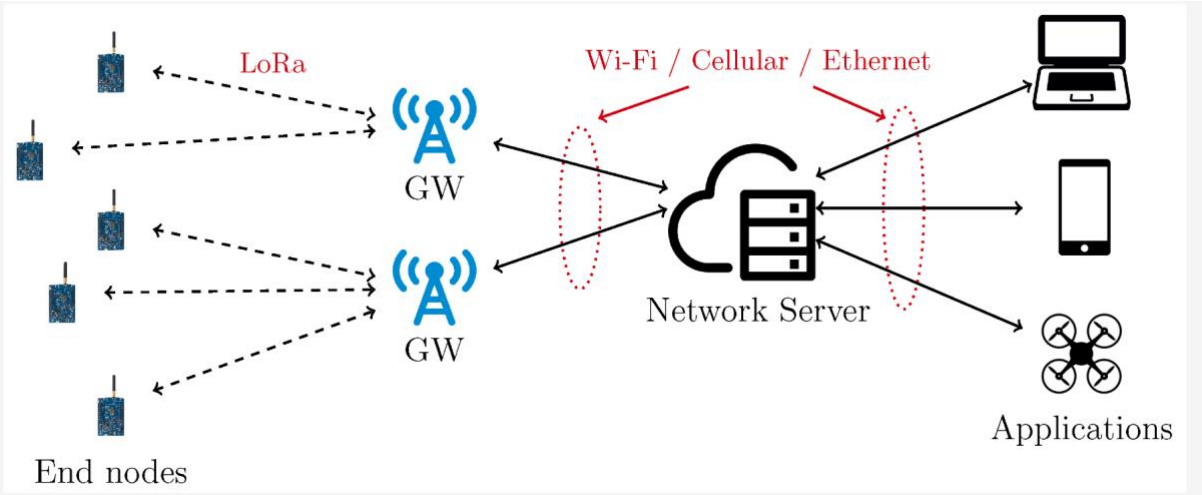


Figure 5 An example of LoRaWAN architecture (Codeluppi 2020).

1.2.2 OGC Data Access Standards

1.2.2.1 OGC

The Open Geospatial Consortium (OGC), established in 1994, is an international voluntary consensus standards organization that develops and implements access standards for geospatial data (OGC 2022). With over 500 voluntary consensus members and more than 40 standard organizations, the OGC provides freely available standards that are widely adopted in popular projects such as the Global Earth Observation System of Systems (GEOSS) (Fritz 2008), and the European INSPIRE Directive (Bartha 2011). These standards are open standards for geospatial data, with no license fees required, and can be grouped into five categories, which are data encoding standards, data access standards, processing standards, visualization standards, and metadata and catalog services standards.

- Data encoding standards, the OGC provides such as the Geography Markup Language (GML) standard (S. a. Cox 2003), which establishes a set of rules for structuring format for spatial data.
- Data access standards, such as the Sensor Observation Services (SOS) (Na 2007) and SensorThings API (STA) (Liang 2016), provide access to geospatial data via a network, allowing for actions such as GET and POST.
- Processing standards, such as the Web Processing Service (WPS) (WPS 2007), provide the ability to monitor and control operations.
- Visualization standards, such as the Web Map Service (WMS) (WMS 2006), provide a set of interfaces for requesting map images via the internet.
- Finally, metadata and catalog services standards, such as the Catalogue Services for the Web (CSW) (CSW 2010), support the search and publication of descriptions for data, services, and related objects.

In the context of sensor data modeling, the SOS and STA standards are particularly noteworthy for sensor network, as they provide well-defined interfaces for accessing measurements and observations collected from a wide variety of sensors. These

standards will be further examined in subsequent sections to demonstrate their modeling approaches.

1.2.2.2 SOS

The Sensor Observation Service (SOS) is an official OGC standard since 2007, designed to provide a standardized method for accessing and querying sensor observations and metadata over the internet (Na 2007). This standard defines a set of web service interfaces that enable clients to discover and retrieve sensor data, metadata, and related information, such as sensor locations, capabilities, and quality information. Additionally, the SOS standard specifies an encoding format for sensor observations and metadata, based on the ISO/OGC Observation and Measurement (O&M) model (S. Cox 2007) and the OGC SensorML model (Botts 2007), which are commonly used in the sensor data community. The SOS standard provides a flexible and extensible framework for integrating various types of sensors, platforms, and networks, thereby supporting the interoperability of sensor data across different domains and applications. This standard has been widely adopted in various domains such as environmental monitoring, meteorology, and civil protection (Yang 2010, Fazio 2015).

The SOS standard is composed of three main components: Core, Enhanced, and Transactional. The Enhanced and Transactional part primarily provide supplementary methods to support the Core, such as the RegisterSensor, InsertObservation, and GetFeatureOfInterest methods. The core component of SOS is the Observation. This standard serves as a framework for sensor data modeling, utilizing the SensorML model for metadata descriptions, and relying on the O&M model for the measurements. The SensorML model includes inputs, outputs, and parameters for all processes, making it a comprehensive representation of sensor systems. The O&M model is designed to describe sensor observations and measurements with the core class diagram depicted in Figure 6. The OM_Observation class is central to this model and comprises five attributes (S. Cox 2007):

- phenomenonTime, which is mandatory, is described by TM_Object and represents the time at which the result applies to the property of the feature-of-interest.
- resultTime, also mandatory, is described by TM_Instant and indicates the completion time of the entire observation.
- validTime, although optional, is described by TM_Period and specifies the time period during which the result is intended to be used.
- resultQuality, also optional, is described by DQ_Element (ISO 19115-1:2014 2014) and characterizes the quality of the result.
- parameter, which is optional, is described by NameValue, is used to specify any arbitrary event-specific parameters.

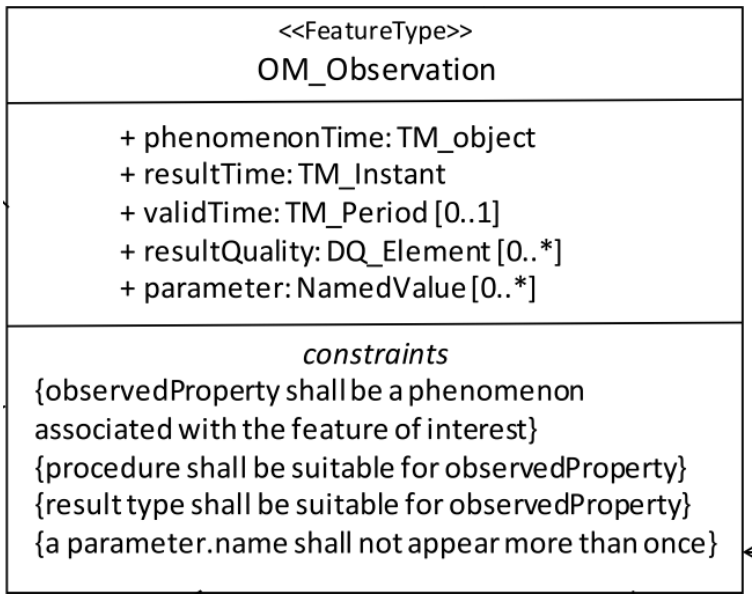


Figure 6 Core class diagram of O&M model (S. Cox 2007).

1.2.2.3 SPS

The Sensor Planning Service (SPS) is a service interface that facilitates the planning of tasks for sensors (Simonis 2011). It has been an official OGC standard since 2011. The main function of the SPS is to provide a way for users to search for and request tasks for sensors, and for the SPS to assign the tasks to the appropriate sensors. Once the sensor

completes the task, the SPS sends a web notification to the users and the collected data is sent to the next step, such as the SOS. The SPS is a key component in the management and coordination of sensor networks, and it plays a crucial role in the efficient collection and dissemination of sensor data.

1.2.2.4 SensorThings API

The SensorThings API (STA) has been an official OGC standard since 2016. It is an open solution that is specifically designed to interconnect Internet of Things (IoT) sensing devices and geospatial data. STA is composed of two parts: the tasking part, which is focused on controlling sensor devices, and the sensing part, which is dedicated to sensing and collecting observations from sensors (Liang 2016). The core of STA is the data model, which is depicted in Figure 7. This data model is created based on the ISO/OGC Observations and Measurements (O&M) model (S. Cox 2007) and illustrates the different entities and their properties, as well as the relationships between entities.

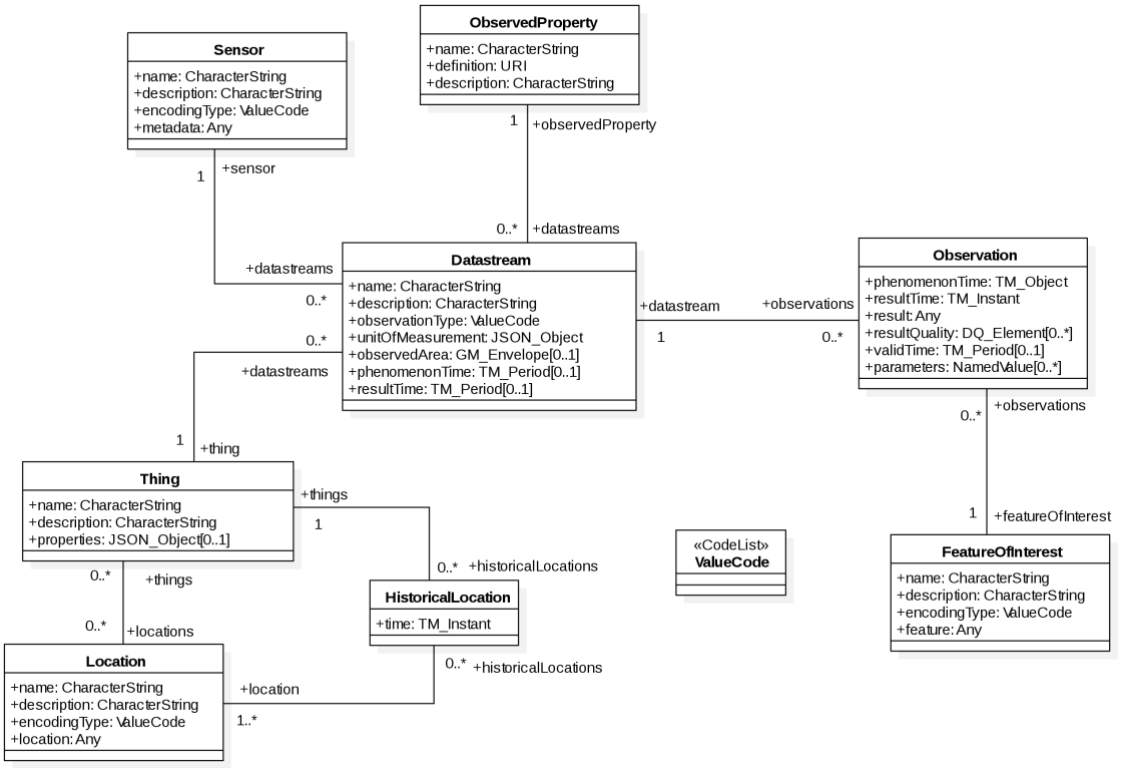


Figure 7 SensorThings data model – sensing part (Liang 2016).

STA conceptual data model is composed of mainly 8 entities. Table 3 provides a specific description about each entity of STA:

Entity Name	Description
Thing	A Thing can be geo-referenced in different spaces. For example, in an environmental context, a Thing may refer to a monitoring station, a river, or a moving ship. A Thing has three attributes: name, description, and properties. Additionally, a Thing may be associated with one or multiple Locations and Datastreams.
Location	A Location represents the latest position of a Thing. For example, it may indicate the position of the latest captured measurement. A Location has four attributes: name, description, encodingType, and location. The encodingType attribute specifies the format of the location attribute; its value is one of the ValueCode enumeration. The location attribute contains the actual geographic coordinates of the Thing.
HistoricalLocation	A HistoricalLocation expresses all positions of the Thing. It can provide the previous or the latest location of the associated Thing together with the time.
Sensor	A Sensor describes a sensing device or an observation producer that observes a property or phenomenon with the goal of producing an estimate of the value of the property.
ObservedProperty	An ObservedProperty represents a property that is observed, e.g., temperature, or humidity.
Datastream	A Datastream describes a set of Observations of the same ObservedProperty which are produced by the same Sensor and Thing.
Observation	An Observation represents the main value of a property or a sensor. The event can be numerical values in various formats i.e., JSON or array format. This entity is similar to the O&M data model (S. Cox 2007). An Observation is linked to a time of collection, also known as the phenomenonTime, which indicates when the observation was recorded. This information is crucial to understand the context of the observation and to identify trends over time.
FeatureOfInterest	A FeatureOfInterest describes additional value of an Observation. FeatureOfInterest can describe the location of the observation, e.g., FeatureOfInterest can be the geographical area or volume sensed in the case of remote sensing.

Table 3 Entity description of the STA data model (Liang 2016).

There most relations are one-to-many, i.e., Thing and Datastream relation, Datastream and Observation relation. From the relation we can see that a Datastream has exactly one Thing, and a Thing can have zero to many Datastreams.

1.2.2.5 STA and SOS Comparison

SOS can be viewed as a standard for modeling using both sensorML and O&M data models and presenting data based on SOAP (Simple Object Access Protocol). In the context of data modeling and exchanging for sensor and IoT, the main limitation of this standard is that these interfaces are not optimized to develop lightweight client applications to explore and visualize observation data.

STA was developed by OGC to take over SOS standards for modeling and exchanging sensor data and metadata. Their data model is mainly designed based on ISO/OGC O&M data model. It also provides a HTTP interface (Hypertext Transfer Protocol) for querying data. It combines two parts: (i) *sensing* part similar to SOS and (ii) tasking part is similar to SPS. In the technical perspective, OGC STA is superior to OGC SOS in several aspects, e.g., STA supports event-based data flows, primarily relying on the Message Queuing Telemetry Transport (MQTT) protocol (Cohn 2014) which enables stream processing of data.

Furthermore, OGC STA offers a range of benefits for the IoT community (Liang 2016). Firstly, it enables the proliferation of new, high-value services with lower development overhead and a wider reach. Secondly, it reduces risks, time, and cost across the full IoT product cycle. Thirdly, it is a free, open solution that aligns with the key vision of IoT, which is to provide low-cost and simple sensors. Lastly, it simplifies the connections between devices-to-devices and devices-to-applications, facilitating the integration of IoT devices and systems, making it easier for developers to create and deploy new services and applications. Overall, the OGC STA is a valuable tool for the IoT community, providing a standard for data models and communication protocols that enables interoperability and scalability across a wide range of applications. We provide a synthetic comparison of the common techniques between STA and SOS in Table 4, based on existing comparisons (Na 2007, Liang 2016, Santhanavanich 2018).

Standard Model Name	**OGC STA (SensorThings API)**	**OGC SOS (Sensor Observation Service)**
Published when	2016	2007
Original encoding	JSON	XML
Architecture style	Resource Oriented Architecture	Service Oriented Architecture
Binding	REST	SOAP
Capabilities to insert new sensors and observation results	HTTP POST	SOS specific interfaces, e.g., *InsertSensor(), InsertResultTemplate(), InsertResult(), 3D*
Deleting existing sensors	HTTP DELETE	SOS specific interfaces, i.e., *DeleteSensor()*
3D location of a moving sensor	Supported directly through *Location* and *HistoricalLocation* entity	Supported by SOS specific interfaces
Sensor metadata	Supported. (OGC O&M specification)	Supported. (OGC O&M specification)
Updating properties of existing sensors or observations	HTTP PATCH and JSON PATCH	Supported
Publish–subscribe Support (similar message queue)	MQTT and SensorThings MQTT Extension	Not supported
Deleting observations	HTTP DELETE	Not supported
Linked data support	JSON-LD	Not supported
Return only the properties selected by the client	*$select*	Not supported
Return multiple O&M entities (e.g., FeatureOfInterest and Observation) in one request/response.	*$expand*	Not supported

Table 4 Technical comparison between STA and SOS (Na 2007, Liang 2016, Santhanavanich 2018).

1.3 Big Spatial Vector Data Management

In this section, we provide an overview of the storage and processing technologies for big spatial vector data. We present the concept of big data and big spatial vector data. Then, we discuss about big data storage and processing systems.

1.3.1 Big Data

Big data, as defined by (De Mauro 2015), "*represents the Information assets characterized by such a High Volume, Velocity and Variety to require specific Technology and Analytical Methods for its transformation into Value*.". It refers to extremely large and complex data sets. These data sets can come from a variety of sources, such as social media, sensor networks, and more. In the context of the environment, big data can be used to monitor and study phenomena such as climate change, natural disasters, and the spread of invasive species.

Managing and analyzing big data can present a number of challenges when using traditional approaches, as highlighted in the literature by authors such as (X. L. Dong 2013) and (Patgiri 2016). One of the primary challenges is the *volume* of data, which is often larger than traditional datasets. This can make it difficult to store and process the data using conventional methods and may necessitate the use of specialized tools and technologies such as cluster management and cloud computing. Another challenge is the *velocity* of big data, which refers to the high frequency of incoming data. This can make it difficult to process the data in real-time and may require the implementation of specialized tools and technologies such as stream processing and real-time analytics. The *variety* of big data can also pose a challenge, as it may come in different formats and models from various sources. This can make it difficult to handle diverse data using a single system and may necessitate the use of specialized tools and technologies such as data integration and data warehousing. In addition, big data can also exhibit inconsistencies and uncertainty, commonly referred to as *veracity*. Veracity challenges can arise from the heterogeneous nature of data sources and the complexity of data processing, which may impact the quality of data and the accuracy of analysis. Finally, extracting useful information from big data can be a complex task due to the heterogeneity of data sources, also referred to as *value*. The challenge is to handle big data with heterogeneous sources. Overall, managing and analyzing big data can be a significant challenge, requiring specialized tools and technologies, as well as skilled personnel with expertise in big data management and analysis.

1.3.2 Big Spatial Vector Data

Big spatial data (BSD) refers to large and complex sets of data that possess spatial data components (Eldawy 2015). Big spatial vector data (BSVD) is a subcategory of BSD in vector format. In the context of IoT environmental monitoring, this geo-referenced data may be generated in real-time. It is used to monitor and control various systems and processes, as well as to analyze and visualize the data generated by IoT devices and extract meaningful insights. The IoT, which refers to the network communication of physical devices such as sensors and actuators that are connected to the internet and able to communicate with each other and with other devices and systems, also plays a key role in the generation and collection of BSD. It enables the collection of vast amounts of data from a wide range of sources in real-time.

One of the main challenges in working with BSD, especially big spatial vector data in the context of IoT is data management (Yao 2018) which includes the storage, retrieval, and processing of large, complex sets of data coming from heterogeneous sources. This requires the use of advanced data management techniques and technologies such as distributed storage and processing, cloud computing, and NoSQL databases. Data processing and data analysis is another important challenge in working with BSVD. This includes the ability to handle the high volume, velocity, and variety of data, as well as the ability to handle the real-time data processing and the real-time data analysis. (Tatbul 2010) point out the challenges of integrating a streaming processing engine (SPE) with other SPE or with DBMS (database management system). They note that the research field is promising, and still open as new use cases appear, and systems are very heterogeneous which makes integration harder.

Data visualization is another key challenge in working with BSVD. This includes not only the ability to extract meaningful insights from the data, such as identifying patterns, trends, but also the ability to represent the data in a clear and meaningful way, through maps, graphs, and other visualizations. In the next sections, we first describe the database model and storage. Next, we recall the different approaches for large spatial databases and present the components of ELK stack. Then, we recall the data processing and analytics concept.

1.3.3 Big Data Storage

The database model and storage are the fundamental step in data management. The research community has made significant contributions for this area, particularly for Big Spatial Data (Guo 2020). In this section, we discuss the different database models for large spatial data. Then, we present in detail the ELK stack. We present as well existing systems based on ELK stack for spatial data.

1.3.3.1 NoSQL Databases for Large Spatial Data

Generally, spatial data are principally stored into either relational SQL databases or NoSQL databases. In the context of big spatial data, geospatial databases need to be adaptable and scalable, with a flexible database schema and a fast query execution time. However, relational SQL databases such as PostgreSQL (Momjian 2001) and its extension PostGIS (Obe 2015), SQL Server (SQL Server 2023) are fixed schema databases and limited in scalability. NoSQL databases ("not only SQL") are non-relational databases, they are distributed systems and have flexible storage schema (Guo 2020). There are six types of NoSQL database models: document, graph, wide-column, key-value, multi-model, search engine. Table 5 presents these database models and their corresponding popular NoSQL solutions.

Database Model	**Databases**
Document	MongoDB (Amirian 2014), Couchbase (Vatsavai 2012)
Graph	Neo4j (Schmid 2015)
Wide-column	Cassandra (Baralis 2017), HBase (Patroumpas 2014)
Key-value	Redis (Lee 2015)
Multi-model	Amazon DynamoBD (Sivasubramanian 2012)
Search engine	Elasticsearch, Splunk, Solr (Zhen 2016)

Table 5 The 10 most popular NoSQL databases (Guo 2020).

(Guo 2020) concluded that "*document databases are the best platform for geospatial data processing, as they load fast and have a good execution time, good query*

performance, and abundant geospatial functions and index methods.". Additionally, Elasticsearch can be categorized either as a search engine or document database model. (Guo 2020) indicated as well that Elasticsearch and MongoDB are better than other NoSQL databases in terms of supporting vector geometry functions.

Elasticsearch supports both streaming and geospatial data models and has significant advantages over its competitors for analyzing geo-referenced streaming data thanks to its design. On one hand, (Barnsteiner 2015) showed that Elasticsearch is as efficient as native Time Series Database (TSDB) (openTSDB 2010) for time series data. On the other hand, (Guo 2020) argued that Elasticsearch supports many geometry object structures and functions as spatial DBMS. Moreover, its query engine supports aggregation queries and connects natively to Kibana for building analytical dashboards. In Table 6, we provide a comparative study of databases with spatial and temporal capabilities. We show in this table that Elasticsearch provides important aggregation functions, geospatial functions, formats and indexes. Elasticsearch is sufficiently suitable for spatial data warehouse, with the streaming capabilities in addition. InfluxDB is the reference solution for time-series data, however it lacks support for spatial data types. InfluxDB (Naqvi 2017) contributors are currently working on this feature but it is still experimental and limited. Moreover, document databases, e.g. MongoDB, have been shown performing well with geospatial data queries, and outperforming relational spatial databases such as PostGIS (Agarwal 2016, Bartoszewski 2019). In the next subsections, we present Elasticsearch.

Databases	Query Language	Primary Database Model	Secondary Database Model	Supported Objects	Supported Geometry	Main Supported Geometry Functions	Supported Spatial Indexes	Geometry Data Format	Visualisation Feature	Support Aggregation Queries
Mongodb (Amirian, 2014)	NoSQL	Document	Spatial DBMS, Search engine, Time Series DBMS	Point, Polygon, MultiLineString, MultiPolygon, GeometryCollection	LineString, MultiPoint,	$geoIntersects, $geoWithin, $nearSphere	$near,	2dsphere index, 2d index based on geohash	GeoJSON objects, Legacy Coordinate Pairs	MongoDB Charts	Yes
PostGIS (Obe, 2015)	SQL	Spatial DBMS	Relational DBMS	Point, Polygon, MultiLineString, MultiPolygon, GeometryCollection	LineString, MultiPoint,	ST_Contains, ST_Intersects, ST_Within, ST_Area, ST_Boundary, ST_Centroid, ST_Disjoint, ST_Distance, (PostGIS, 2022)	etc.	R-Tree	Well-known text (WKT), Well-known binary (WKB), Geographic Mark-up Language (GML), Keyhole Mark-up Language (KML), GeoJSON, Scalable Vector Graphics (SVG)	ArcGIS	Yes
InfluxDB (Naqvi, 2017)	NoSQL	Time Series DBMS		Not available		Not available - experimental		Not available	Not available	Grafana	Yes
Elasticsearch (Elasticsearch, ELK, 2020)	NoSQL	Search engine	Spatial DBMS, time series DBMS, document	Point, Polygon, MultiLineString, MultiPolygon, GeometryCollection	LineString, MultiPoint,	geo_bounding_box, geo_distance, geo_polygon, geo_shape: (intersects, disjoint, within, contains), geohash grid, geo_bounds, geo_centroid, geo_line		GeohashPrefixTree and QuadPrefixTree	GeoJSON and WKT	Kibana, Grafana	Yes

Table 6 Basic information about four database management systems for geospatial data.

1.3.3.2 ELK Stack

In this section, we present ELK stack and the comparison between Elasticsearch and its competitors. The ELK stack is composed of four main open-source projects: Beats, Logstash, Elasticsearch, and Kibana (Elasticsearch, ELK 2020). Beats are data shippers, Logstash is a data processing pipeline, Elasticsearch is a document-oriented database, and Kibana is a visualization tool.

Beats and Logstash

Beats are lightweight data shippers that mainly ship data from the source, such as files or databases to Logstash. Logstash is a processing pipeline engine with real-time pipelines. It ingests data from multiple sources, transforms, and ships the transformed data to various destinations such as Elasticsearch (Bajer 2017). Each data processing event of Logstash is processed in three main stages: (i) input stage specifies the input source, (ii) filter stage regroups the transformations, and (iii) output stage defines the destinations.

Elasticsearch

Elasticsearch (ES) is the heart of the ELK stack. It is an open-source search engine and document-oriented distributed database with real-time and full-text searching. In ES, records are called documents. The documents are JSON objects that are stored within an Elasticsearch index. An ES Index is comparable to a database in the relational database world. A document is similar to a row in a table and the document fields are table attributes. An index field can take on various types e.g., a string, a number. The process of mapping an index in ES is comparable to creating a database schema. It involves defining the field types and overall structure of the ES index. ES is also designed as a distributed database (Elasticsearch 2021). It extends the concept of relationship between clusters, nodes, indexes and shards.

- A cluster can have one or multiple nodes.
- An ES index can be divided into one or several shards (partitions).

- Shards of an index can be populated across one or more nodes in the same cluster.

Based on the relationship above, the greatest advantage of ES in terms of storage is scalability and reliability.

- *Scalability* means that ES can easily increase the size of storage, e.g., adding new nodes by configuring them in the same cluster, or adding new shards for an existing index without suspending this index.
- *Reliability* means that if some nodes stop working, others in the same cluster can take the workload of the failing one.

ES supports a varying number of field datatypes in a document (Elasticsearch, ELK 2020) which can be classified into four primary categories in ES (as illustrated in Figure 8). They are They are core datatypes (e.g., string, numeric, date, boolean), complex datatypes (e.g., object for single JSON objects), geo datatypes, and specialised datatypes (e.g., IP address).

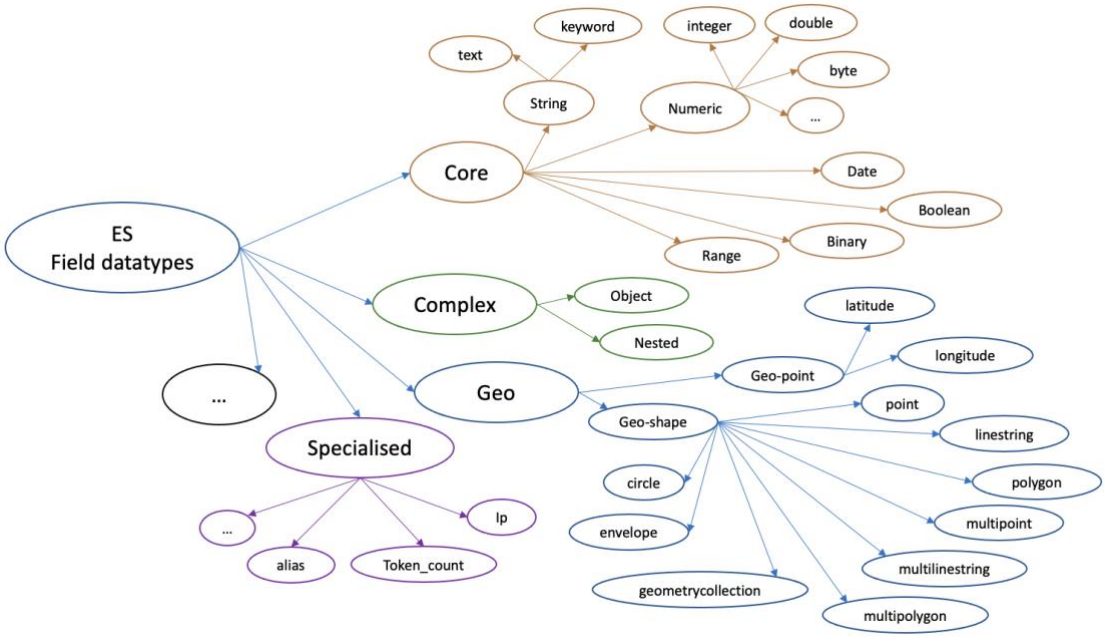


Figure 8 ES field datatypes.

ES supports two groups of spatial data types: geo-point and geo-shape. Geo-point is a pair of latitude and longitude data. Geo-shape includes a point or set of points (e.g., point, linestring, polygon, multipoint, multilinestring, envelope, and circle).

In the following, we summarize the query language of Elasticsearch. ES supports a DSL query (Domain Specific Language) based on JSON data (Elasticsearch, ELK 2020). A query structure combines three parts: (i) the API method; (ii) an address that includes URL, index, type, and method; and (iii) query body. ES has five main query categories of query types (Elasticsearch, ELK 2020): full-text queries, boolean queries, geo queries, search template queries, and aggregation queries (as illustrated in Figure 9). ES aggregation queries have two main types: (i) metric aggregation query and (ii) bucket aggregation query. Metric aggregation queries aggregate the values of some fields over a set of documents. The interesting metric aggregation queries for our project are statistical queries (e.g., min, max, average) and geo function queries (e.g., geo-bound). Bucket aggregation queries consist in grouping documents with respect to a field. There are three main categories for bucket queries: (i) filter queries to compute one bucket, (ii) range queries to compute buckets by some ranges, and (iii) term queries to compute one bucket for each value with respect to some index field.

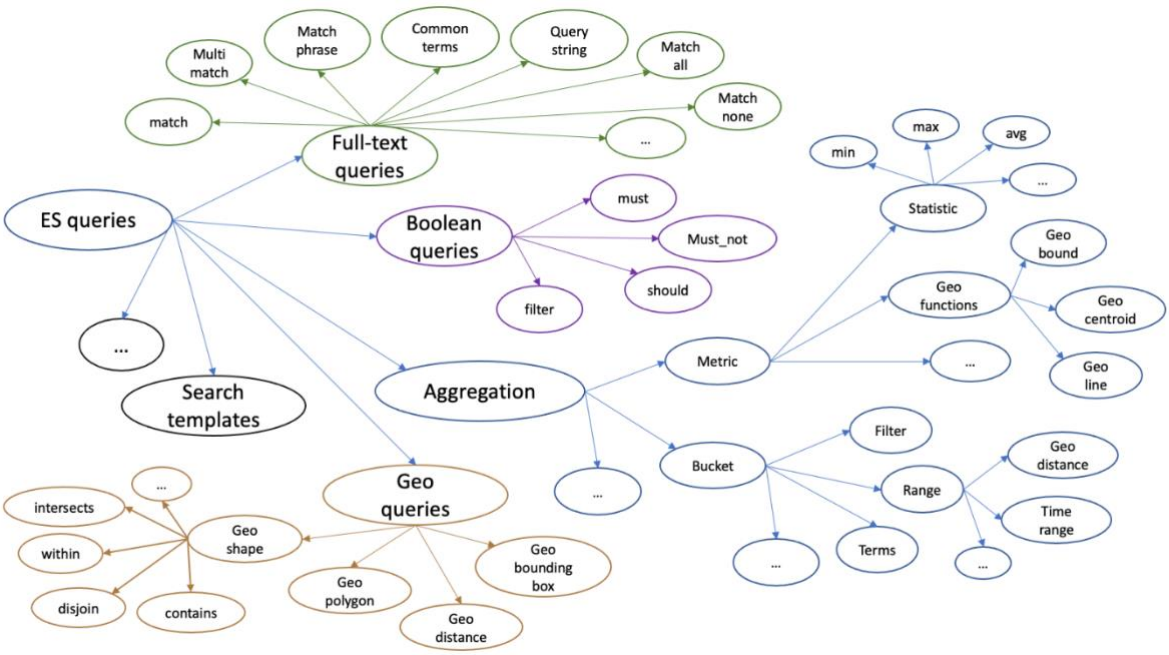


Figure 9 Query types in ES.

Kibana

Kibana is a tool designed for Elasticsearch that enables data visualization and management (Bajer 2017, Tewtia 2021). It combines a variety of visualization methods, such as charts, plots, maps, and data tables. Dashboards, which combine multiple visualizations in a single interface, are a key feature of Kibana. Additionally, Kibana provides a user-friendly interface for writing queries in the ES query language. Elasticsearch and Kibana are commonly used together in various use cases for data storage and visualization (Bajer 2017, Dubey 2018). The primary method of visualizing geospatial data is through a map (Wilke 2019). Kibana offers the most common types of spatial visualization maps such as point maps, choropleth maps, heat maps.

- A point map is the simplest method of visualizing spatial data, where filtered data is presented as a point on the map e.g., a building, a farm.
- A choropleth map represents variations of a measure through coloration.
- A heat map describes a range of values using colors or shades.

Figure 10 presents an example of map visualization using Kibana (ELK stack 2022) for tracking a flight in North USA. It displays the original and modified flight paths between Logan international airport and Bangor international airport in the USA. It also represents areas affected by bad weather through a heatmap and points indicating lightning strikes. This visualization explains the main reason for modifying the flight path.

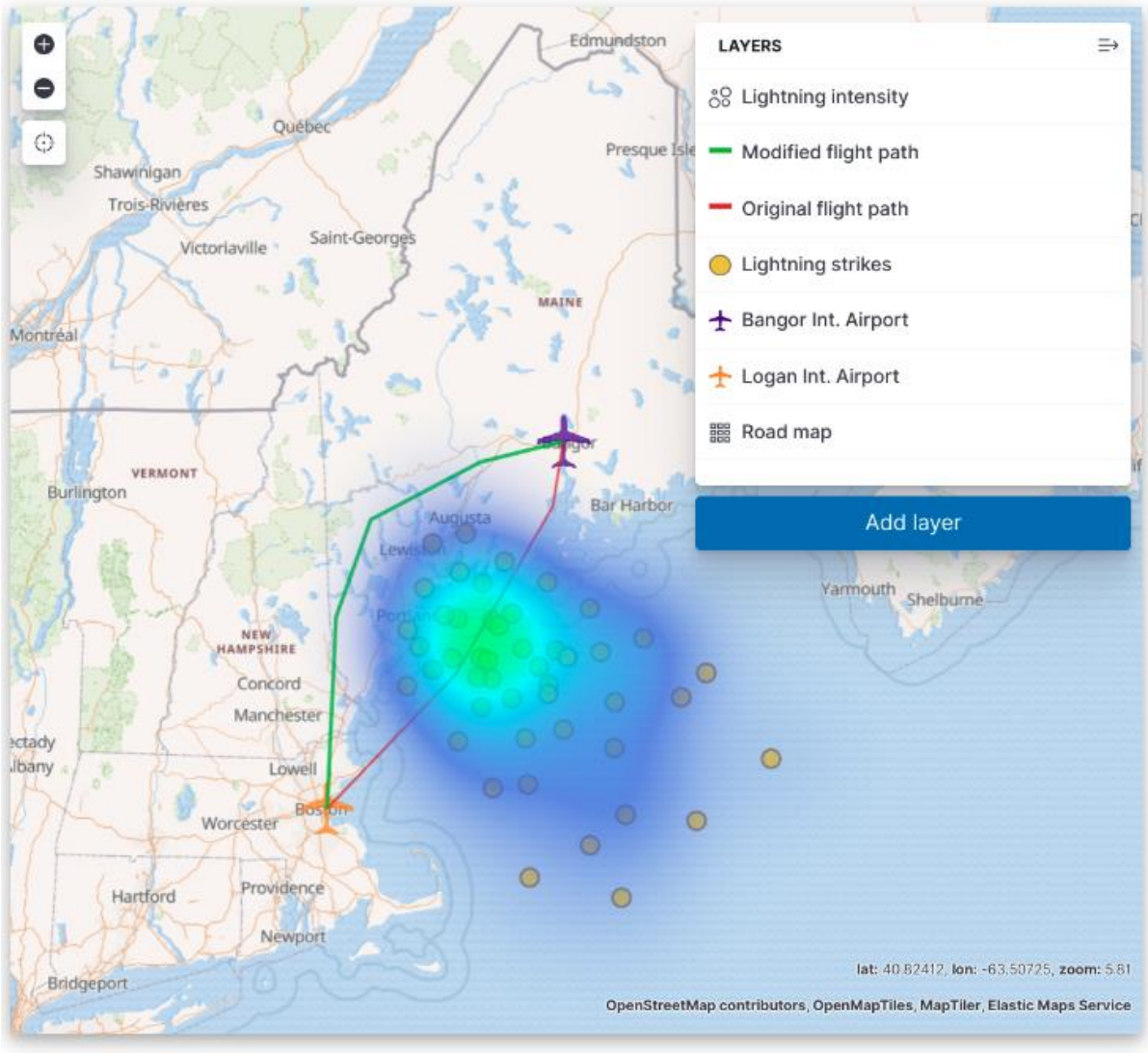


Figure 10 Example of maps visualization of Kibana (ELK stack 2022).

1.3.3.1 Existing Systems based on ELK Stack for Spatial Data

(Quoc 2015) and (Bartlett 2019) proposed Elasticsearch as a scalable storage, and fast querying engine for big spatial sensor data. Both papers showed that ES has the ability to handle complex analytical spatiotemporal query functions. (Kramer 2020) provided a software product to store and query different geospatial file formats such as geojson, gml, cityml in both MongoDB for document storage, and Elasticsearch for indexing data and fast searching. They used ES mainly for storing JSON files and to retrieve them. (Bajer 2017) proposed ELK stack for managing IoT data. They argue that ELK stack is appropriate for such use cases as it handles high insert throughput, has user-

friendly GUI for data visualization, and has comprehensive capabilities for spatio-temporal queries. We indicate as well that the latest version of geonetwork (Geonetwork 2022), which is a well known geo-referenced data catalog has adopted Elasticsearch. However, no prior work has used and evaluated Elasticsearch for data warehousing and analytics in the context of streaming geo-referenced sensor data. It is this knowledge gap that serves as a driving force for our contribution, as detailed in Chapter 2.

1.3.4 Big Data Processing

The processing, analysis, and extraction of valuable insights from large volume of data, particularly big spatial data has become a significant challenge for various fields. In response to this challenge, various tools have been developed (M. M. Alam 2021). In this section, we present Apache Spark (Zaharia 2012) an open-source distributed computer system proposed for rapid processing of large-scale big dataset and provide a comparison with other well-established frameworks. Then, we present GeoSpark (Yu 2015), an extension of Apache Spark for spatial data processing.

1.3.4.1 Apache Spark

Apache Spark (Zaharia 2012) is a high-performance cluster computing system designed to process large amounts of data efficiently. Unlike traditional systems such as Hadoop (Shvachko 2010), which reply on disk-based storage, Spark operates primarily in-memory. Spark's core feature is the Resilient Distributed Databases (RDDs) data abstraction, which involves distributing sets of items across a cluster of machines. RDDs are created through parallelized transformations such as filtering, joining, or grouping, and can be recovered in case of data loss thanks to their lineage that tracks how each RDD was built from other datasets through transformations. This lineage feature ensures fault tolerance of Spark by rebuilding lost data.

Spark's workflow management is achieved through a Directed Acyclic Graph (DAG), with nodes representing RDDs and edges representing RDD operations. RDDs can

undergo two types of transformations: narrow and wide. Transformations are operated on partitions of RDDs. Narrow transformations do not require data to be shuffled across partitions to produce the subsequent RDD. Conversely, wide transformations require data to be shuffled across partitions to create the new RDD. Examples of operations that require wide dependency include Reduce, GroupByKey, and OuterJoin, which initiate a new stage and lead to stage boundaries.

Spark offers four primary modules, including (i) SparkSQL (M. a. Armbrust 2015) for structured data processing and SQL operations, (ii) Spark Structured Streaming (M. a. Armbrust 2018) for processing unbounded structured datasets, (iii) MLlib (Meng 2016) for machine learning, and (iv) GraphX (Gonzalez 2014) for graph processing. These modules make Spark a versatile and powerful tool for big data processing and analysis.

SparkSQL (M. a. Armbrust 2015) is a module of Spark designed explicitly for processing structured data. It provides two primary features that enhance its functionality. First, it presents a higher-level abstraction, called a DataFrame, which structures data as a table with columns, much like a relational database. Second, it includes a flexible optimizer, known as Catalyst, which is represented as a tree and follows a general rule library for manipulating the tree. The Catalyst tree transformation framework involves four phases: analysis, logical optimization, physical planning, and code generation. Catalyst optimizes the query plan, as shown in the rounded rectangles in Figure 11, by analyzing and optimizing a logical plan, proposing physical plans with their respective costs, and generating code.

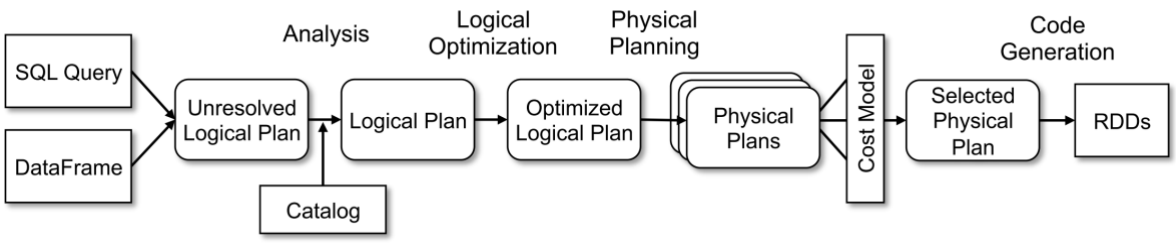


Figure 11 Phases of query planning in SparkSQL (M. a. Armbrust 2015).

Table 7 displays a comparison of the three popular big streaming frameworks, highlighting their relevant aspects for streaming data processing (Chintapalli 2016, Inoubli 2018). Apache Spark offers several processing approaches, such as real-time, batch, and micro batch, making it a versatile choice. However, these features come at the cost of higher latency and resource consumption. Additionally, Spark has the advantage of supporting SQL and connecting natively to a wider range of data sources.

	Apache Spark (Zaharia 2012)	**Apache Storm** (Toshniwal 2014, Storm 2014)	**Apache Flink** (Carbone 2015)
Processing approach	Real-time, Micro Batch Streaming	Streaming - Trident	Run time streaming
Streaming engine	Spark streaming processes data streams in micro-batches	Designed as DAG with spouts, bolts and streams used to process data	A streaming engine for such workloads; micro-batch, and batch
Data format	Discretized Stream or also called DStream (it is a continuous stream of data), DataFrames	Tuples	DataStream
Programming languages	Python, Java, Scala	Java, Scala	Python, Java
Cluster manager	Hadoop YARN (Vavilapalli 2013), Kubernetes, Standalone, Apache Mesos	Zookeeper	Hadoop YARN, Kubernetes
Streaming query	SparkSQL, Structured Streaming	No	No
Latency	Few seconds	Sub-second	Sub-second
Sliding window	Time based	Time based and count based	Time based
Message	Exactly-once	At-least-once	Exactly-once
CPU consumption	**	*	***
RAM consumption	***	**	***
Processing power	100x faster than Storm	Millions of tuples per second per nodes	1000s per nodes

Table 7 Comparison of architecture characteristics of Spark, Storm, Flink (Chintapalli 2016, Inoubli 2018).

While Spark and SparkSQL offer powerful data processing capabilities, they do not have built-in support for spatial data and its operations. To address this limitation, we

present GeoSpark (Yu 2015) and compare it with the most widely used Spark-based systems for spatial data handling in following sections.

1.3.4.2 GeoSpark

GeoSpark (Yu 2015) is an extension of Spark that enables the loading, processing, and analysis of large-scale spatial data. It presents the Spatial RDD (SRDD), which enhances the native RDD with spatial data types such as point, line, and polygon. It supports fundamental spatial operations as well including range query, kNN query, and join query.

To improve the speed of spatial query processing, GeoSpark incorporates several indexing techniques, including Quad-Tree (Finkel 1974), and R-tree (Guttman 1984) for the SRDD. These techniques, combined with the implementation of the Filter and Refine model (Wood 2008), significantly improve the performance of spatial query. Recently, GeoSpark has been endorsed by the Apache Foundation and has been renamed Apache Sedona (Sedona 2022).

In addition to GeoSpark (Yu 2015), several other Spark-based systems have been extended for spatial data processing, including SpatialSpark (You 2015), Simba (Xie 2016), SparkGIS (Baig 2017), and LocationsSpark (Tang 2020). Table 8 displays a comparison of noteworthy spatial streaming frameworks for large-scale data.

Spatio-temporal Systems	Underlying System/ Architecture	System Type	Spatial Data Types	Partitioning	Indexing	Query Language	Supported Queries
Tornado (Mahmood 2015)	Apache Storm	Spatio-textual Stream	{srcid, oid, (x, y), t, text} A-Grid	A-Grid	Adaptive Indexing Global: Spatial (A-Grid) Local: Spatio-textual (KD-Tree)	Atlas (SQL-Like)	Snapshot, Continuous (Range, kNN, Join)
SSTD (Chen 2020)	Apache Storm	Spatio-textual Stream	Point	QT-tree (Spatial, Textual)	Global: QT-tree Local: Object, Query	N/A	Snapshot, Continuous (Range, kNN, Top-k)
GeoFlink (Shaikh 2020)	Apache Flink	Spatial Stream	Point	Grid	Grid-based	N/A	Continuous (Range, kNN, Join)
GeoSpark (Apache Sedona) (Yu 2015)	Apache Spark	Spatial Stream	Point, LineString, Polygon, Rectangle	Uniform-Grid Voronoi, R-Tree, Quad-Tree, KDB-Tree	R-Tree, Quad-Tree	Extended Spark SQL	Range, kNN, Spatial Join, Distance Join

Table 8 The four most popular big spatio-temporal systems (M. M. Alam 2021).

1.4 Data Analytics and Integration

Data analytics, as defined by (Rajaraman 2016), is "*concerned with extraction of actionable knowledge and insights from data.''*. One process of data analytics is the data integration which is the process of incorporating data from multiple sources into a single view or location (Lenzerini 2002, X. L. Dong 2013). In general, data integration has two well-known methods: data warehouse and mediation system. Data warehousing consists in collecting and incorporating source data into a single data store. The process consists in collecting data from heterogeneous sources, transforming, and storing data into a single data repository. The transforming process regroups a variety of steps, such as, cleaning, aggregating and joining data sources. Mediation system consists in defining an integrated schema together with mappings between the integrated schema and local schemas. In mediation systems, data remain in local sources and user queries are rewritten through a mapping process. The resulting queries are run on top of the local data stores. Any operation including different data sources, such as joins, are executed by the mediation system. In this section, we first recall the fundamental spatial queries concept. Then, we present data warehouse and mediation systems. Finally, we present geospatial visualization with examples for spatial data analytics.

1.4.1 Data Warehouses

Data warehouses (DW) are designed specifically for analytical queries (Bedard 2001, Inmon 2005, Wrembel 2006). In the context of data integration, a data warehouse is usually a central hub for storing and organizing data from different sources. It may be involved to clean and transform the incoming data as well as store them. Data in data warehouses are organized into either facts or dimensions. They are generally modeled in either star schema or snowflake schema. A fact is a focus of interest for the decision-making process. It consists of measures or metrics. A dimension is a fact property, it describes one of the fact's analytical coordinates. Dimensions are used to aggregate data (Jarke 2002). Data warehouses are usually described using multidimensional conceptual models. The multidimensional data structures of data warehouses are the data cubes. A popular analytical tool for data warehouses is the OLAP tool (Online Analytical Processing). The main OLAP operations are roll up, drill down, slicing, and dicing (Bedard 2001, Matei 2014).

Spatial data warehouses and spatial OLAP extend the concepts of a data warehouse and OLAP (Bimonte 2010, K. Boulil 2012). In particular, they provide a method to store, aggregate and analyze spatial data and geo-referenced environmental data (Nipun Garg 2011, Miralles 2011, K. a.-P. Boulil 2013). In spatial data warehouses, facts or dimensions may consist of spatial objects.

Data warehouses require ETL processes to get data from sources (Lenzerini 2002, Albrecht 2008, Bansal 2015). ETL consists of three processes: (i) extracting data from multiple heterogeneous or homogeneous sources, (ii) transforming data and (iii) loading the data into data warehouses. ETL process can be either batch ETL or Streaming ETL. On one hand, batch ETL processes data in batches at a specific time. On the other hand, streaming ETL processes data continuously. The later approach allows analysis of data as soon as it is generated and collected from the sources.

Facts

In data warehousing, a fact is a key element that is critical for decision-making processes (Jarke 2002). Essentially, a fact is a collection of measures or metrics that are

used to analyze and report on data. These measures are typically quantitative attributes, such as temperature or air humidity. To perform detailed analysis, facts are associated with dimensions that provide context to the data. A dimension is a fact property that describes one of the fact's analytical coordinates. For example, to calculate the average temperature of a specific city on a particular date or in a particular month, the temperature values must be combined with dimensions such as time and location.

Dimensions

In data warehousing, dimensions play a determining role in aggregating data and providing context to facts (Jarke 2002). A dimension is essentially a category that organizes data into hierarchies and provides a level of granularity. For example, a geography dimension might include hierarchies for country, department, and city. These levels of hierarchy enable users to analyze data at different levels of detail. Dimensions are used to slice and dice data for analytical queries and to access different levels of granularity through drill-down and roll-up operations. This allows users to examine data from different perspectives and gain deeper insights into the data.

1.4.2.1 Modeling

In this section, we present two schemas widely used to model data warehouses: star schema (Kimball 1996), and snowflake schema (Gray 1998).

Star Schema

The star schema derives its name from its physical schema diagram, as it looks like a star with a central fact table and multiple dimension tables surrounding it. Figure 12 illustrates an example of the star schema model for a relational database as presented by (Han 2012). The schema comprises measurements of sales and four dimensions, including time, item, branch, and location. The fact table contains two quantitative attributes and foreign keys to the dimension tables. The main advantage of the star schema is its simplicity, where each dimension has a direct relationship to the fact entity, making it easy to comprehend and query. Additionally, it offers excellent performance for aggregation queries and has a high ability to scale, making it an ideal choice for large and complex data sets.

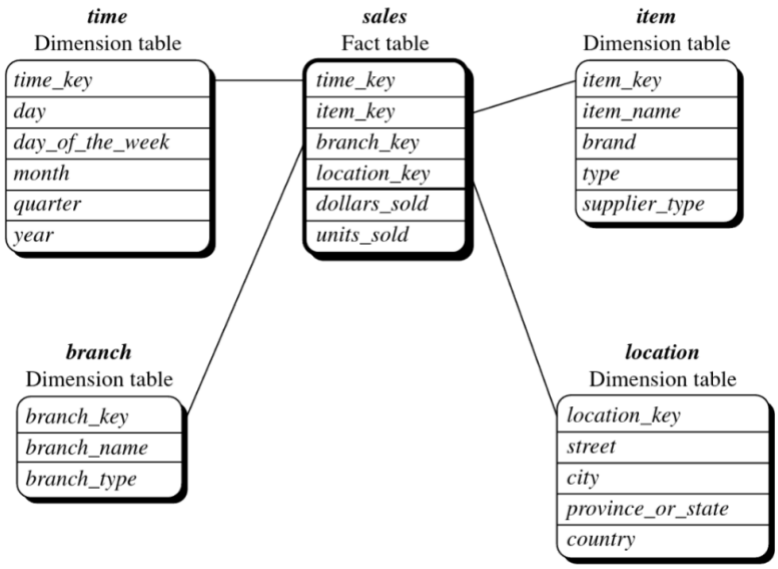


Figure 12 Example of fact and dimension tables of the star schema model (Han 2012).

Snowflake Schema

The Snowflake schema is a variation of the star schema model in data warehousing. Its mainly distinguishing feature is the normalization of dimension tables, which significantly reduces data redundancy. The normalized dimension tables are linked to each other through foreign key relationships, similar to those found in a relational database. Figure 13 provides an illustrative example of the Snowflake schema model for a relational database, where the sale fact table closely resembles the fact table of the Star schema in Figure 12. However, the main difference between these two schemas is the representation of the dimension tables. For instance, in the Snowflake schema, the single dimension table for items in the Star schema is normalized into two tables, namely, item and supplier. The Snowflake schema may entail more complex queries than the Star schema, but it offers a more flexible and scalable solution for managing complex data sets.

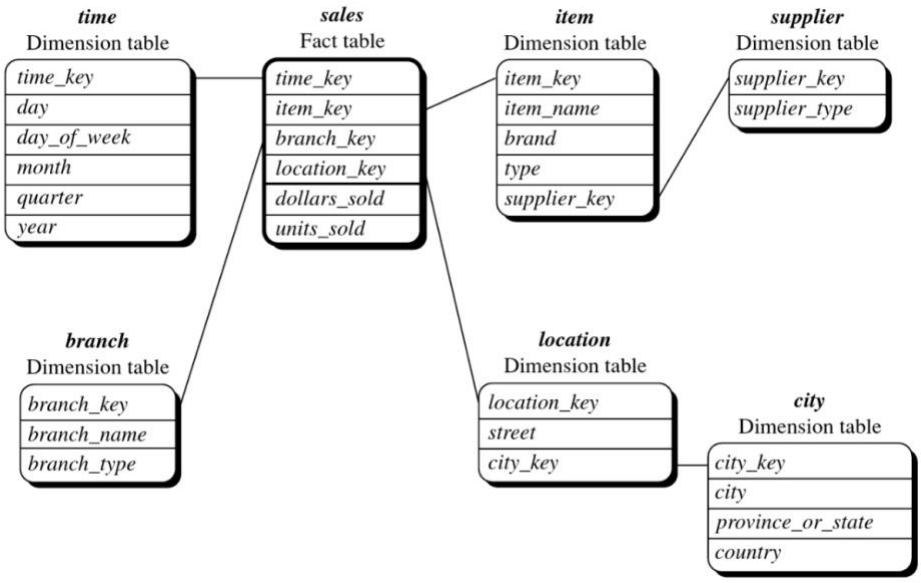


Figure 13 Example of fact and dimension tables of the snowflake schema model (Han 2012).

1.4.2.2 Conceptual Modeling of Spatial Data Warehouse

One of the most essential steps for the data warehouse designing is to build conceptual model. (K. a. Boulil 2015) has proposed a spatial data warehouse conceptual model. The authors show that it is one of the most complete conceptual models for spatial data warehouses based on the standard UML. We use this UML profile in our contribution in Chapter 4.

A UML profile is an extension mechanism of UML metamodel to customize UML for specific domains and use cases. Stereotypes, tag definitions, and constraints are utilized in the definition of profiles, as identified by (OMG 2011) and are applied on UML metaclasses such as class and association. A stereotype is an extension of a UML metaclass, while tagged values indicate properties of a stereotype. Constraints provide clarity on the definition of stereotypes and tagged values. At the model level, stereotypes are described as (<<stereotype name>>), while tagged values are represented as (<<tagged value name = value>>). (K. Boulil 2012) proposed a SDW core model package to represent spatial data warehouse modeling using the UML metamodel representation. Figure 14 displays the UML metamodel of the SWD core

model package. A model (<<SDWCoreModel >>) is composed of one or more packages (<<Hypercube>>). A Hypercube describes a subject of analysis.

Each Hypercube includes a Fact class (<<Fact>>) and a set of dimensions (<<Dimension>>). The Fact class combines a set of (<<Measure>>) properties. This set of measures can be empty. Fact is linked to at least one dimension relationship (<<DimRelationship>>). Each (<<Dimension>>) has a non-empty set of hierarchies (<<Hierarchy>>) which represent a level of details for analyzing facts. Finally, each (<<Hierarchy>>) includes one or multiple aggregation levels (<<AggLevel>>). Authors have categorized dimensions, hierarchies, and aggregation levels into three groups: spatial, temporal, and thematic. A spatial dimension (<<SpatialDimension>>) expresses the location of factual data, whereas the temporal dimension (<<TemporalDimension>>) refers to the time when the data occurred. Lastly, a thematic dimension (<<ThematicDimension>>) holds context information.

A Spatial hierarchy (<<SpatialHierarchy>>) comprises of one or more spatial aggregation levels, whereas a temporal hierarchy (<<TemporalHierarchy>>) consists of one or more temporal aggregation levels. On the other hand. A thematic hierarchy (<<ThematicHierarchy>>) includes thematic aggregation level. A thematic aggregation level, also referred to as (<<ThematicAggLevel>>) consist of descriptive attributes. For example, a spatial aggregation level is commune, while a temporal aggregation level is time hour, and a thematic aggregation level is company.

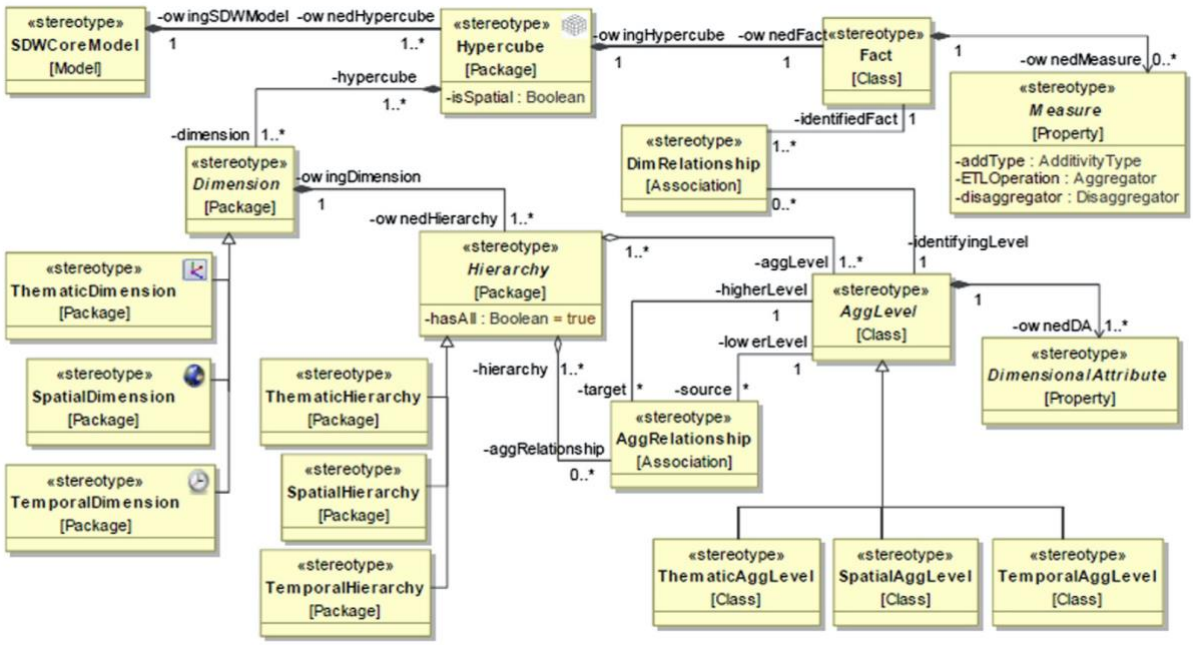


Figure 14 The UML metamodel of the SDW core model package (K. Boulil 2012).

1.4.2.3 Modeling ETL Process

(Trujillo 2003) proposed an approach to formalize conceptual model for ETL process in UML. This approach supports the user to design a complex ETL into a set of simple processes. They proposed to define each ETL mechanism as a UML stereotype as illustrated in Table 9. To simply representation of the ETL process, each stereotype has a graphical icon. For example, the aggregation mechanism characterized by Aggregation stereotype aggregates data based on some criteria. The ETL designer can define the grouping criteria and the aggregation functions such as SUM, AVG, MIN, and MAX in a note. Figure 15 represents a portion of an ETL process by using the Trujillo ETL design approach.

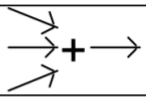
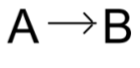
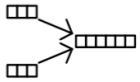
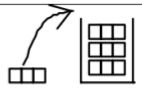
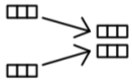
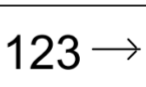
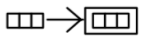
ETL Mechanism (Stereotype)	Description	Icon
Aggregation	Aggregates data based on some criteria	
Conversion	Changes data type and format or derives new data from existing data	
Filter	Filters and verifies data	
Incorrect	Reroutes incorrect data	
Join	Joins two data sources related to each other with some attributes	
Loader	Loads data into the target of an ETL process	
Log	Logs activity of an ETL mechanism	
Merge	Integrates two or more data sources with compatible attributes	
Surrogate	Generates unique surrogate keys	
Wrapper	Transforms a native data source into a record based data source	

Table 9 ETL mechanisms and icons of (Trujillo 2003).

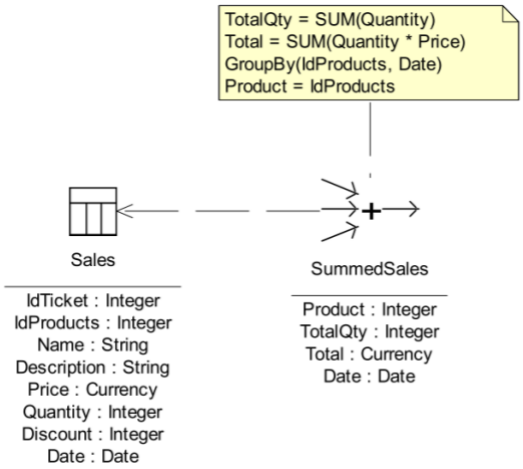


Figure 15 Aggregation example using the defined stereotype icons (Trujillo 2003).

1.4.2 Mediation System

Mediation system provides a uniform view of data from different sources (Wiederhold 1992). In the context of data integration, mediation system is usually used to facilitate the combination of data between different sources and formats. A mediation system includes mediated schemas (also called global schemas) and mapping techniques. The most popular techniques of mediation system are GAV (Global as View) and LAV (Local as View) (Halevy 2001). GAV is a definition of relations in global schemas based on relations in local schemas. LAV is a definition in local schemas on relations in global schemas. When a query q is submitted, the mediation system (i) uses the mapping (between global schema and sources) to rewrite the query q into a set queries Q and then (ii) executes the queries $q' \in Q$ on the corresponding data sources, (iii) gets the results from data sources, (vi) merges the result and return it to user.

GAV

The Global As View (GAV) is a technique for mediating between different data sources in the data integration. It involves creating a global view (also called integrated schema) of the data from the different sources, which can then be queried as if it were a single, unified data source (Halevy 2001). This allows for the data to be accessed and used in a consistent and unified way, regardless of the data sources. This approach is applied to the TSIMMIS integration system (Garcia-Molina 1997). In GAV method, the process of rewriting queries is handled by the mediator, which is responsible for mapping the data from the various sources into the common schema, and for resolving conflicts that may arise between the sources. GAV is especially useful when you have data spread in different places, different formats and different structures which need to be integrated for a common purpose. However, GAV has the disadvantage of requiring rewriting completely the mapping between local schemas and the integrated schemas whenever a local schema changes.

LAV

The Local As View (LAV) includes defining the local sources as a set of views made on the global schema (Halevy 2001). Unlike GAV, a change in a local schema only requires updating the mapping between these local sources and the global schema.

However, LAV includes a more complex query rewriting process. Some projects applied the LAV technique (T. Kirk 1995, Goasdoue 2000).

1.5 CEBA Project

The CEBA architecture, depicted in Figure 1, is designed based on the core requirement that it can handle diverse data flows without any prerequisite regarding their structure and content, while providing only minimal data analysis functionalities (Sarramia 2022). In terms of data management, the design incorporates four main functionalities: ingestion, storage, cataloging, and access. Additionally, it includes a high-level metadata management system, ensuring that datasets are accompanied by metadata at a minimum.

Regarding IoT system data, the architecture specifically considers low-output sensor networks such as LoRa (Augustin 2016) or SigFox (Sigfox 2023), characterized by low velocity. The system offers near real-time access to this data, as defined by the specific use case, along with access to historical and cold data. The ConnecSens project (ConnecSenS 2015-2020), funded by the European Regional Development Fund program of the European Union from 2015 to 2020 (ERDF 2023), has developed an open-source data collection platform using LoRa. This project serves as a key partner in the development of the IoT component within CEBA.

The CEBA is composed of the following layers:

- An ingestion layer that handles streaming data sources using Beats.
- A dual data storage layer that enables near real-time visualization and simple data querying. It utilizes Logstash, ElasticSearch, Grafana, and PostgreSQL.
- A long-term data storage layer provided by PostgreSQL and an AWS S3 Mesocenter service.
- A data access and discovery layer facilitated by a website designed with the Symfony framework and a GeoNetwork data catalog.

The CEBA architecture, displayed in Figure 1, consists of multiple servers that host one or several services, along with a reverse proxy for security purposes. The architecture includes the following servers:

- Sensor Data Server: This server receives and publishes data from LoRa sensor networks, which are managed by the ConnecSens project. It utilizes Chirpstack, Mosquito, and Paho library.
- Elastic Search Cluster: Comprising three servers, this cluster is responsible for indexing and retrieval of data.
- Transport/Transformation and Visualization Server: This server hosts services such as Filebeat, Logstash, and Grafana, which handle the transportation, transformation, and visualization of data.
- Web Server: This server acts as a web server to serve various functionalities.
- Metadata Generation Server: This server manages the generation of metadata using R and the geometa library.
- Data Catalog Server: Geonetwork is hosted on this server, which serves as the data catalog.
- DBMS Servers: Two servers are dedicated to the Database Management System (DBMS) PostgreSQL.
- Additionally, the AWS S3 service is hosted and managed by Mesocenter Auvergne.

An ingestion multi-pipeline near real-time system has been developed using the Elastic Stack to enable instant data visualization and handle diverse use cases (refer to Figure 16). This pipeline is capable of handling data inputs from different sources such as files, databases, and data flows. In the case of data from sensor network deployments, ingestion is accomplished by reading daily created files that are continuously updated throughout the day. Technically, once a data file is created or updated with new data, Filebeats, a lightweight shipper, sends the data to a data collector called Logstash, both of which are components of the Elastic Stack (Elasticsearch, ELK 2020). Logstash functions as a data streaming pipeline, receiving data from various sources (Input), cleansing, and enriching each data point with relevant information (Filter), and directing the data to the appropriate CEBA server (Output).

The data routing to appropriate storage locations is accomplished during indexing, utilizing an index naming convention that incorporates the experiment name, the end node's name, and the measurement date. This approach allows for the effortless isolation and retrieval of variables, facilitating the generation of time series. This strategy offers several benefits, including the storage and preservation of raw data via files, as well as near real-time presentation to the user through indexing. Additionally, it enables swift reindexing in the event of bugs or errors in data transmission.

The CEBA can efficiently integrate new data providers by ingesting data from various sources, including databases through Logstash. Additionally, MQTT streams can be ingested using Filebeat, expanding the capabilities of data ingestion.

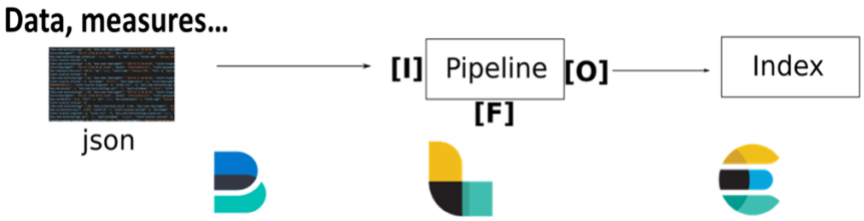


Figure 16 Schematic representation of generic ingestion pipeline (Sarramia 2022).

1.6 Discussion and Summary

This thesis focuses on the integration and analysis of environmental sensor data, aiming to provide systems and techniques to overcome challenges cited in this chapter. We discussed in this chapter the main topics related to this research. We explained the nature of data studied in this research which is mainly environmental sensor data generated by sensors and other IoT devices. This data is mainly characterized by the location, timestamp, and measurements. We described the standardization of sensor data access by Open Geospatial Consortium (OGC) and provided a comparison of the two models: the SensorThings API (STA) and the Sensor Observation Service (SOS). We included a review of several topics related spatial vector data including different database models used for storage and management of spatial vector data. We highlighted the benefits of NoSQL data models and described in detail the ELK stack.

Additionally, we presented the data processing frameworks Apache Spark and its adaptation for spatial vector data, Apache Sedona. Finally, we discussed the techniques for data integration.

Our first contribution is related to spatial data warehouse and the ELK stack. Previous literature has highlighted the advantages of NoSQL data models for spatial data, regarding to query capabilities and performance. However, no prior work has used and evaluated Elasticsearch for data warehousing and analytics in the context of streaming geo-referenced sensor data. Hence, we propose a method and a dedicated architecture to represent and query a spatial data warehouse (SDW) model with ELK stack. We implement multidimensional models in Elasticsearch and propose analytical dashboards with Kibana. We also evaluate the proposal with respect to several aspects.

Our second contribution aims to overcome the limit of data warehousing. Existing literature studied data integration extensively, but the works focused on spatial data integration do not address both streaming and large data sets. Additionally, existing work for data integration in the context of Big Data does not consider neither streaming nor spatial data, and data processing frameworks lack support for integrating different data sources under a uniform schema. It is indeed difficult to handle integration of a large number of heterogeneous data sources. Therefore, we propose stream data integration with geospatial capabilities for real-time analytics of environmental data.

In a final chapter, we address the modeling of data warehouses for sensor data. We propose a generic multidimensional model for data warehousing SensorThings-compatible source.

CHAPTER 2 A NEW APPROACH BASED ON ELK STACK FOR THE ANALYSIS AND VISUALIZATION OF GEO-REFERENCED SENSOR DATA

The results obtained in this chapter have been presented in two scientific publications: the 7th International Conference on Geographical Information System Theory, Application and Management (Ngo 2021), and the Springer Nature Computer Science journal (Ngo 2023).

2.1 Introduction

Data warehouses are a well-established methodology and system for conducting data analysis, as described in Section 1.4.1 of Chapter 1. This approach involves aggregating data from various sources, integrating it, and storing it in a centralized data repository. Traditionally, data warehouses were built on top of relational databases, which provided a robust and reliable foundation for data analysis. However, recent advancements in data management have led to the emergence of NoSQL databases, which offer unique advantages in terms of query capabilities, query answering performance, scalability, and schema changes. As such, the literature has begun exploring the benefits of NoSQL databases as an alternative to traditional relational data warehouses, as highlighted in (Bicevska 2017). NoSQL databases have also emerged as a reliable option for developing geospatial information systems, as highlighted in (Guo 2020). They highlight the ability of NoSQL systems to manage big spatial data that exceeds the capacity of traditional databases. These capabilities leverage NoSQL databases as an interesting choice for handling large-scale geospatial data, which requires efficient and scalable storage, retrieval, and analysis mechanisms.

In this chapter, we address the challenge of implementing a data warehouse of geo-referenced data in a NoSQL system. Our proposed solution includes a dedicated architecture and method for representing and quering a spatial data warehouse (SDW) model using the ELK stack (ELK stack 2022). We note that we provided a

comprehensive description of the ELK Stack in Section 1.3.3.2 of Chapter 1. The choice of the Elastic stack was made because it is a distributed solution, redundant at the data level and scalable. It has a comprehensive query language and is efficient in terms of query processing. Elastic stack is well suited for sensor data and spatiotemporal analytics.

Our contributions include a model or SDW architecture based on the ELK stack, and a performance evaluation for this type of architecture. To illustrate our approach, firstly we demonstrate the implementation of two multidimensional models relevant to environmental sensor data in Elasticsearch. Then, we present a streaming ETL component called IAT (Integration and Aggregation Tool) for loading sensor data into Elasticsearch. Finally, we show the comprehensive query capabilities of Elasticsearch and its ability to handle a growing number of dimensions and data, offering a promising approach to address the challenge of implementing a spatial data warehouse in a NoSQL system. Our proposed SDW model and ELK stack architecture provide a powerful and scalable solution for managing and analyzing large-scale geo-referenced data in a NoSQL system.

Our proposed methods are tested on CEBA, an Environmental Cloud for the Benefits of Agriculture project (Sarramia 2022). The main objective of CEBA is to afford a better understanding of changes in environment quality over time. CEBA consists of several components, including Elasticsearch, and collects data primarily from various sensors deployed in Auvergne (France) for scientific projects such as (ConnecSenS 2015-2020). The main data flows of CEBA consist of (i) collecting data primarily from sensor sources, (ii) ingesting the collected sensor data in Logstash via Beats shippers, and then (iii) storing the collected data, mainly in Elasticsearch database. However, CEBA lacks an analytical component to analyze the ingested environmental sensor data.

To conclude, the contributions of this work are:

- A data warehouse architecture for geo-referenced sensor data based on the ELK stack, which is composed of IAT for the ETL process.
- An illustration on two multidimensional models for geo-referenced sensor data.
- An example of the Kibana dashboard for the proposed multidimensional models.

- Evaluations of Elasticsearch for analytical queries on the models and IAT with our benchmark dataset.

The chapter is organized as follows: in Section 2.2, we present our work and the architecture composed of the ELK stack, along with the use case for analytical queries and the dashboard on Kibana. We then present the evaluation of Elasticsearch and IAT with our benchmark dataset, followed by a comparative evaluation with MongoDB.

2.2 How to Represent a Query and Implement a Spatial Data Warehouse with Elasticsearch?

In this section, we present the main components and steps to represent and build a data warehouse on top of Elasticsearch. We drive the presentation with the data from our use case, which involves a set of sensors deployed in natural areas, that collect environmental measurements.

2.2.1 Sensor Data Representation

In our proposed approach, we utilize the ISO/OGC Observation and Measurement Standard, known as O&M (refer to Section 1.2.2.2 of Chapter 1), to describe our sensor data model. Figure 17 depicts the conceptual model for the observations. The studied observations includes four sensor data sources of CEBA, namely Auzon, Aydat, Montoldre, and Zatu. Each data source has different measurements. The observation classes of our project sensors inherit from the OM_ComplexObservation class, which itself inherits from the OM_Observation class.

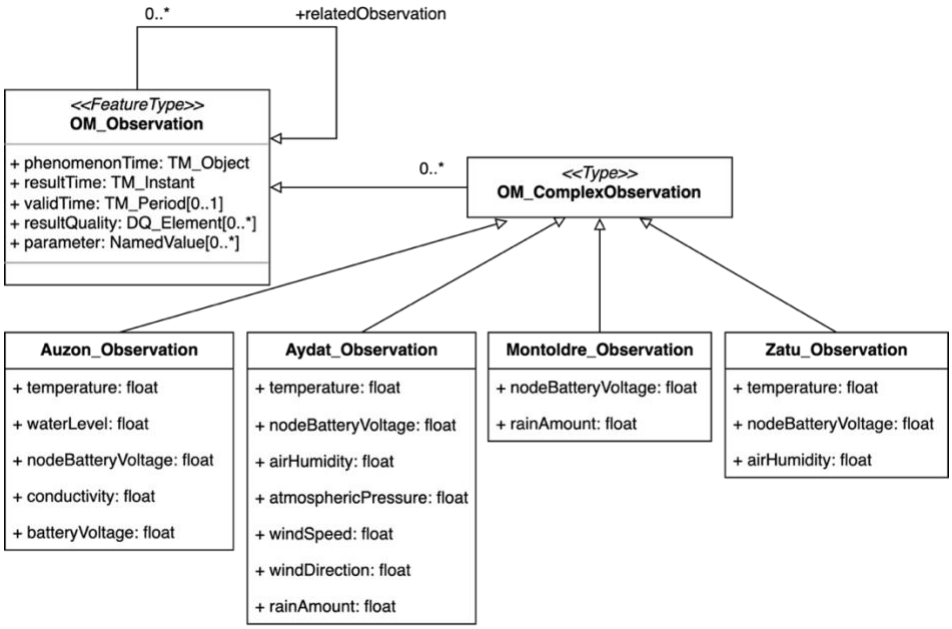


Figure 17 Class diagram of OM_ComplexObservation conceptual model of four sensor projects in CEBA.

2.2.2 Multidimensional Models

To design multidimensional models, the UML profile proposed by (Pinet 2010) specifically for environmental data warehouses is employed. In this profile, classes serve as representations of both facts and levels of dimensions. A fact class comprises measure attributes that are the focus of analysis, while a dimension is a hierarchy of member classes, each of which represents a level of analysis in the dimension. Associations between members are established through aggregated associations, which are depicted using an arrow '<-' in the diagrams. At least one class of member in a dimension must be linked to a fact class.

In model A, displayed in Figure 18, the fact consists of a set of measurement types that allows users to analyze environmental factors across three dimensions: (i) *time* at four levels of hierarchy (hourly, daily, monthly and yearly), (ii) location with four levels of hierarchy (geographic coordinates, communes i.e., French towns or villages, French departments, and French regions), and (iii) information with two levels of hierarchy

(physical sensor or device that collected the measures and scientific project during which the data were collected).

In Model B, displayed in Figure 19, the fact is either geographic coordinates or measures. This model enables users to aggregate the facts with respect to three dimensions: (i) time, at four levels of hierarchy (hourly, daily, monthly and yearly), (ii) location, with three levels of hierarchy (communes i.e., French towns or villages, French departments, and French regions), (iii) information, with two levels of hierarchy (sensor device information and scientific projects).

In a data warehouse, aggregations that one can compute are based on the facts and the dimensions structure. The choice of model structure can result in different types of queries. In Model B, where the fact can be either geographic coordinates or measures, it allows for queries that target the location fact, such as determining the area covered by each sensor device every hour. In Model A, aggregations are only based on numerical values, with geographic location considered as a dimension. However, in Model B, spatial aggregations on a geo-referenced attribute are possible due to the fact that geographic location is treated as a fact rather than a dimension.

Throughout the remainder of this chapter, we refer mainly to the first model presented as model A. Having described the initial data collected from the data sources and the data cube models, we now proceed to present the architecture of our system, for collecting and transforming data from the sensors to comply with the data cube models. Additionally, we provide the aggregation queries for each model in Section 2.2.4.1, and the aggregation queries for both models using Kibana, as outlined in Section 2.2.4.3.

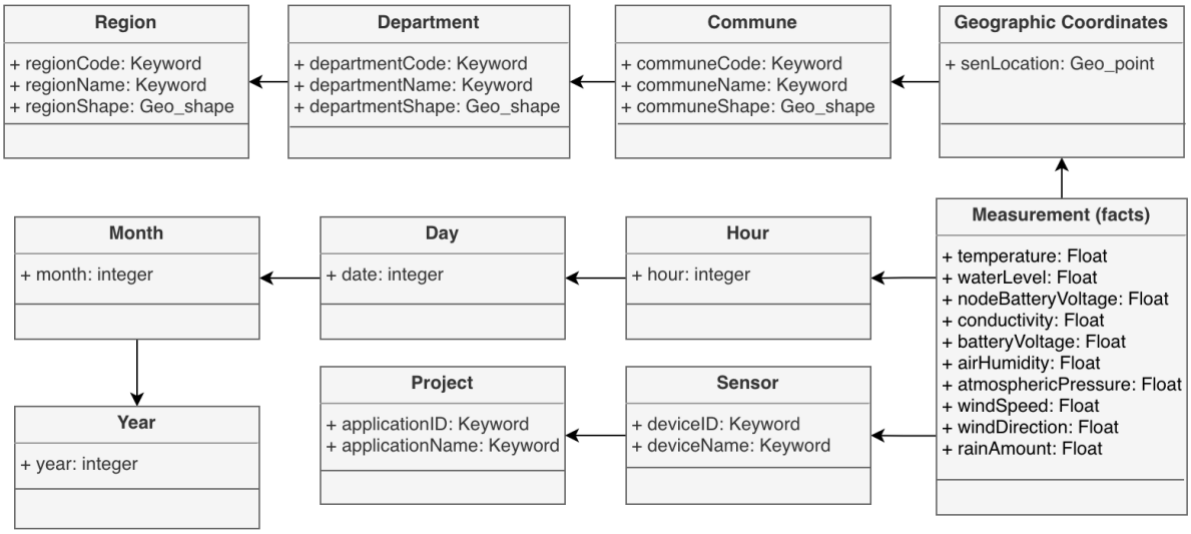


Figure 18 Multi-dimensional conceptual model for the measurement fact (Model A).

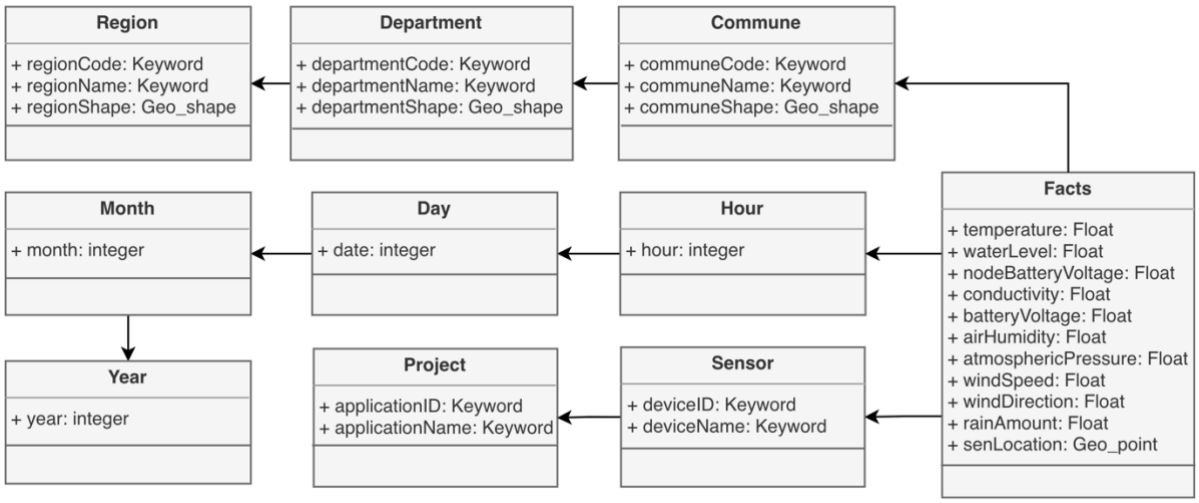


Figure 19 Multi-dimensional conceptual model: geo object and measurement as fact (Model B).

2.2.3 System Architecture

In this section, we present our proposed system architecturedisplayed in Figure 20. The architecture consists of two main parts: (i) the ELK stack, which includes the pipeline Beats, Logstash, Elasticsearch and Kibana, and (ii) the Integration and Aggregation Tool (IAT), which is mainly responsible for populating the data cube in Elasticsearch (ES).

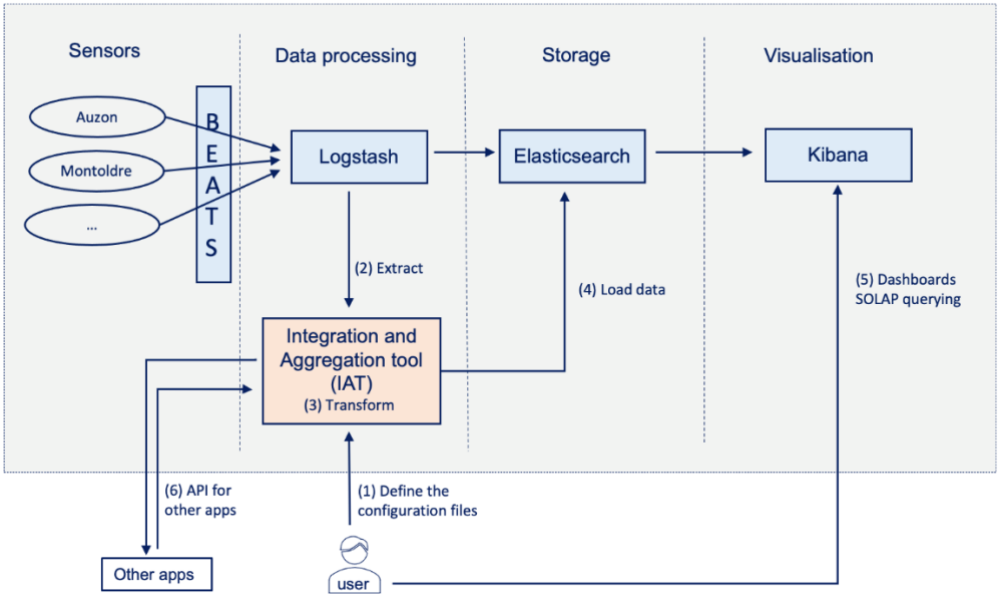


Figure 20 System architecture.

In more detail, sensor data is collected by Beats and then transmitted to Logstash, which listens continuously for new records generated by sensors and receives them from Beats. We use Logstash to standardize the data coming from different sensors to ensure consistency across the board. For instance, each sensor family may have its unique timestamp column pattern or use different terminologies for measures, which may lead to discrepancies in data. Furthermore, we suggest using Logstash to remove metadata fields to ensure that only relevant data is processed. Next, IAT extracts the data from different sources in Logstash and transforms them in order to comply with the data cube models. The resulting output is stored in an Elasticsearch index. The IAT process is user-configurable, which describes (i) the models (ii) the input data, as well as (iii) the transformation process. The functionalities of IAT that are relevant to our use case are (i) fields mapping, (ii) time window aggregation, and (iii) integration with external sources. These functionalities of IAT are also explained in Section 2.2.4.2. Finally, we use Kibana to create dashboards for visually analyze the data.

2.2.4 Data Cube, ETL and Queries

In this section, we provide a detailed representation and implementation for the data cube and the ETL system, as well as their corresponding data visualization and textual queries.

2.2.4.1 Elasticsearch Data Cube and Query Workload

We propose a streamlined approach to store the data cube by utilizing a single Elasticsearch index. This method involves mapping the attributes in the dimensions and facts of the models to fields in the ES index. Our use case demonstrates that both models can be efficiently implemented within a single index. There are two primary reasons for storing all the data cube in one index:

- Firstly, Elasticsearch does not perform well with join queries (Pilato 2017). As a NoSQL system, it is not very efficient to join data from different indices (although join queries do exist in ES). By keeping all the data in one index, we avoid the performance issues that may raise from attempting to join data from multiple indices.

- Secondly, ES is vertically scalable, meaning that larger indices do not significantly impact ES query performance. In fact, fewer indices with more data are generally recommended over many indices with little data. As a result, storing all the data cube in a single index is a more efficient and practical approach.

Figure 21 is a snippet of the ES index schema which stores the data cube. As an example, field deviceID is mapped to deviceID in the dimension hierarchy “Sensor” in both models, as detailed in Section 2.2.2.

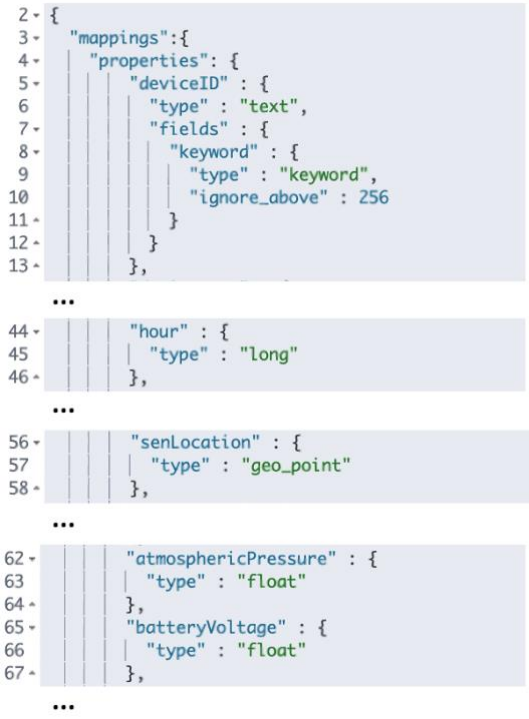


Figure 21 A snippet of the target index schema.

Each record of the data cube is a document in the ES target index. A snippet of a document from the target index (data cube) is illustrated in Figure 22. As we can see, it contains all of the attributes, e.g., applicationID, year, senLocation, and windSpeed.

Figure 22 A snippet of a document in the target index

In the following, we present the analytical queries. Aggregation queries in ES are a combination of two types of queries:

- Bucket aggregation query: given a set of documents in an ES index, this query returns one or more buckets of the documents grouped by one and only one field. Each document in the index belongs to at most one bucket. This query is equivalent to a GROUP BY in SQL semantics.

- Metric aggregation query: given a set of documents, this query aggregates the values of an index field by a metric function.

To achieve a GROUP BY operation with respect to several fields, ES query language allows to compose several bucket queries in a nested manner.

```
GET target_index_1year\_search?size=0
{
  "aggs":{
    "2":{
    "terms":{
      "field":"hours",
      "order":{
        "_key":"asc"
      },
      "size":24
    }
  },
    "3":{
      "terms":{
        "field":"deviceName.keyword",
        "order":{
          "_key": "desc"
        },
        "size":20
      },
      "aggs":{
        "temperature":{
          "stats":{
            "field":"temperature"
          }
        },
        "airHumidity":{
          "stats":{
            "field": "airHumidity"
          }
        },
```

Figure 23 Snippet of a query on two dimensions for model A.

We show a snippet of a query for Model A (refer to Figure 23). This aggregation query groups documents by (2) time dimension (and more specifically the hour attribute), (3) information dimension by the device name field. It then aggregates the values of each measure in Model A and computes some statistical functions, such as temperature and

air humidity, as depicted in the figure. A visualization example of this query for computing the average air humidity measure on the Kibana dashboard, is presented in the visualization 3 in Section 2.2.4.3 (refer to Figure 24).

2.2.4.2 Data Integration with IAT

IAT is a dedicated streaming ETL (Extract, Transform, Load) application, specifically designed to integrate sensor records into a data cube, named the target index. For our use case, we have defined three essential functionalities for the transformation process: (i) field mapping, (ii) time window aggregating, and (iii) integration with other data sources via API. We detail the functionalities of IAT below:

- Field mapping is responsible for mapping fields between data sources and the target ES index.
- Time window aggregation is used to continuously aggregate sensor records based on their timestamp. This functionality is particularly useful when data sources have different frequencies of record generation. The user defines the configuration for time window aggregation. IAT then aggregates data within the configuration window of time size. For example, if the user configures IAT to aggregate sensor data in intervals of one hour, it will compute the average of all measurement fields.
- IAT can integrate sensor data with additional data sources to enhance analytics, such as commune and department in Model A and Model B.

The process is driven by a user configuration where the user specifies the parameters of the functionalities described above. We share our implementation of IAT in GitHub[1].

[1] GitHub: https://github.com/AnnaNgo13/es_etl

2.2.4.3 Dashboard on Kibana

After populating the target index (data cube), we propose to design a dashboard on Kibana to visualize interesting analytical queries based on model A described in Section 2.2.2. Figure 24 shows a snapshot of this dashboard with the following visualizations:

- Visualization 1 (top left): This visualization can be used to detect if sensors having the same application id (i.e., concerning the same Project), deployed in the same area and under similar conditions, collect highly divergent values, which could be due to a defective sensor. To achieve this, we (i) filter by application id and month, (ii) group by device names, and (iii) aggregate the temperature measurement. We can see that one device detects low temperatures compared to others for three days.

- Visualization 2 (bottom left): In this visualization, we want to see the impact of battery voltage on the measurement values. If the battery loses power suddenly, the measurement may become inaccurate. We can see in the visualization that the drop in battery voltage occurred at the same time as a drop in temperature. We can conclude that the battery issue impacts the temperature measurement.

- Visualization 3 (top right): In this visualization, we want to monitor the values of some specific measurements.The values of a measurement should be in a specific interval. Otherwise, the measurements should be inaccurate. In this visualization, we see the variation of air humidity over time aggregated by device. We can see that the values go below 60% (the expected minimum value) during a time interval.

- Visualization 4 (bottom right): In this visualization, we aggregate sensors by geo grid. By hovering over some cells, we get the aggregation of all measurements collected by the sensors of this cell.

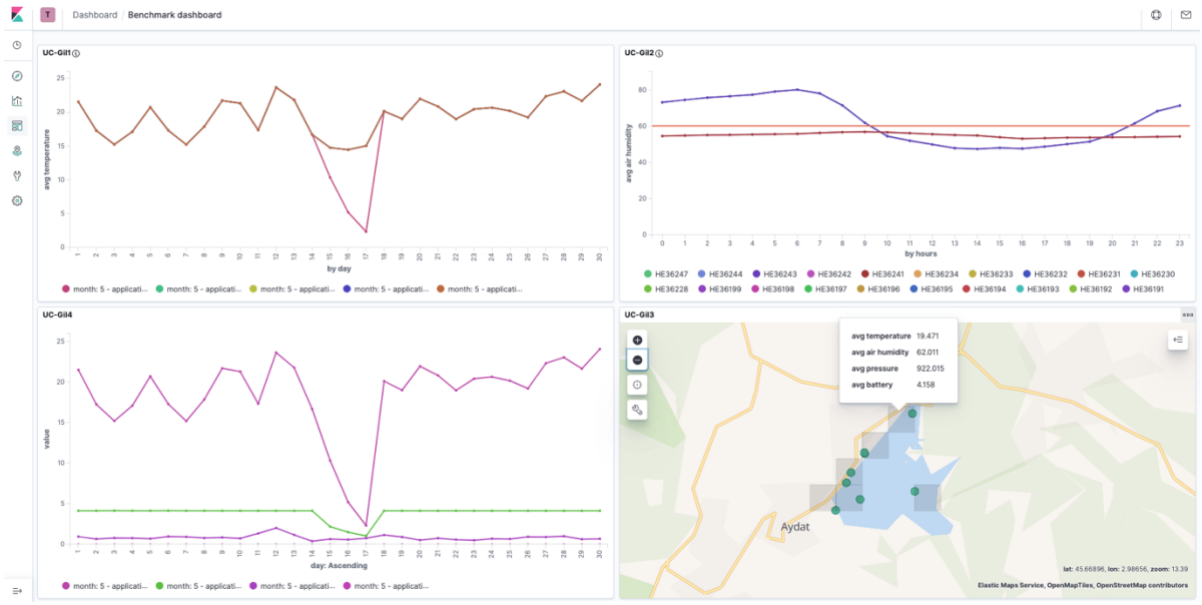


Figure 24 Benchmark dashboard.

Each visualization is presented on a higher scale in Appendix A (Figure 50, Figure 51, Figure 52, Figure 53). The steps to create a visualization on Kibana are in Appendix B.

2.2.4.4 Textual Queries

Figure 25 shows an example of an analytical query related to Model B. The left-hand side of the figure is the query in ES query language, while the right-hand side is a snippet of its corresponding result. Recall that the ES query result is represented in JSON format. This query aggregates documents by two buckets: (i) by dimension level hour and (ii) by dimension level sensor name. It then operates two spatial functions on the senLocation field in the aggregated documents: (i) geo centroid and (ii) geo bounds. Geo centroid computes the weighted centroid from all coordinate values for the senLocation field. Geo bounds computes a bounding box containing all geo values for the senLocation field.

```
GET target_index\_search?size=0
{
  "aggs": {
    "agg_time":{
      "terms":{
        "field":"hours",
        "order":{
          "_key":"asc"
        },
        "size":24
      }
    },
    "agg_devices": {
      "terms": {
        "field": "deviceName.keyword",
        "size": 10
      },
      "aggs":{
        "agg_centroid":{
          "geo_centroid": {
            "field": "senLocation"
          }
        },
        "agg_bound": {
          "geo_bounds": {
            "field": "senLocation"
          }
        }
      }
    }
  }
}
```

```
{
  "key" : "HE36241",
  "doc_count" : 8783,
  "agg_centroid" : {
    "location" : {
      "lat" : 45.761224958114326,
      "lon" : 3.1108499877154827
    },
    "count" : 8783
  },
  "agg_bound" : {
    "bounds" : {
      "top_left" : {
        "lat" : 45.761224958114326,
        "lon" : 3.1108499877154827
      },
      "bottom_right" : {
        "lat" : 45.761224958114326,
        "lon" : 3.1108499877154827
      }
    }
  },
```

Figure 25 Query in ES query language and snippet of its result for Model B.

2.3 Evaluation and Experiment

In this section, we evaluate the performance of our solution based on Elasticsearch for sensor data warehousing and analytics and demonstrate the ability of our system to handle real-time data collected by sensors for analytical queries. Our primary objective is to analyze the scalability of the system, i.e., how the execution time performance changes as the volumn of data increases. Specifically, we measure the query execution time performance of the Elasticsearch engine, along with its memory and disk consumption, and we evaluate the data ingestion through IAT. To evaluate the query execution time, we run workloads based on the models presented above and measure their execution time, as well as the disk and memory usage of Elasticsearch. Moerover, we compare Elasticsearch to its direct competitor MongoDB with respect to analytical queries execution time

2.3.1 Dataset

For the purpose of evaluating our system, we built a large dataset including collected measurements of four different projects, i.e. Auzon, Aydat, Montoldre, Zatu.

The Auzon project focuses on the Allier ecosystem services, particularly the Allier river and its hydraulic appendages in France. These areas are vital for biodiversity, denitrification, recreation, and fishing leases. Monitoring the hydraulic appendages is crucial as they serve as an indicator of changes in the river system, which is dynamic and experiences temporary reconnection during floods and regular restoration actions. The project utilizes the ConnecSens observation system, which deploys a network of sensors to continuously monitor environmental variables such as water level, temperature, and conductivity in the Allier river and its associated alluvial aquifers.

Aydat is a lake in south-central France, covering an area of 60 hectares and having a small watershed of about 16.8 hectares. It has a maximum depth of 15 meters. The lake is eutrophic and frequently experiences cyanobacterial blooms. It is part of the OLA network, which focuses on studing lake ecosystems in France. OLA stands for "Observatoire des Lacs" in French, and its English translation is Lakes Observatory. To understand the lake's hydro-ecological processes and address the issue of cyanobacterial proliferation, various sensor systems such as one HYDROLAB HL7 multiparameter sonde, and three Aquatroll 200 data loggers are deployed. HYDROLAB HL7 multiparameter sonde comprises eight sensors (a conductivity sensor, Hach LDO® Dissolved Oxygen Sensor, temperature sensor, turbidity Sensor, chlorophyll-a sensor, blue and green algae sensor, rhodamine sensor, and finally a pressure sensor for water depth measurement). These sensors use different transmission technologies (LoRa using ConnecSens technology with a gateway connected to the internet for Aquatroll, Short Message Service for HYDROLAB HL7) at least hourly. The collected data is sent to CEBA, where reference datasets are created by projects associated with the observatory.

The Montoldre project is focused on studying the functioning of agrosystems at the Montoldre site, which includes a 100-hectare farm used for research on sensor networks. As part of the ConnecSens project, the goal is to deploy sensors to improve

understanding of water dynamics in agricultural plots. This involves implementing a network of soil moisture and temperature sensors, along with a weather station, on selected plots. The data collected from these sensors provides valuable information about soil water content and its availability for corn crop growth.

The ZATU project (Zone Atelier Territoires Uranifères) is a part of the french LTSER network since 2015, uses CEBA as a data repository and dataset catalog (LeRéseauDesZonesAteliers 2023). ZATU's research observatory, located in France, focuses on studying the connections between ecology, society, and radioactivity. The observatory generates diverse data from various disciplines, inlcuding a project involving the development of a wireless radon sensor. In the future, the obervatory plans to deploy multiple wireless sensor networks to measure variables such as air randon (an exmample ad-hoc system has been designed by the LPC laboratory, this is a physics laboratory of Clermont Ferrand, based on the Algade AER sensor system), water level, temperature and flow (using HYDROLAB HL7 multiparameter sonde and Aquatroll 200 data loggers), and weather with the data being uploaded and stored in CEBA[2].

The initial dataset obtained from the four projects comprises metadata in its original format. Nonetheless, the dataset exhibits a considerable number of empty records and incomplete measurement collections, particularly in the Auzon project dataset. Many data frames related to system functioning, such as reboots, radio signal performance, and battery levels, are not considered in this dataset. However, the interesting point is field name reflects the meaning of the data it contains As a result, we have devised the subsequent procedure to obtain a suitable dataset for the purpose of benchmarking our proposal:

1. Remove irrelevant fields that do not provide valuable insights for the dataset, such as "adr" and "fPort" configuration fields. For the Auzon dataset, we retain 14 out of 49 fields, which accounts for approximately 30% of the relevant data.

[2] All these sensors have been deployed by January 2023 but were not available at the time of the study.

2. Identify the relevant measurement code(s) within the dataset.
3. Drop empty rows in important columns and exclude uninteresting measurements from the analysis.
4. Eliminate duplicated records based on a unique key computed by functional dependencies. For example, for Auzon dataset, the key is (devEUI, applicationID, data-node-timestampUTC, data-CNSSRFDataTypeId).
5. Merge rows based on the unique key or longitude and latitude.
6. Improve the understanding of field meanings and subsequently remove unnecessary fields that do not contribute to the analysis.
7. Further refine the dataset by eliminating duplicate rows and increasing the data frequency.
8. Augment the dataset to cover a four-year period, providing a broader temporal perspective.

By implementing this process, we aim to gain a comprehensive understanding of the data and improve its quality for subsequent analysis and interpretation across the four projects.

Consequently, in the dataset, each sensor device contains approximately 1440 records for every day, i.e., one record per minute. Table 10 describes the dataset built to evaluate our system. The properties of each dataset are shown, e.g., the number of collected records over four years, measurement columns, and the number of devices. The columns related to dimensions are listed in Table 11. The dataset is available in a public data repository[3].

Data source (Projects)	Auzon	Aydat	Montoldre	Zatu
Number of measurement columns	5	7	2	3
Number of devices	6	7	9	8
Number of dimension levels	14	14	12	10
Number of collected records	12501840	13910400	15552000	16911360
Size	5.12 GB	5.04 GB	5.28 GB	4.56 GB

Table 10 Dataset summary over 4 years.

[3] https://drive.google.com/drive/folders/1ATdzq_p-jwrhPLyWkrfE_O8nCE8LD6s4?usp=sharing.

Common columns
Device ID Device name Application ID Application name Data-node-latitude Data-node-longitude Data-node-timestamp

Table 11 Common columns.

2.3.2 Hardware

We ran our experiments on a Linux Ubuntu machine with 16 GB RAM and a 6-core Intel Core i5 CPU 8400 and 500 GB disk space. We used ELK stack version 7.6.0, and Elasticsearch with one node and one shard, and we used a single node MongoDB version 4.4.12.

2.3.3 Evaluation of IAT

As noted in Section 2.2.4.2, IAT consists of a pipeline that ingests data and transforms it to follow the schema of the target index in Elasticsearch that represents the data cube. It consists of three parts: (i) transforms the columns, (ii) aggregates records by time, and (iii) adds additional information through external sources. These operations are described by users in a configuration file. In the context of evaluation and to stress IAT, we ingest the whole dataset presented above in one go, to evaluate its processing time. The IAT source code as well as the mapping files for each data source are available in GitHub (refer to Section 2.2.4.2). Figure 26 reports the time to pipeline data with respect to different data intervals. We run this experiment with the Auzon data source. We omit the other data sources as the results are similar. Generally, we can see that the evolution of the processing time is linear with respect to the time intervals, which shows that the processing time of one record is almost constant. The average processing time of one record is around 2 milliseconds (ms).

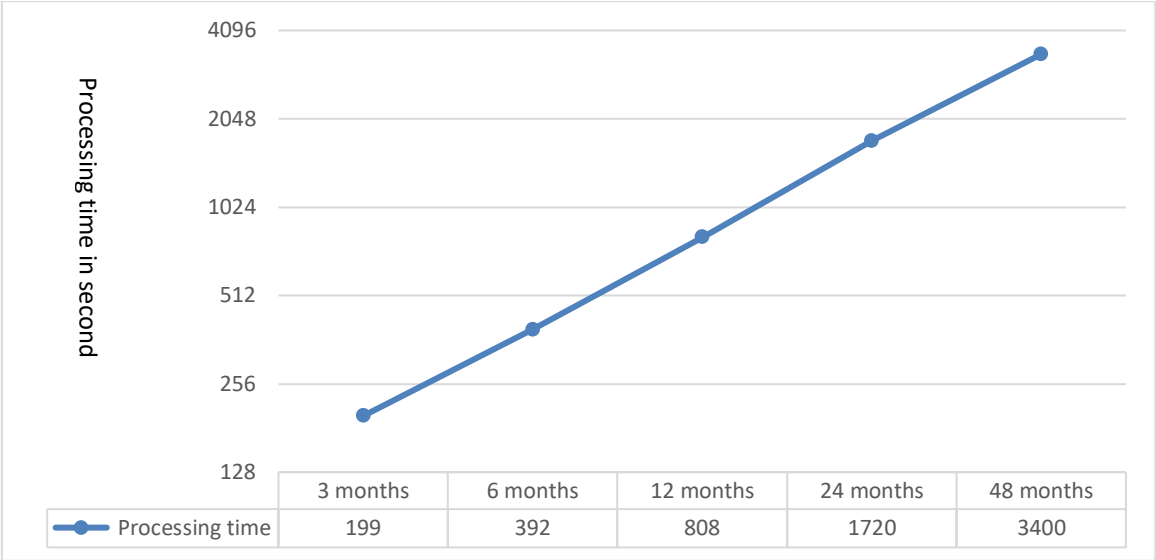


Figure 26 IAT pipelining processing time.

2.3.4 Evaluation of Elasticsearch

In this section, we evaluate Elasticsearch performance in testing spatio-temporal analytical queries for geo-referenced environmental sensor data. For each model described in Section 2.2.2, we run its corresponding workload 10 times and we report the performance indicators. For query execution time, we consider the value returned by Elasticsearch in the query result body. The results confirm that ES is a reliable tool for analytical queries in the context of environmental data. Note that the queries are executed on the top of the target index which was populated by IAT. For this experiment, we have configured IAT to aggregate the sensor records by windows of 60 minutes. We show in Table 12 the characteristic of the target index. The number of fields of the target index is 20 including 10 measurement fields. The total number of documents stored in the target index is 973089 with a total disk usage of 86 MB. The smaller disk usage is due to the ES indexing and data compression. A dump file of the target index content in JSON files is four times greater.

Number of fields	20
Number of measurement columns	10
Physical storage by ES	86 MB
Number of documents	973089

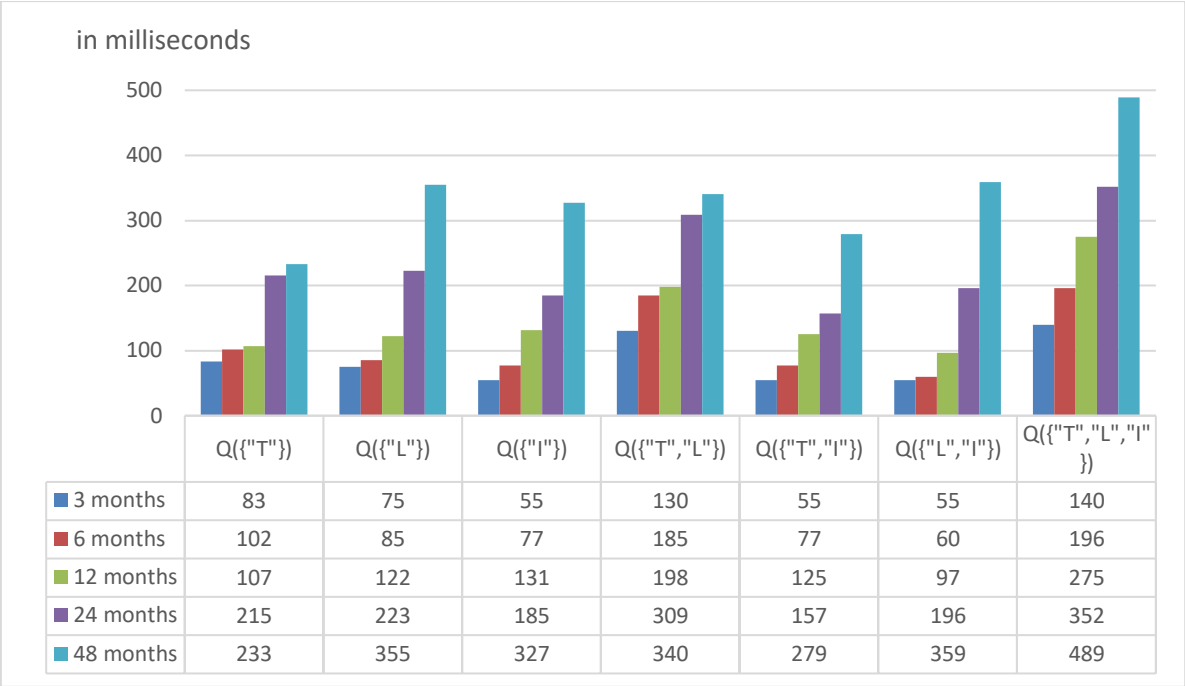
Table 12 Size of target index with 4 years of data.

As explained in Section 2.2.4.1, the ES query engine provides two types of aggregation: (i) bucket aggregation and (ii) metric aggregation. A bucket aggregation query consists in grouping documents (records) by some fields while a metric aggregation query performs a function on the field values of a set of documents. Hence, a general analytical query consists of a composition of one or more bucket queries and only one metric query.

For a fair and complete evaluation of ES query execution performance, we present query patterns. Given a set of dimensions X of a multidimensional model, a query pattern *Q(X)* represents all aggregation queries that include a bucket query (i.e., Group By) for each dimension in X. For example, Q({“Information”,“Location”}) groups all queries that group documents by both Location and Information dimensions. For the following results, we evaluate ES query execution performance with respect to query patterns. The execution time of a query pattern is the average execution time of queries related to that pattern.

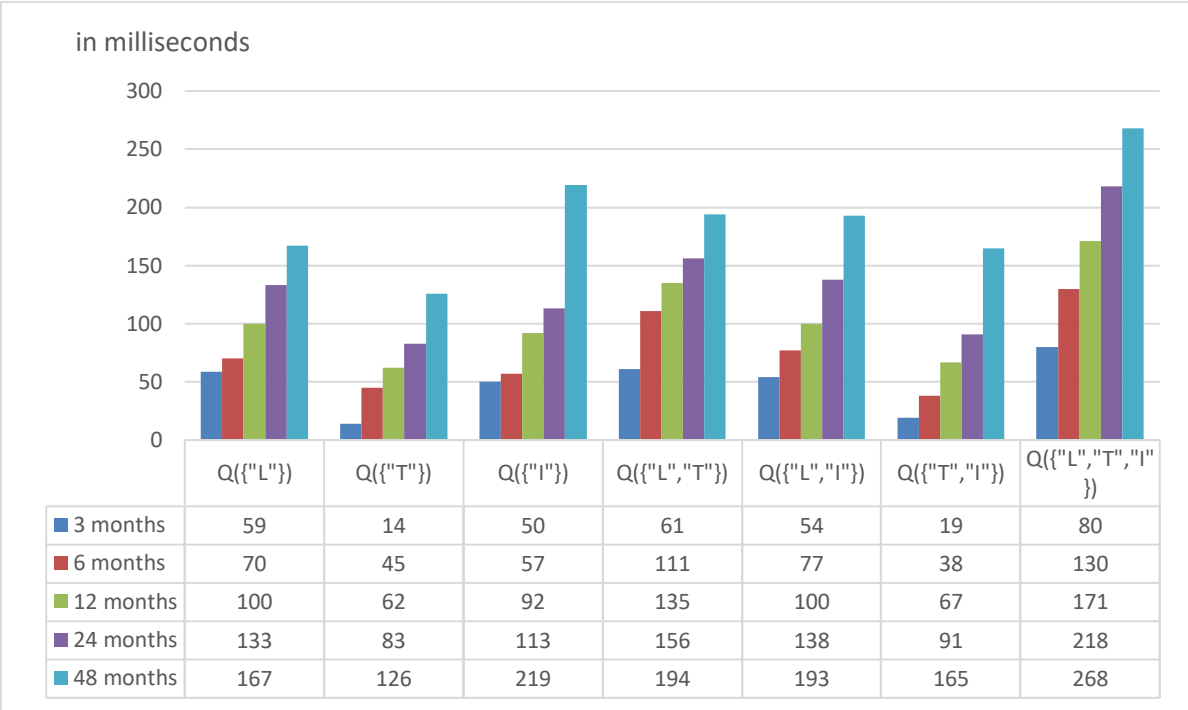
Therefore, we compare the query execution performance by: (i) different query patterns, (ii) dataset size (time period of the dataset), and (iii) different models. Figure 27 and Figure 28 display the execution time of queries for respectively Model A and Model B for the different query patterns. The letter “T” is for the “Time” dimension, “L” is for the “Location” dimension, and “I” is for the “Information” dimension. Note that queries are executed without using the cache or pre-computed queries. As Elasticsearch engine natively uses a query cache, the execution time reported by ES can often be 0 ms. Hence, we clear the cache after each query execution to avoid misleading results.

Generally, we can see the same behavior for the two models. The average execution time is slightly increasing with the number of bucket queries. For queries with two buckets, we see that $Q(\{"T","L"\})$ in Figure 27 takes more time than other queries. This query pattern groups documents by both the Time and the Location dimensions. Bucket queries with these two dimensions take more time than those with the Information dimension because the latter contain string attributes (encoded as keywords in the index mapping) which are indexed by ES better than any other data types.



	Q({"T"})	Q({"L"})	Q({"I"})	Q({"T","L"})	Q({"T","I"})	Q({"L","I"})	Q({"T","L","I"})
3 months	83	75	55	130	55	55	140
6 months	102	85	77	185	77	60	196
12 months	107	122	131	198	125	97	275
24 months	215	223	185	309	157	196	352
48 months	233	355	327	340	279	359	489

Figure 27 Execution time for Model A by query pattern.

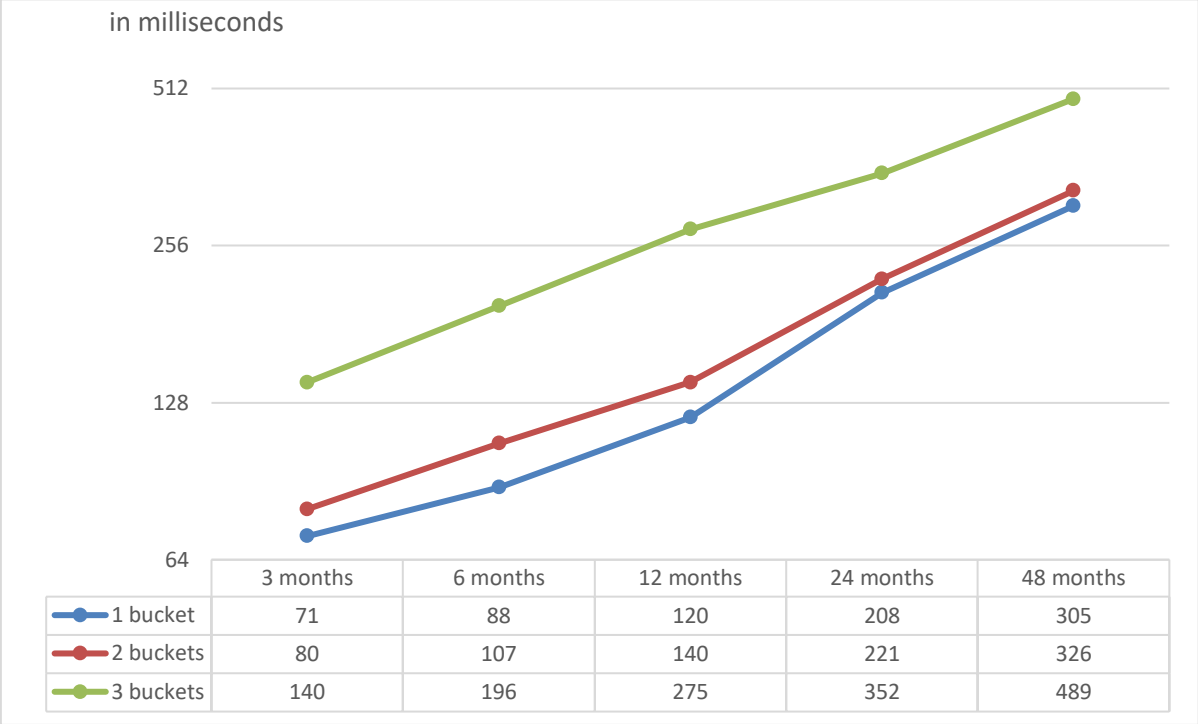


	Q({"L"})	Q({"T"})	Q({"I"})	Q({"L","T"})	Q({"L","I"})	Q({"T","I"})	Q({"L","T","I"})
3 months	59	14	50	61	54	19	80
6 months	70	45	57	111	77	38	130
12 months	100	62	92	135	100	67	171
24 months	133	83	113	156	138	91	218
48 months	167	126	219	194	193	165	268

Figure 28 Execution time for Model B by query pattern.

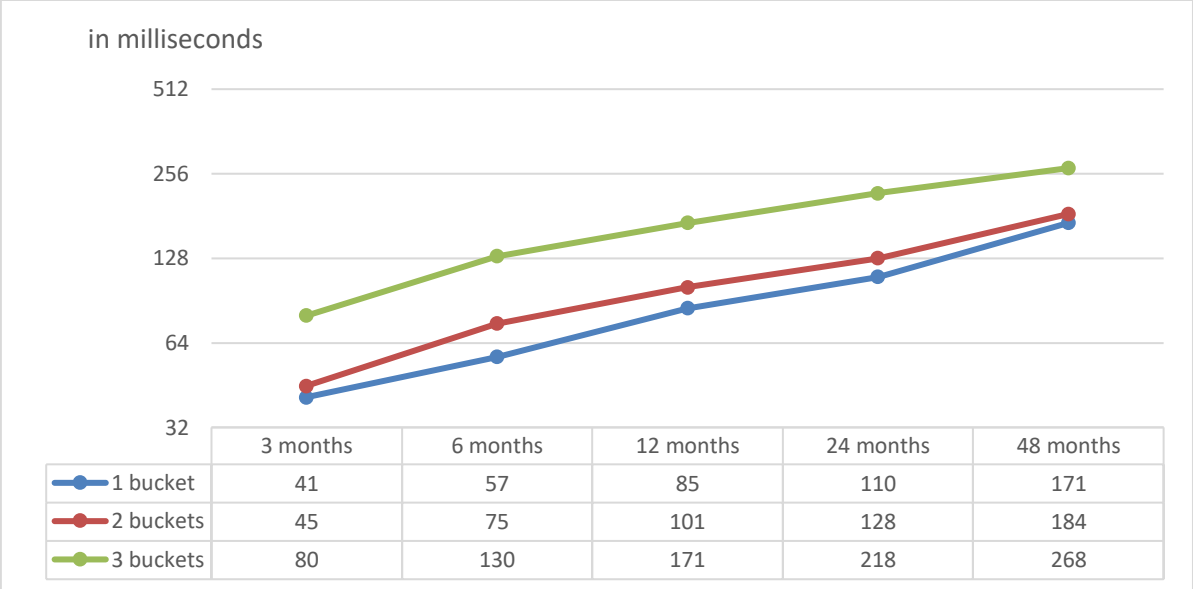
Figure 29 and Figure 30 show the variation with respect to the dataset size. The execution time axis follows the power 2 logarithmic scale. We can see for both models

that the increase of the execution time is sub-linear with respect to the dataset size. This shows that ES is not impacted by the size of the index.



	3 months	6 months	12 months	24 months	48 months
1 bucket	71	88	120	208	305
2 buckets	80	107	140	221	326
3 buckets	140	196	275	352	489

Figure 29 The execution time of Model A by varying the dataset size.

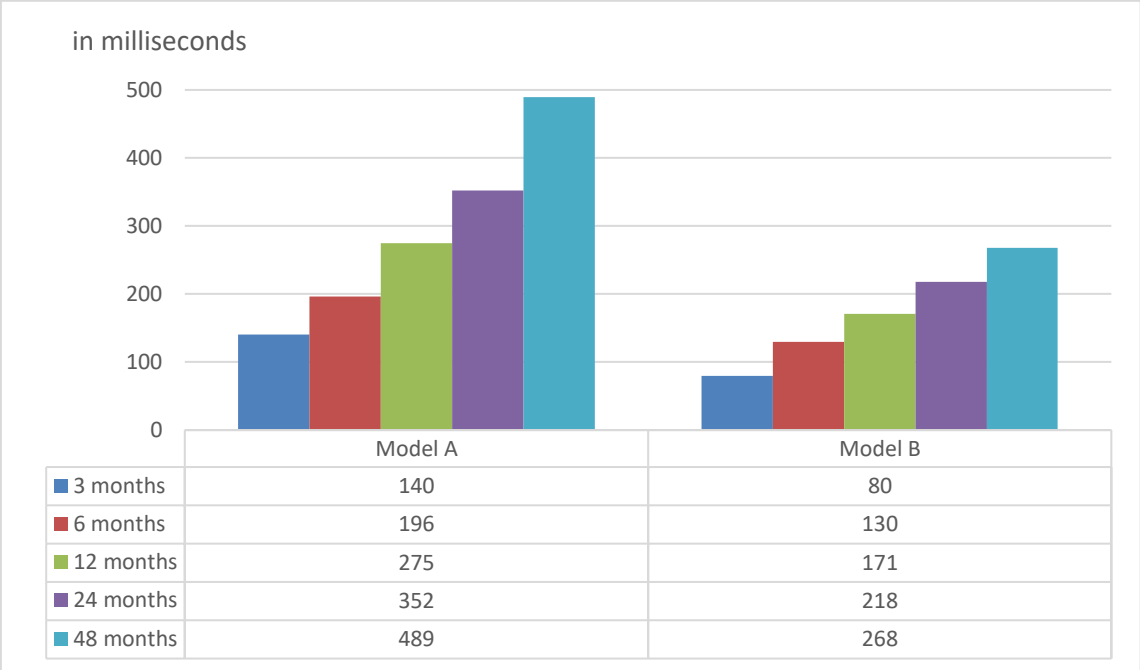


	3 months	6 months	12 months	24 months	48 months
1 bucket	41	57	85	110	171
2 buckets	45	75	101	128	184
3 buckets	80	130	171	218	268

Figure 30 The execution time of Model B by varying the dataset size.

Figure 31 shows the difference between the two models. The results include all query patterns. We can see that the queries of Model B are faster. This is mainly due to the difference in the Location dimension. For Model B, bucket queries are made on

keyword string fields which aggregate faster than the geo-point data type. The second finding is similar to those in Figure 29 and Figure 30: the increase of the execution time is sub-linear with respect to the dataset size.



	Model A	Model B
3 months	140	80
6 months	196	130
12 months	275	171
24 months	352	218
48 months	489	268

Figure 31 Comparison of execution time between two models with respect to dataset size.

Lastly, we run a comparative evaluation with MongoDB. We only consider MongoDB because (i) it is the nearest database to ES in terms of features and capabilities, (ii) it satisfies the requirements of CEBA for scalability, schema constraints and high performance, and (iii) it is capable of handling geo-referenced sensor data warehousing. Figure 32 shows the results. We run the query patterns presented before with our benchmark dataset. The results show that ES outperforms MongoDB for all queries. Note that we added indices in MongoDB for fields concerned by these queries. We observed that MongoDB does not use the indices optimally during aggregation queries, while ES leverages the indices for these benchmark queries.

	Q({"T"})	Q({"L"})	Q({"I"})	Q({"T","L"})	Q({"T","I"})	Q({"L","I"})	Q({"T","L","I"})
ES	233	355	327	340	279	359	489
MongoDB	2580	3267	2637	3573	2931	3559	3002

Figure 32 Comparative evaluation of Elasticsearch and MongoDB with 4 years dataset.

2.4 Discussion and Summary

In this chapter, we presented our proposal of environmental sensor data warehousing with ELK stack. We presented our use case which involves analyzing environmental sensor data, deployed in the wild. We presented the architecture of the system, composed of IAT for the ETL process, Elasticsearch for storage and analytics, and Kibana for visualization. We provided multidimensional models and presented their implementation in Elasticsearch. We also showed the query capabilities of ES and we ran a set of experiments to assess the viability of our solution.

The experiments showed that Elasticsearch is able to efficiently handle growing numbers of dimensions. We carried out a set of experiments to assess the value of our solution based on ES for geo-referenced data analysis. The result showed that IAT performs the ETL process efficiently. The time needed for the processing of the sensor records and their loading into ES is linear to the size of the dataset. The reported time for one record was around 2*ms*, which proves that IAT can process a data stream efficiently with high throughput. Moreover, we showed that the evolution of the query execution time is sublinear with respect to the dataset size. Also, we showed that increasing the number of bucket queries in the aggregation query, i.e., including more

dimensions, does not significantly impact the query performance. The reported disk and memory usage show that ES compresses the stored data without impacting the query performance. These results show that our solution should manage use cases with bigger datasets. Therefore, given Elasticsearch's efficient query performance and resource usage, in addition to its query capabilities for geo-referenced data analysis, it is a viable tool for spatial data warehouse solutions.

For future work, we aim to provide Elasticsearch with additional spatial query capabilities such as convex hull. We also want to investigate analyzing our use case data with big spatial data systems such as Apache Spark. Moreover, we aim to add support of OGC SensorThings API (Liang, 2016) to IAT in order to build sensor data warehouses from sources in OGC Sensor Web.

To conclude, this chapter presents a promising approach to spatial data analysis using data warehousing techniques with the ELK stack. Although this approach requires a step of loading data into a centralized repository beforehand, the experimental results demonstrate the efficiency and scalability of this proposal. Furthermore, the proposed future work provides an interesting direction for further research in this field. In the next chapter, we will explore real-time analysis of sensor data through mediation technique.

CHAPTER 3 A MEDIATION SYSTEM FOR CONTINUOUS SPATIAL QUERIES ON A UNIFIED SCHEMA USING APACHE SPARK

The results obtained in this chapter have been submitted to the Big Earth Data journal (Taylor & Francis), currently in the second round of the revision process.

3.1 Introduction

Real-time decision-making necessitates the prompt processing of data as soon as it is generated, to make timely and informed decisions. Traditional data warehousing, as discussed in the previous chapter, involves a fixed multidimensional model and a prior step of Extract, Transform, and Load (ETL). The technique of mediation offers an alternative approach for integrating data from disparate sources without the need for ETL. Specifically, given a set of data sources and their schemas $S_1, \dots, S_n$, a global schema G and mappings between G and $S_1, \dots, S_n$, mediation consists of rewriting a query expressed on schema G into a set of queries on the local data sources. In the domain of spatial data integration with mediation technique, early work of (Boucelma 2003) introduced VirGIS, a WFS-Based spatial mediation system to integrate data from heterogeneous GIS (Geographical Information Systems). It complies with the openGIS standards and specifications such as GML (Geography Markup Language) and WFS (Web Feature Service). However, no recent work tackled the challenges of analyzing streaming data or large datasets associated with the sensor data through the mediation technique.

Big spatial data frameworks such as Apache Spark can integrate and process large datasets from different sources. However, these frameworks are hard to use when the data sources are heterogeneous and numerous. To make complex streaming and spatial data analysis accessible, we propose a novel system based on mediation technique for stream-static data integration. The system allows administrators to configure a mediated schema and mappings between it and the data sources. Users can then express queries in

the mediator SQL grammar, which the system rewrites into an Apache Spark (Zaharia 2012) application submitted to a Spark cluster.

We evaluate the proposed system with respect to different groups of queries and demonstrate its superiority in handling spatial data and continuous spatial queries. The contributions of this work are:

- A mediation system for integrating heterogeneous stream-static spatial data sources.
- A dedicated SQL grammar for the expression of continuous spatial queries.
- Implementation of a query tuner in the mediation system.
- Evaluation of the mediation system with respect to several parameters.

The chapter is organized as follows: we introduce the mediator SQL grammar. Then we describe the use case used to explain our work. We present the local schemas and integrated schemas of our use case. Then, we present the architecture and the algorithms of the mediation system. Finally, we evaluate the system with respect to several aspects and we conclude and provide perspectives for future work.

3.2 A Mediator for Continuous Spatial Queries

In this section, we describe our proposal, a mediation system for integrating multiple heterogeneous data sources. Our system utilizes Apache Sedona as a processing engine, which we described in Section 1.3.4.2 of Chapter 1. Our main contribution is a mediator that simplifies integration of both stream and static spatial data. We provide a dedicated SQL grammar for the expression of continuous spatial queries. The mediator translates user continuous spatial queries on the mediated schema into a Spark app. We apply the GAV approach for the mediation. The mediator administrator defines the mapping of entities of mediated schema to entities of local schemas.

3.2.1 Dedicated SQL Grammar

In this section, we describe the SQL syntax supported by our mediation system. The syntax is dedicated to express queries with continuous spatial semantics. Figure 33 illustrates the grammar. The system supports data retrieval statement SELECT. This statement is used to retrieve rows from relations in the mediated schema. Regarding the SELECT statement, the system supports the clauses WHERE, GROUP BY, HAVING and ORDER BY with the same semantics as in ANSI SQL. Additionally, we introduce the clause WINDOW to express continuous queries. The WINDOW clause is used to return results with respect to the sliding window. It accepts two parameters, (i) window length and (ii) sliding length. Both parameters can be expressed in either seconds, minutes, or hours. We note that this clause is not available in standard ANSI SQL. Regarding the list of spatial functions and predicates, our mediation system supports those adopted by OGC and detailed in Section 1.1.6 of Chapter 1 such as intersects, distance. The grammar is presented on a higher scale in Appendix C (Figure 59 and Figure 60).

```
<Query>    ::= SELECT   <SelectList>
                FROM     <FromList>
                WHERE    <WCondition>
                GROUP BY <GroupByExpressionList>
                WINDOW   <WindowList>
                HAVING   <HavingCondition>
                ORDER BY <OrderByList> |
                SELECT   <SelectList>
                FROM     <FromList>
                WHERE    <WCondition>
                GROUP BY <GroupByExpressionList>
                WINDOW   <WindowList>
                HAVING   <HavingCondition>   |
                SELECT   <SelectList>
                FROM     <FromList>
                WHERE    <WCondition>
                GROUP BY <GroupByExpressionList>
                WINDOW   <WindowList>  |
                SELECT   <SelectList>
                FROM     <FromList>
                WHERE    <WCondition>
                GROUP BY <GroupByExpressionList>
                WINDOW    <WindowList>
                ORDER BY <OrderByList>   |

<SelectList>   ::= <AttributeList>, <AggAttributeList> |
                   <AttributeList> |
                   <AggAttributeList>, <SelectList>

<AttributeList> ::= <Attribute> |
                    <Attribute>, <AttributeList>

<Attribute> ::= LITERAL

<AggAttributeList> ::= <AggFunction>(<MeasurementAttribute>)|
                       <AggFunction>(<MeasurementAttribute>), <AggAttributeList>

<AggFunction> ::= MIN|
                  MAX|
                  AVG|
                  MEDIAN|
                  ...

<FromList>     ::= <Relation> |
                   <Relation> , <FromList>

<MeasurementAttribute> ::= <Relation>.<Attribute> |
                           <Attribute>,  <MeasurementAttribute>

<WCondition>  ::=       <WCondition> AND <WCondition> |
                        (<GeoAttributeList> <Comparator> <GeoAttributeList>) |
                        (<GeoAttributeList> <Comparator> NUMERICAL) |
                        (<Attribute> LIKE LITERAL ) |
                        (<SpatialFunctionCall> <Comparator> NUMERICAL ) |
                        <SpatialPredicateCall>, <WCondition>


<SpatialFunctionCall> ::= <SpatialFunction>(<GeoAttributeList>,<GeoAttributeList>) |
                          <SpatialFunction>(<GeoAttributeList>,GEO_OBJECT)

<SpatialPredicateCall> ::= <SpatialPredicate>(<GeoAttributeList>,<GeoAttributeList>) |
                           <SpatialPredicate>(<GeoAttributeList>,GEO_OBJECT)

<SpatialPredicate> ::= ST_Contains  |
                       ST_Within    |
                       ST_Intersects|
                       ...

<SpatialFunction> ::= ST_Distance |
                      ...

<GeoAttributeList> ::= (<Relation>,<Attribute>) |
                       <Attribute>,  <GeoAttributeList>

<Comparator> ::= <  |
                 >  |
                 =  |
                 <= |
                 >= |

<GroupByExpressionList>  ::=              |
                             <Attribute>  |
                             <Attribute>, <GroupByExpressionList>

<WindowList>   ::= <Amount> <Unit>,<Amount> <Unit>

<Amount> ::= NUMBER
<Unit> :: minutes|
          hours

<HavingCondition> ::=  <HavingCondition> AND <HavingCondition> |
                       <HavingCondition> OR <HavingCondition> |
                       <Attribute> LIKE LITERAL  |
                       <Attribute> <Comparator> NUMERICAL

<OrderByList>  ::=   <Attribute> <keyword>

<keyword>  ::=  ASC |
                DESC
```

Figure 33 Dedicated SQL grammar.

3.2.2 Motivating Example

We present our proposal by using a motivating use case example that involves integrating four data sources. We begin by illustrating the local data sources. Then, we describe the integrated schema that corresponds to the requirement of the data analysis. Finally, we provide an example of a continuous query related to the integrated schema that can be executed within our system.

Figure 34 displays the UML class diagram for the local data sources and the integrated schema (IS).

- Data source 1 (S1) consist of two tables: (i) department and (ii) commune in France. A department is an administrative region, while a commune refers to a French town in France. The table schemas for these tables are as follows:
  - S1.department(*departmentCode*,departmentName,regionName)
  - S1.commune(*communeCode*, communeName, communeShape, regionName, regionShape, departmentCode)
- Data source 2 (S2) stores the geographical coordinates of buildings, categorized as industrial, residential, or administrative. The table schema for this source is as follows:
  - S2.building(boundaries, type)
- Data source 3 (S3) includes both static and stream data, with static information stored in the device and measure tables, while the observations table contains streaming data. The table schemas for this source are as follows:
  - S3.observation(*measureID*, measureTime, value, location, deviceID)
  - S3.device(*deviceID*, deviceName, applicationID, applicationName)
  - S3.measure(*measureID*, measureName)
- Data source 4 (S4) represents a continuous stream of data collected by active sensors, with each record containing a timestamp and location information. The table schema for this source is as follows:
  - S4.observation(measureName, measureTime, value, location, deviceID, deviceName, applicationID, applicationName).

The requirement of the integration is to be able to analyze environmental indicators around residential areas, industrial zones, etc. Hence the integrated schema has four relations, Sensor, Building, Commune, Region. The relations in the integrated schema are mapped to the relations of local schemas using the GAV mapping technique (detailed in Section 1.4.2 of Chapter 1). These mappings are listed in Table 13.

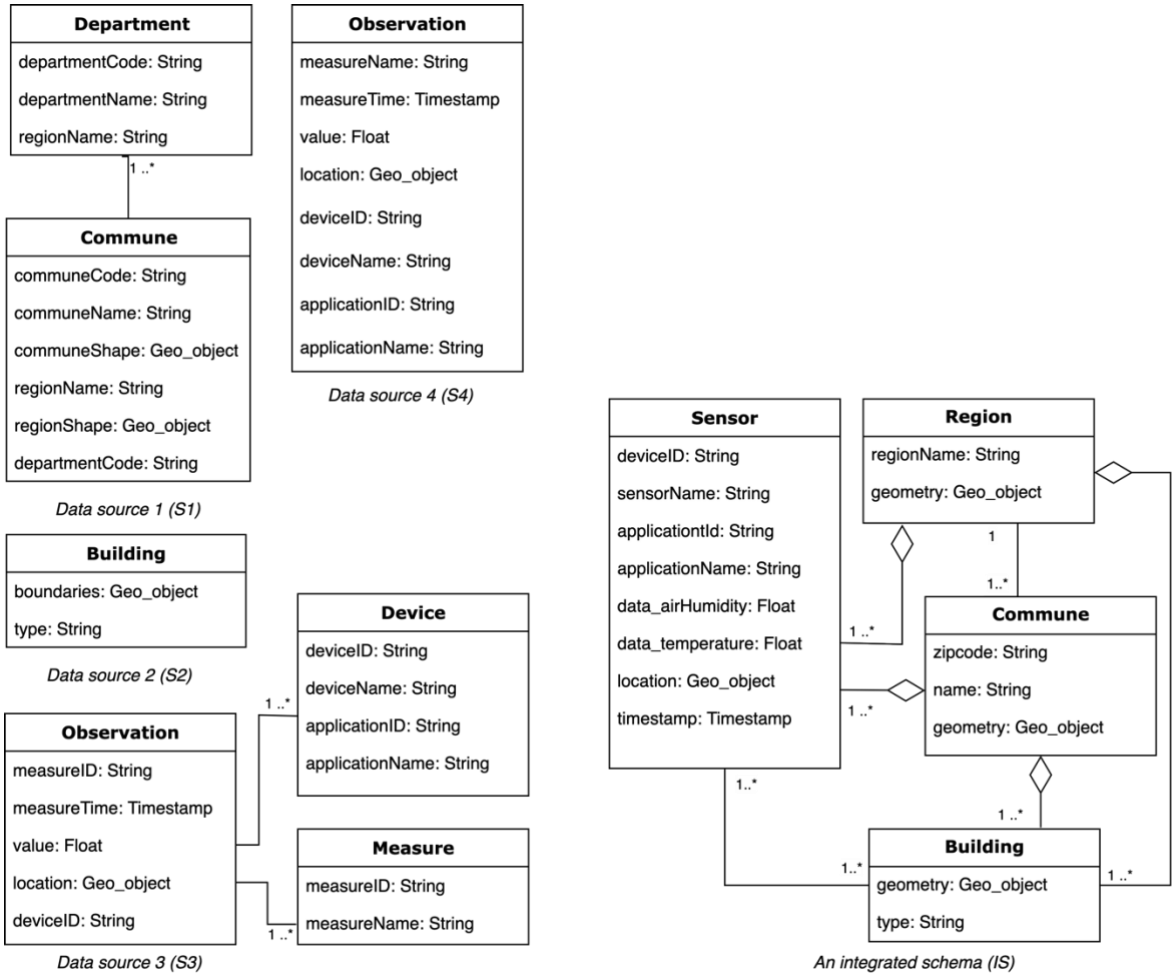


Figure 34 Local and integrated schema.

Global schema		Relations
G.region(regionName, geometry)	⊇	S1.department(departmentName, departmentCode, regionName)
G.region(regionName, geometry)	⊇	S1.commune(communeName, communeCode, communeShape, regionName, regionShape, departmentCode)
G.commune(zipcode, name, geometry)	⊇	S1.commune(communeName, communeCode, communeShape, regionName, regionShape, departmentCode)
G.building(geometry, type)	⊇	S2.building(boundaries, type)

G.sensor(deviceID, sensorName, applicationID, applicationName, data_airHumidity, data_temperature, location, timestamp)	⊇	S3.observation(measureID, measureTime, value, location, deviceID) ∧ S3.device(deviceID, deviceName, applicationID, applicationName) ∧ S3.measure(measureID, measureName), measureName=”temperature”
G.sensor(deviceID, sensorName, applicationID, applicationName, data_airHumidity, data_temperature, location, timestamp)	⊇	S3.observation(measureID, measureTime, value, location, deviceID) ∧ S3.device(deviceID, deviceName, applicationID, applicationName) ∧ S3.measure(measureID, measureName), measureName=”airHumidity”
G.sensor(deviceID, sensorName, applicationID, applicationName, data_airHumidity, data_temperature, location, timestamp)	⊇	S4.observation(measureName, measureTime, value, location, deviceID, deviceName, applicationID, applicationName) , measureName=”temperature”
G.sensor(deviceID, sensorName, applicationID, applicationName, data_airHumidity, data_temperature, location, timestamp)	⊇	S4.observation(measureName, measureTime, value, location, deviceID, deviceName, applicationID, applicationName) , measureName=”airHumidity”

Table 13 The relations in the integrated schema.

One considers the following type of continuous queries that involves the aggregation of sensor measurements within a specific geographic area over a time window: *get continuously an aggregation of sensor measurements in the last m units of time, every n units of time, within a certain geographical area.* In the context of our use case example with the data sources we have considered, we could retrieve the average air temperature and air humidity measured in the last hour within a 10 meters radius of building in Clermont-Ferrand, France, every 10 minutes. This query can be expressed with our SQL syntax within our system, as depicted in Figure 35. We note that letter “G” denotes the global schema.

```
SELECT AVG(G.sensor.data_temperature), AVG(G.sensor.data_airHumidity)
FROM G.sensor, G.commune, G.building
WHERE
Intersects(G.commune.geometry, G.building.geometry) and
Distance(G.sensor.location,G.building.geometry)<10m and
G.commune.zipcode=63000
GROUP BY WINDOW 1hour 10minutes
```

Figure 35 Running query example *Q*.

In the next sections, we will provide a more detailed explanation of the system architecture and the mediator algorithm, as well as how the mediation system validates and executes this query.

3.2.3 System Architecture

The system architecture of our mediation system is described in Figure 36, which consists of two main components: the mediator and Apache Spark. The mediator is composed of three components: query parser, query rewriter, and query tuner, all of which we designed.

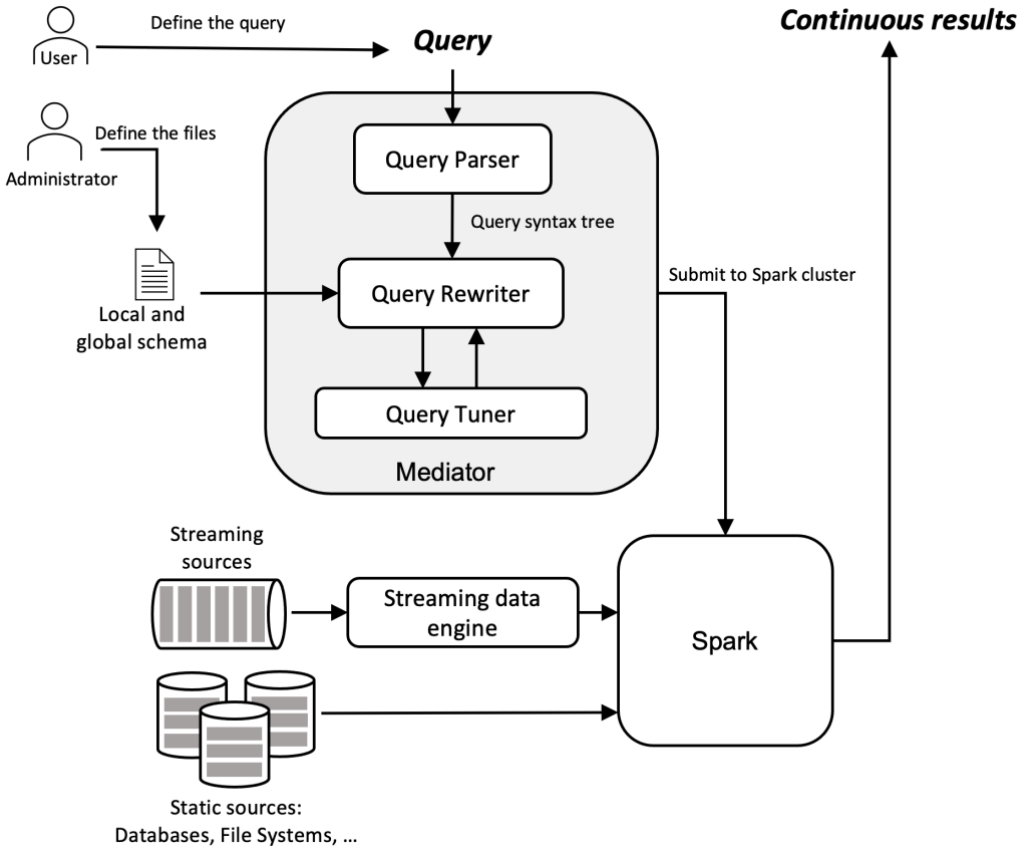


Figure 36 System architecture.

The workflow of our mediation system is as follows:

- First, the query parser uses a parser tree to analyze the input queries and match the clauses with the mediator SQL grammar (refer to Section 3.2.4.1).
- Next, the mediator rewrites the query into a Spark application according to the mappings provided by the administrator (refer to Section 3.2.4.2).

- Then the query tuner modifies the rewritten query according to a set of optimization rules to achieve higher query execution performance (refer to Section 3.2.4.3).

- Once a query is submitted to the Spark cluster, Spark workers ingest data from the data sources and the continuous result is returned to the user.

We will provide a more detailed description of each component in the following sections, explaining how they work together to achieve the desired behavior.

3.2.4 Mediator Algorithm and Components

We propose the global procedure, displayed in Figure 37, which takes a user input query and generates Spark application. The procedure requires three inputs: (i) the user query, (ii) the global schema configuration file, and (iii) the local schema configuration file. Its output is the Directed Acyclic Graph (DAG) of Spark transformations.

The global procedure of our mediator involves five main steps as follows:

i. The *query parser* parses and builds the syntax tree.
ii. For each syntax tree, the *query rewriter* constructs a Directed Acyclic Graph (DAG) transformation.
iii. The transformed DAGs are then combined into a single DAG.
iv. The tuner optimizes the DAG using methods such as push-down filter.
v. The resulting DAG is returned.

We delve into each of these steps in the subsequent subsections.

Procedure

Input: User query *Q* on global schema, Global schema config, local schema config

Output: DAG

1. // Query Parser
2. Parse and build syntax tree *ST* for *Q*
3. **For** each table T_i in query *Q* **do**
4. Parse and build syntax tree ST_i of the transformation query of the table T_i
5. // Query rewriter: For each syntax tree: map clauses to spark transformations and build a transformation DAG
6. Build transformation DAG *D* from syntax tree *ST*
7. **For each** ST_i in $ST_1 \ldots ST_n$ **do**
8. Build transformation DAG *Di* from syntax tree ST_i
9. // Assemble the different dags of transformations to make one DAG
10. Assemble $D, D_1, \ldots D_n$ into one transformation DAG D
11. // Query tuner: push down filters, improve joins, …
12. Optimize D
13. // Get and return the DAG
14. Return D.

Figure 37 Procedure of rewriting queries.

3.2.4.1 Query Parser

In the mediator, a parser tree is implemented to parse the query and match the clauses with regards to the grammar depicted in Figure 33. It also retrieves the expressions of the clauses i.e., tables, columns, and predicates.

In our generic syntax, the WINDOW clause enables time window-based aggregations and joins. The clause consists of two values: (i) window interval length and (ii) sliding length. Consider a query with expression “WINDOW 1hour 30minutes”, suppose the query processing starts at t0 = 12:00, the windows would be [12:00, 13:00], [12:30, 13:30], [13:00, 14:00], [13:30, 14:30], and so on. When a window end time is earlier than the current time, the data related to this window is discarded. The parser is implemented with two components: (i) Tokenizer and (ii) Validator. The tokenizer separates the query into a list of tokens based on a predefined vocabulary, while the validator validates whether the query respects the SQL grammar.

While validating the query by our grammar, the syntax tree is constructed with the recognized items. The syntax tree for the running query example *Q* is displayed in Figure 38.

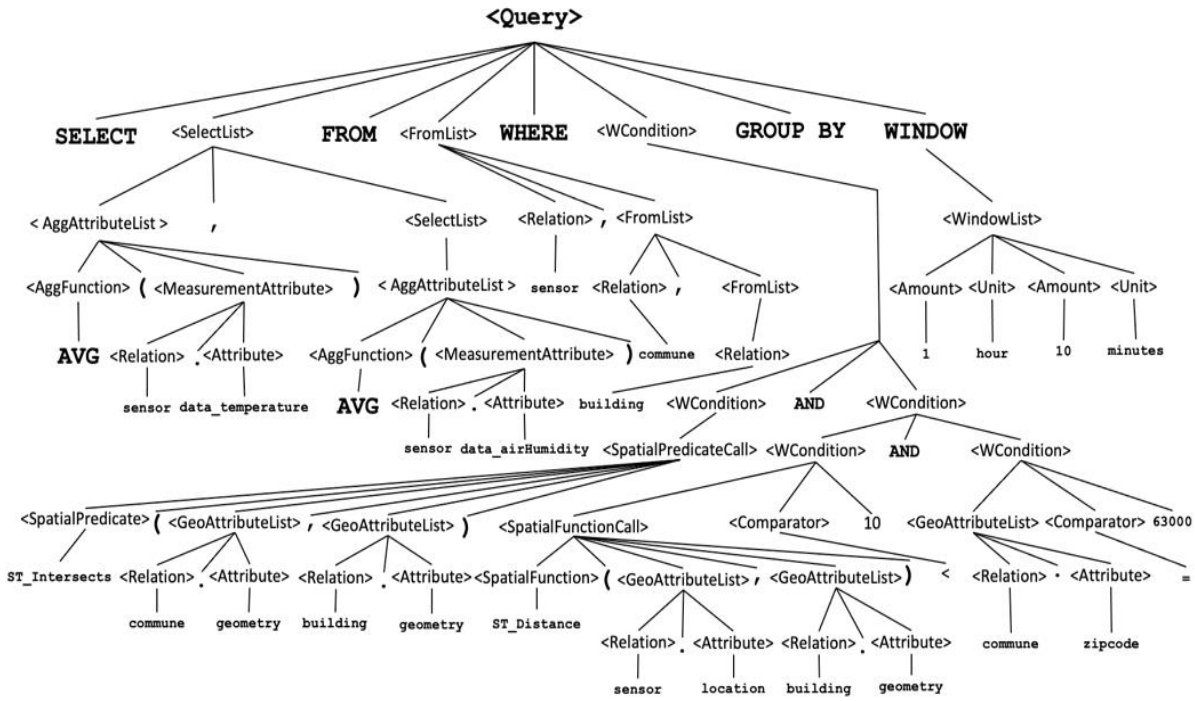


Figure 38 Running query example *Q*: Query syntax tree.

3.2.4.2 Query rewriter

When a query is submitted, the mediator generates a Structured Streaming Spark application, which can be represented as Directed Acyclic Graph (DAG) of transformations on Spark dataframes. Conceptually, Spark dataframe is equivalent to a table in relational databases. Each transformation is applied on a dataframe (also called df) and produces a new dataframe. The initial dataframes in this workflow are those that load data from the local sources specified in the query.

The transformations used in our query rewriting process are provided by the Spark programming model and are listed in Table 14. As SQL is a declarative language and the Spark programming model is functional, there is no one-to-one mapping between SQL clauses and dataframe transformations.

Dataframe Transformation	Description
select	projects a set of columns. Column names are obtained from <AttributeList> and <AggAttributeList>.
filter	filters rows by the given condition. The condition is set using the criteria defined in both <WCondition> and <HavingCondition>.
join	joins two dataframes those columns are present in a predicate or a function in <Wcondition>.
groupby	groups the dataframe using a set of columns, which are defined in <GroupbyExpressionList>. It also handles windowed grouping with elements in <WindowList>.
agg	computes aggregates for columns. The aggregation function and aggregated column are specified in <AggAttributeList>.
sort	sorts the dataframe by the column specified in OrderBy.

Table 14 Dataframe transformation description.

Therefore, query rewriter maps each subtree of the syntax tree to a spark transformation. For instance, the expression *"commune.zipcode=63000"* in the WHERE clause is mapped to the transformation *df.filter("zipcode=63000")*, where *df* refers to the input dataframe of this transformation. The inner joins that are defined implicitly in WHERE clause are as well mapped to "join" transformations. For example, the expression: *Distance("sensor.location","building.geometry")<10m*, is mapped to the transformation: *df_left.alias("a").join(df_right.alias("b"),exprs="Distance("a.location","b.geometry")<10m)"*.

Afterwards, the query rewriter assembles the transformations in the following order: "filter" -> "join" -> "select" -> "groupby" -> "agg"-> "filter"-> "sort", while respecting the order of joins that matches the logic of the query. Figure 39 part A displays the directed acyclic graph of transformations related to the running query example *Q*. The initial dataframes (at the bottom) are loaded from data sources, and the output dataframe containing continuous result of *Q* is the one that results from the "agg" transformation.

Note that even though the Spark app is implemented as a sequence of transformations, however, the processing does not necessarily occur in the same order. Indeed, Apache

Spark has a property called lazy processing where the Spark engine creates one optimized query plan for all the transformations. This technique optimizes parallel processing with minimum shuffling and temporary disk storage. This is an important advantage when working with Spark for integrating several data sources. Moreover, Spark allows for further tuning of the execution. In the next section, we explain some strategies that the mediator can integrate to achieve better performance, especially for our use cases, i.e., integrating streaming and static data.

3.2.4.3 Query Tuner

The baseline construction of a Spark application, as presented earlier, may result in low performance because it does not utilize Spark performance tuning capabilities. Although Apache Spark has an optimizer engine, called Catalyst, in its Spark SQL engine, it may not always achieve optimal performance. Hence, Spark allows developers with the ability to tune their applications. For instance, when joining two dataframes, one can choose how the two dataframes should be partitioned, and such choices can significantly impact the processing performance.

To optimize the performance of our system, we have implemented a query tuner in the query rewriter algorithm that performs two main operations:

- Push-down static-static operations.
- Broadcast join for stream-static joins.

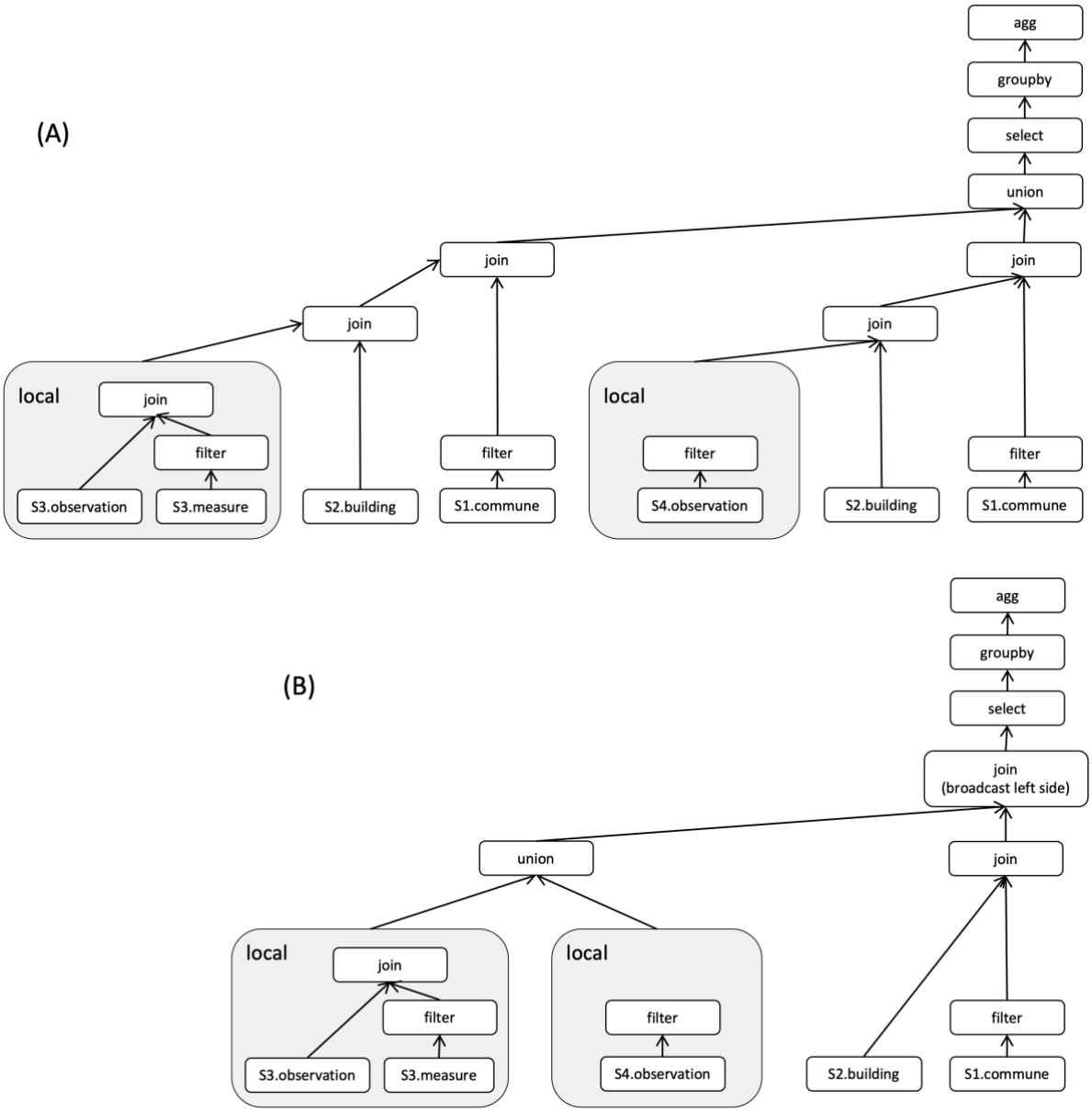


Figure 39 Running query example *Q*: Spark application DAG representation (A) and Optimizer Spark application DAG representations (B).

Figure 39 displays the logical plans of the query with and without tuning. Specifically, Figure 39 part A shows that Spark first joins the static data source *building* and a stream, then it joins the resulting data with the static data source *commune*. With this method, each incoming record is compared to both *commune* and *building* sources.

As these data sources are static, the join result remains unchanged over time. Therefore, we propose a strategy in the mediator to assemble the transformations that improve overall execution performance. We propose to push down and run first all static-static operations, i.e., which include only static data sources. Since, the results of these operations do not change over time, we do not need to compute them for new incoming stream records. Moreover, we implicitly cache the results in each worker of the cluster by specifying a broadcast join, which persists the results of static-static joins in each worker of the cluster. This enables Spark to use this cache copy for computing the join with stream data, thereby improving the join performance by avoiding data shuffling and precomputation. Figure 39 part B displays optimized DAG of transformations.

3.2.4.4 Setup Configurations

In this section, we explain the different inputs that a system administrator should prepare to integrate the system for a specific use case such as the running query example *Q*. The integration module takes two user configuration files: (i) local sources and their schemas, (ii) global schema and the mapping.

Local schemas

The user defines (i) the data store of the local source, and (ii) the columns along with their data types. The supported data stores for streaming sources such as Apache Kafka (Kreps 2011), Logstash, or static sources such as files. The supported data types include e.g., string, float, datetime, geo-object. Due to space limitation, a snippet of the local schema configuration file for the motivating example is presented in Figure 57 in the Appendix C.

Global schema and mapping

For each table, the user defines the local sources, the columns, and their data types. The schema mapping is expressed in the transformation query using SQL. Due to space limitation, a snippet of the configuration file for the motivating example is presented in Figure 58 in the Appendix C.

3.3 System Evaluation

Several works have benchmarked Apache Spark (Karimov 2018) and Geospark (Yu 2015) which recently became Apache Sedona, and showed their superiority to competitors such as (Pandey 2018), (M. M. Alam 2021). Hence, the evaluation in this section focuses on the optimization technique of the mediator described in Section 3.2.4.3 and not on comparing Apache Sedona with its competitors.

3.3.1 Dataset

For the system evaluation, we use two real static datasets and synthetic streams. The first real datasets, *building* is obtained from Open Street Map, and consists of over 423,284 polygons, each record represents a building and described by a polygon (Eldawy 2015). The dataset size is over 70MB. The second real dataset, *commune* contains over 35,228 polygons representing boundaries of communes (cities) in France (Grégoire 2015). Its size is 40MB. These details are depicted in Table 15.

For the synthetic streams, we generate records with real-time timestamps and new coordinates using a real sensor dataset. The coordinates are chosen randomly within a defined range to ensure that queries yield results. The frequency of the generation of streaming records is a variable parameter.

Dataset name	**Geometry**	**Number of geometries**	**Size**
Building	polygon	423284	74MB
Commune	polygon	35228	40MB

Table 15 Dataset of building and commune.

3.3.2 Hardware

We deployed a Spark Cluster on 9 virtual machines: one master, and 8 workers. Each machine is equipped of 2.60GHz Intel(R) Xeon(R) CPU E5-2650 v2 with 4 CPUs, and 8GB RAM, and running Linux Ubuntu. We used the distribution Apache Spark 3.2.2 with Python 3.6.

3.3.3 Metrics

(Karimov 2018) have listed metrics typically used for evaluating streaming systems, such as latency, and throughput. In terms of latency, they have considered two types: (i) event-time latency, which refers to the time interval between data generation and data ingestion, and (ii) processing time latency, which is a time interval between ingestion time and output time. For throughput, they have measured the maximum throughput and proposed sustainable throughput, which is the system's throughput without backpressure, i.e., increasing latencies.

In our experiments in the following sections, we focus mainly the processing time latency. As the query rewriting time by the mediator is negligible (few milliseconds), we do not report it. Additionally, we also compare the size of shuffled data. We recall that shuffled data is the amount of data moving between workers during data processing to reorganize Spark data partitions. The less shuffled data, the better performance.

3.3.4 Experiments

To assess the benefits of the mediator and the optimization proposed in Section 3.2.4.3, we define three different benchmark queries on our running example (refer to Section 3.2.2). These queries, intended to evaluate the system with respect to various aspects, are presented in Figure 40. A brief description of each query is provided below:

- Q1: This query involves a spatial join between the stream and static sources.
- Q2: This query is similar to Q1 but includes a windowed aggregation on the spatial join between the stream and static sources.
- Q3: This query is a windowed spatial join between the streams and static sources.

Q1
```
SELECT *
FROM sensor as s, building as b, commune as c
WHERE c.code = 63000
AND ST_INTERSECTS(b.geometry,c.geometry)
AND ST_Distance(s.geometry, b.geometry)<1
```

Q2
```
SELECT AVG(s.data_temperature)
FROM sensor as s, commune as c, building as b
WHERE c.code = 63000
AND ST_Distance(s.geometry, b.geometry) < 1
AND ST_Intersects(c.geometry, b.geometry)
GROUP BY WINDOW 2minutes 1minute
```

Q3
```
SELECT *
FROM sensor1 as s1, sensor2 as s2, commune as c, building as b
WHERE s1.applicationName = s2.applicationName
AND s1.data_node_timestampUTC >= s2.data_node_timestampUTC
AND s1.data_node_timestampUTC <= s2.data_node_timestampUTC + interval 1 hour
AND ST_Distance(s1.geometry,s2.geometry) < 1
AND c.code = 63000
AND ST_Distance(s1.geometry, b.geometry) < 1
AND ST_Intersects(c.geometry, b.geometry)
```

Figure 40 Snippet of queries.

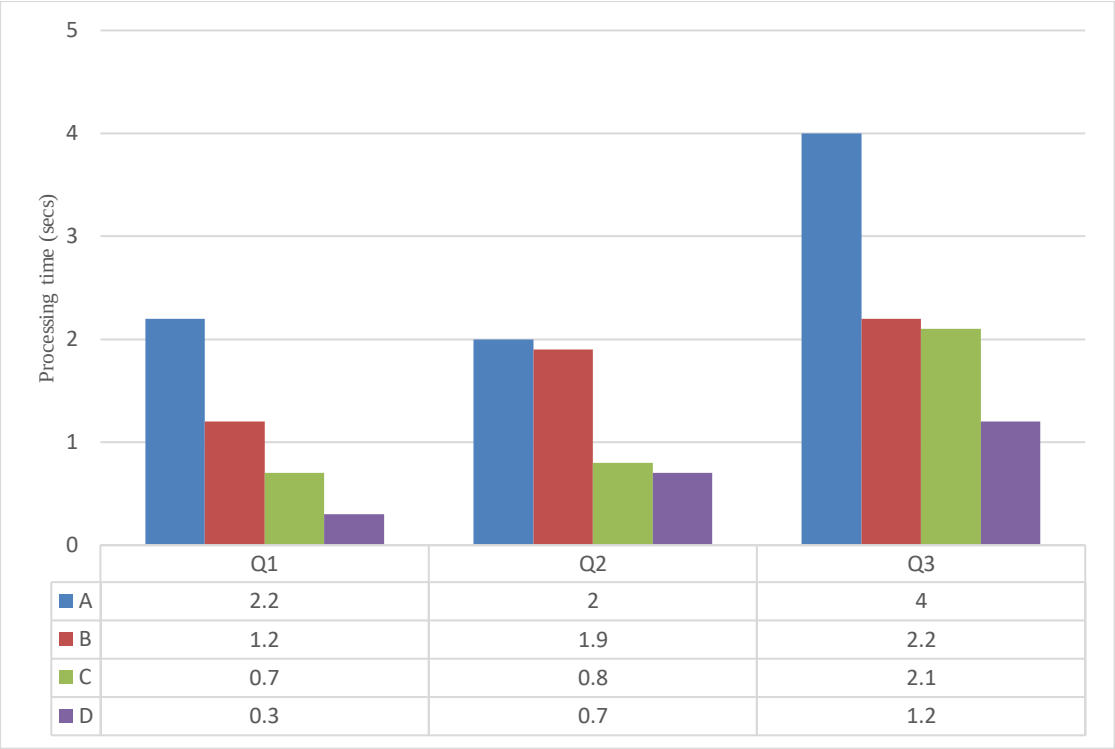
For each query, the mediator handles it with different strategies or methods. We report the average time taken to process a single batch by Spark. The code of our system is available on Github[4]. The applied methods are as follows:

- Method A: The mediator does not add any specific tuning to the query plan generated by Spark SQL optimizer.
- Method B: The mediator pushes down the computation of joins between static sources.

[4] GitHub: https://github.com/AnnaNgo13/streamgeomed

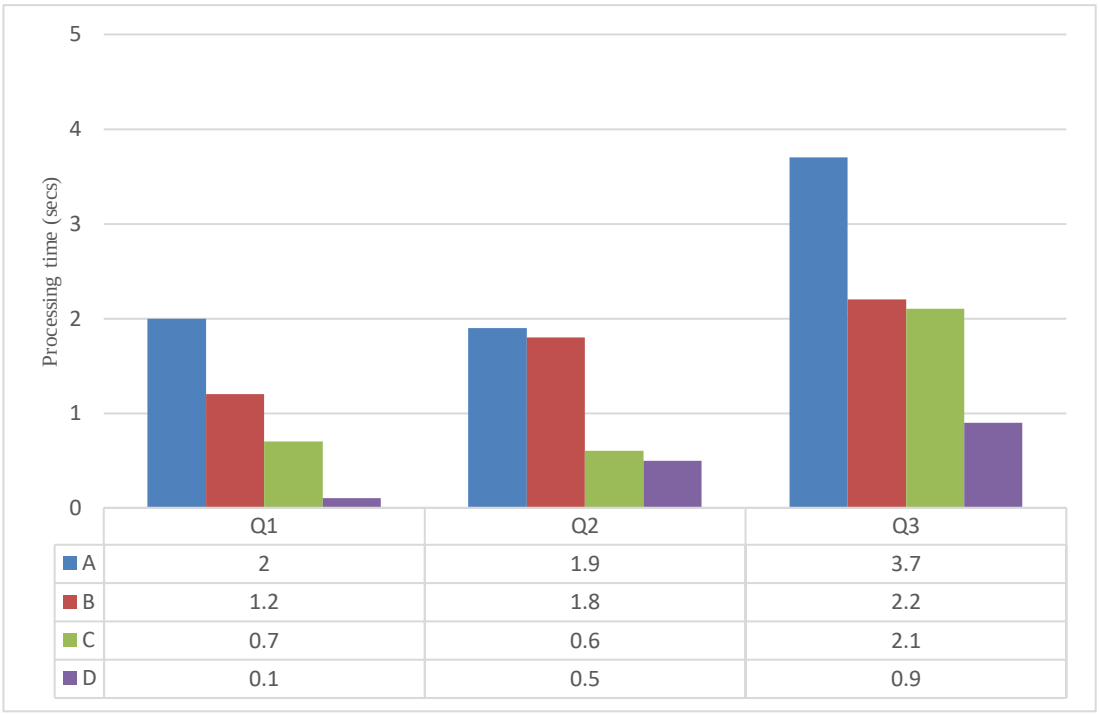
- Method C: The mediator pushes down the computation of joins between static sources and caches these results.
- Method D: This method includes the approach of method C and broadcasts the sensor streaming results to all workers.

Consider λ as the number of records generated by each source every ten seconds. Figure 41 displays the result with $\lambda = 1$ and 4 Apache Spark workers. On the other hand, Figure 42, Figure 43, and Figure 44 display the evaluation results with 8 Apache Spark workers for respectively $\lambda = 1$, $\lambda = 10$, and $\lambda = 100$. First, the results show that the technique used in the method B does not provide a substantial improvement over the baseline method. For example, in Figure 41, it is more interesting to filter the buildings with respect to the sensor records rather than joining the whole datasets *commune* and *building*. Note that Apache Sedona builds indexes for these two spatial datasets. However, for higher λ values, we can see in Figure 44, it shows that joining *building* and *commune* first is more efficient because both datasets are indexed. Joining the larger batch of streaming records with the dataset *building* has higher cost. Caching overcomes this limitation of joining static datasets first. As displayed in Figure 42, even for low λ values, method C is faster than method A for all queries. The method D, which we implemented in our system, further optimizes the queries. Caching avoids precomputation of result that do not change over time, however this result (DAG) is distributed over the cluster workers and requires data shuffling. Hence, the broadcast join method significantly reduces data shuffling, as depicted in Table 16, and therefore it reduces processing time by up to one order of magnitude. Although the broadcast join method comes with a higher memory storage cost, the storage usage during the benchmark queries evaluation was lower than 10 MB per worker machine.



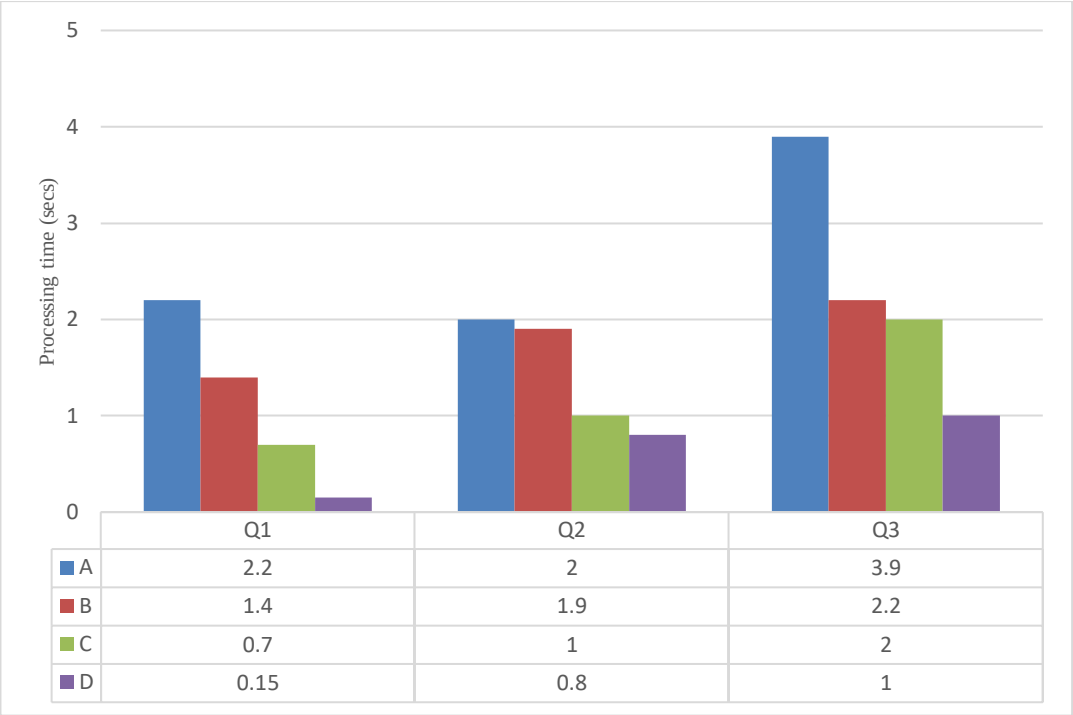
	Q1	Q2	Q3
A	2.2	2	4
B	1.2	1.9	2.2
C	0.7	0.8	2.1
D	0.3	0.7	1.2

Figure 41 $\lambda = 1$with 4 worker nodes.



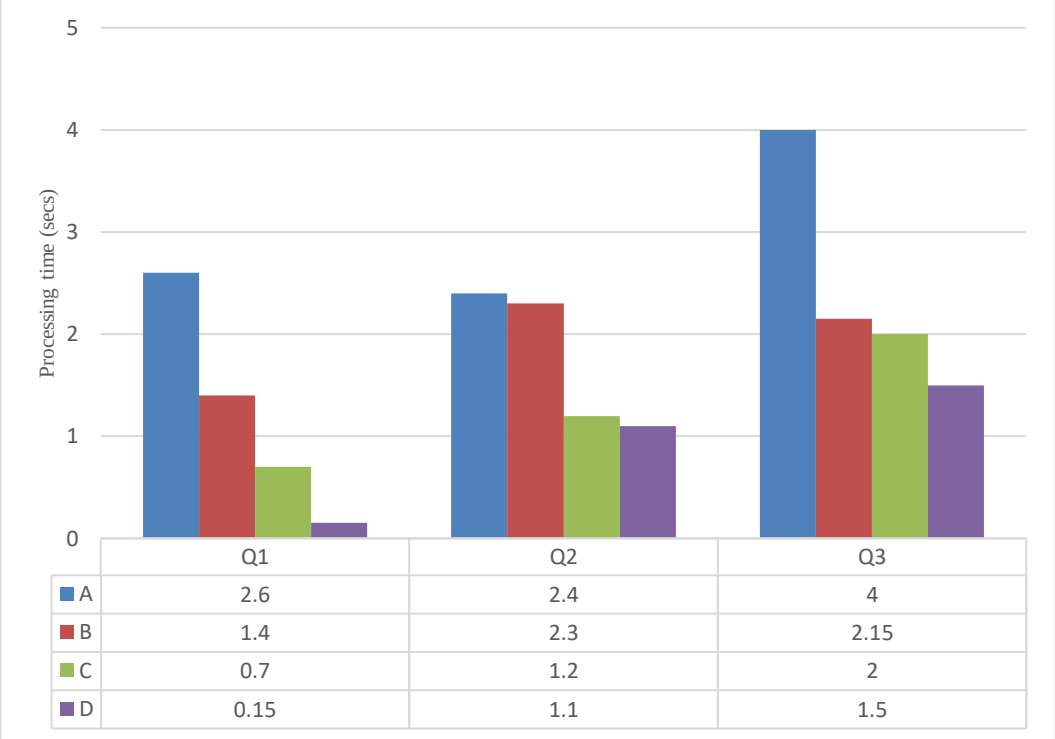
	Q1	Q2	Q3
A	2	1.9	3.7
B	1.2	1.8	2.2
C	0.7	0.6	2.1
D	0.1	0.5	0.9

Figure 42 $\lambda = 1$ with 8 worker nodes.



	Q1	Q2	Q3
A	2.2	2	3.9
B	1.4	1.9	2.2
C	0.7	1	2
D	0.15	0.8	1

Figure 43 $\lambda = 10$ with 8 worker nodes.



	Q1	Q2	Q3
A	2.6	2.4	4
B	1.4	2.3	2.15
C	0.7	1.2	2
D	0.15	1.1	1.5

Figure 44 $\lambda = 100$ with 8 worker nodes.

Method/ Query	C	D
Q1	160 MB	142 MB
Q2	164 MB	148 MB
Q3	5 MB	620 KB

Table 16 Amount of shuffled data for three queries.

3.4 Discussion and Summary

In this chapter, we proposed a mediator for integrating big and streaming data developed on top of Apache Spark, we aimed at addressing the challenges researchers face while analyzing streaming geo-referenced data without prior knowledge of big data frameworks. Our system consists of query parser, query rewriter, and query tuner. In this chapter, we discussed in detail how our mediator handles global and local schemas, and mappings in configuration files, and translates user queries submitted in SQL statements to an Apache Spark application. Furthermore, we also showed through empirical evaluation that the optimization techniques we proposed in the mediator significantly improve the execution performance of the benchmark queries as compared to the native optimizer of Apache Spark.

CHAPTER 4 MODELING DATA WAREHOUSE AND ETL FOR SENSORTHINGS API DATA

4.1 Introduction

In pursuit of interoperability among Internet of Things (IoT) systems, various organizations have developed standards to deal with spatial data such as the Open Geospatial Consortium (OGC) and the Institute of Electrical and Electronics Engineers (IEEE). As discussed in section 1.2.2 of Chapter 1, OGC initially proposed the Sensor Observation Service (SOS) as a standard for accessing sensor metadata and observations. However, it has been acknowledged that SOS is not efficient in terms of its complexity for integration with client applications. To address this limitation, OGC subsequently developed the SensorThings API (refer to section 1.2.2.4 of Chapter 1). In addition to providing a more convenient HTTP API interface, the SensorThings standard includes the messaging protocol MQTT (Cohn 2014), which allows producing events such as new *Observation*, creation of a *Sensor*, or the deletion of a *Thing*.

While the standards primarily focus on data access, they do not adequately address the needs for data analysis. This presents a significant limitation in achieving analytical capabilities on IoT data. Considering these constraints, this chapter presents an approach for the design of a conceptual model for a data warehouse dedicated to sensor data analysis. The proposed approach aims to fill the gap in the current state of the art by providing a reliable approach for sensor data analysis.

There have been numerous studies on the development of approaches for conceptual modeling for data warehouses (Jindal 2012). These models can be either ad-hoc or based on established standards such as Unified Modeling Language (UML) and Entity-Relationship (ER) models. Among these models, those based on UML have the

advantage of being easily understandable by users from diverse backgrounds as UML is widely used across different disciplines.

UML-based models for data warehouses are typically implemented as UML profiles, which propose extensions to UML using stereotypes, tagged values, and constraints. A UML stereotype is a mechanism in UML for extending the vocabulary. It allows for the definition of new model element types that are derived from existing UML element types (e.g., class, attribute) but have additional semantics or notation. This allows for the creation of domain-specific modeling languages that are tailored to the needs of specific application domains.

In this chapter, we propose a generic multidimensional model for SensorThings-compatible data sources using the UML profile proposed by (K. a. Boulil 2015). The proposal of (K. a. Boulil 2015) is currently one of the most complete conceptual models for spatial data warehouse based on UML profiles. We then illustrate it with a case study and describe the corresponding ETL process using UML as well.

4.2 Modeling Data Warehouse for SensorThings API Data

In this section, we firstly propose to redefine the UML metamodel of (K. a. Boulil 2015) for modeling and exchanging IoT sensor data sources that follows the SensorThings API (STA) data model. Then, we propose a generic hypercube model for this purpose.

4.2.1 The Data Warehouse Model Package

To design the generic UML model for STA data source, we define a UML metamodel based on the main parts of the UML metamodel proposed in (K. a. Boulil 2015) presented in Section 1.4.2.2 of Chapter 1. The SensorThings API data warehouse model (STADW) depicted in Figure 45 is composed of one or more packages

(<<Hypercube>>). Each hypercube represents a subject of analysis. A Hypercube has a fact class (<<Fact>>), and a list of dimensions (<<Dimension>>). The Fact class combines a set of measure properties (<<Measure>>). This set can be empty. Fact is linked to at least one dimension relationship (<<DimRelationship>>). Each dimension package (<<Dimension>>) has a set of hierarchies (<<Hierarchy>>) which represent level of details for analyzing facts. Finally, each (<<Hierarchy>>) includes one or multiple level class (<<Level>>). The level has a set of attribute properties (<<Attribute>>). Each hierarchy and level have a set of aggregation relationships (<<AggRelationship>>). This set can be empty.

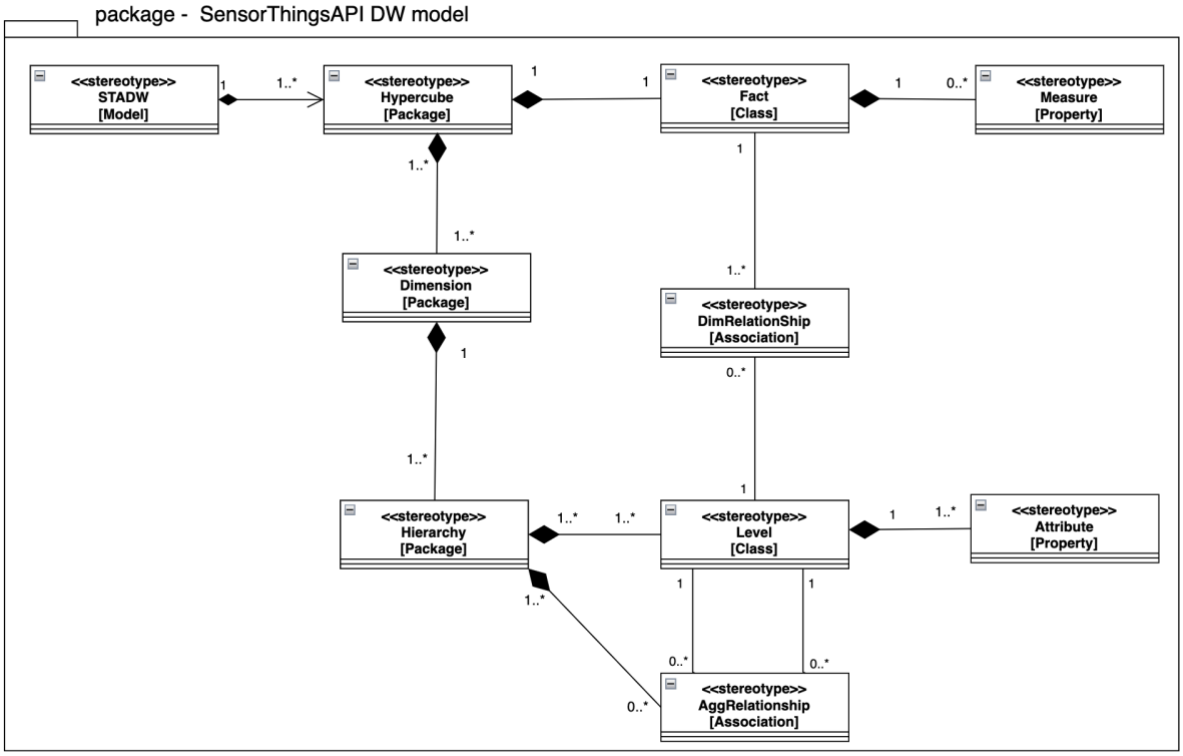


Figure 45 SensorThings API data warehouse metamodel.

4.2.2 A Generic Hypercube Model

Based on the metamodel presented above, in this section, we propose a generic hypercube of the STADW. Figure 46 displays our generic hypercube model in UML. This model comprises a fact and a set of dimensions. The Fact is represented by the

class (<<ObservedProperty>>) which the name is to be replaced by the actual observed property in the case study. This class contains *n* measure values. For dimension classes, the STA hypercube model may have up to four dimensions: Sensor, Thing, Location and Time. Sensor dimension and Thing dimension have one level of hierarchy and they represent accordingly the Sensor entity and the Thing entity of STA data model. The Location dimension have *n* levels of hierarchy, it represents geographical locations of the measurements. These locations may be inferred from the FeatureOfInterest entity or the Location entity of the STA data model. Finally, Time dimension has *n* level of hierarchies as well. The time dimension represents a time collection of measures. This information can be inferred from the Observation entity of the STA data model.

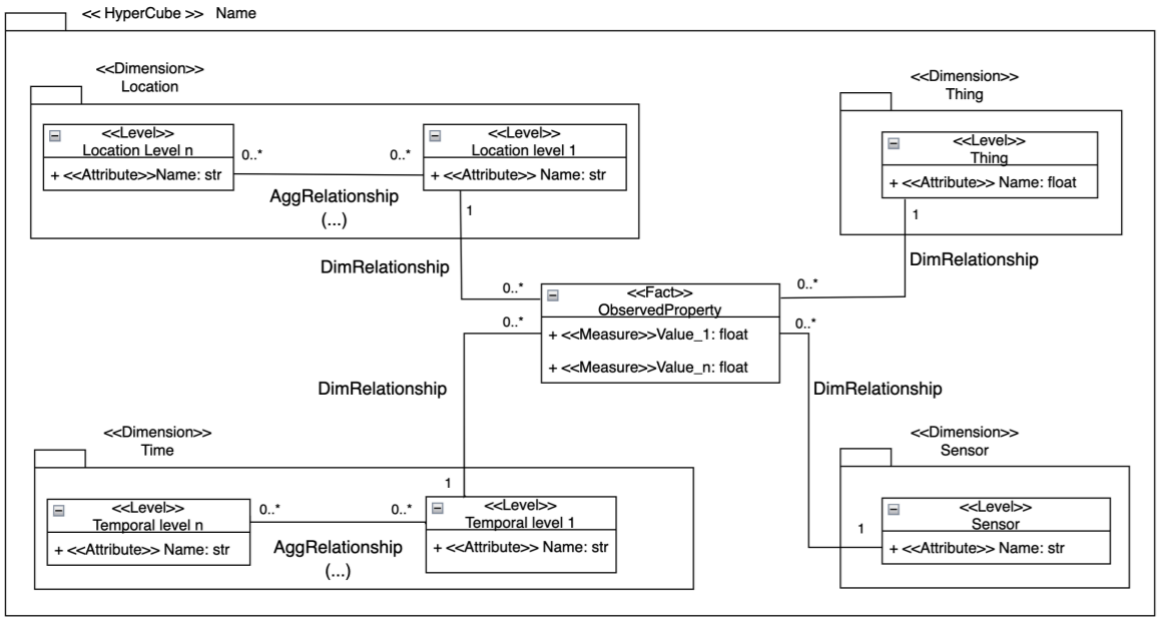


Figure 46 A generic hypercube of the STADW metamodel.

4.3 The ETL Modeling

In the context of data warehouse, the ETL process consists in extracting, transforming, and loading data to the target schema. (Trujillo 2003) proposed an approach to model ETL process in UML. Because this approach is also based on UML, it can be used in complement the modeling method presented in Section 4.2. The approach of (Trujillo

2003) consists in decomposing an ETL complex process into simple processes. Each process is described by an ETL mechanism or a UML stereotype. We aim to use their proposal to design the ETL process for our STADW model presented in the previous section. We display in Figure 47 the metamodel of the ETL process design.

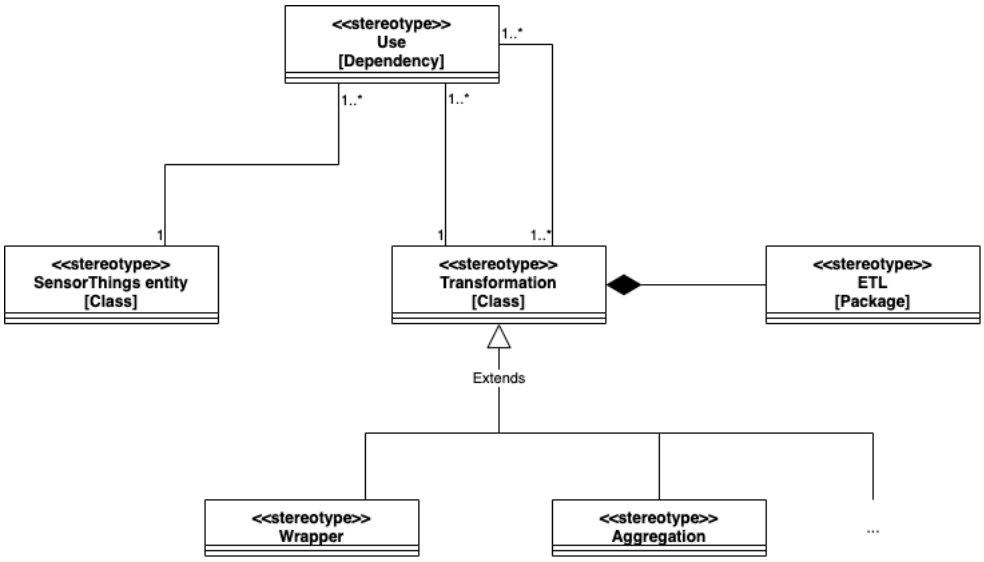


Figure 47 The metamodel of ETL process design.

We recall the main transformations proposed by (Trujillo 2003):

- Wrapper: Transforms any data source into a data structure "Table". This mechanism allows us to define the attributes to keep as well as the transformations required for each attribute.
- Aggregation: Groups the dataset with respect to a column or set of columns and applies an aggregation function.
- Conversion: Converts the data type of a column or creates a new column from a column.
- Join: Joins two datasets with respect to some columns.
- Loader: Loads the dataset into DW storage system.

We note that at the implementation level, each transformation should manipulate and transform a data structure. This data structure can be either a bounded or unbounded set of records.

4.4 Case study

In this section, our objective is to show an application of the generic schema to build a data warehouse. We consider a simple case study where one wants to study a measure *Temperature*, which is collected by several sensor devices in different areas in France. Figure 48 displays the design of the hypercube of the STADW model for analyzing the *Temperature* measure. Accordingly, Figure 49 displays the design of the ETL process.

The STADW model for *Temperature* measure is illustrated in Figure 48. We model the dimension Time with one aggregation hierarchy that has three aggregation levels, i.e., hour, day, and month. We model the dimension Location with one aggregation hierarchy that has three aggregation levels, i.e., point, department, and region. Dimensions Thing and Sensor have one aggregation level as defined in the generic model.

This multidimensional model outlined in this section necessitates the use of two distinct data sources, namely, (i) SensorThingsAPI-compatible data source for sensor data and (ii) French geographical data (Grégoire 2015). The ETL process, as illustrated in Figure 49, is comprised of a series of classes, each of which is responsible for a specific step in the process. Each class is dependent on others, which is represented through a UML association *use*. The required transformations at each step are defined in a yellow-colored UML note. The initial step is *Wrapper* which transforms each native data source into record-based data source and selects a subset of attributes. In this case study, there are two wrapper classes *ObservationWrapped* and *CityWrapped* as there are two data sources. SensorThingsApiRecord describes an aggregation mechanism where sensor record is aggregated by time window, sensor device, thing and location. The time window is set at one hour as the lowest level of hierarchy of the dimension Time is *hour*. *ConvertedRecord* describes the conversion mechanism that convert the attribute *timeWindow* into attributes *hour*, *day*, and *month* to match the Time dimension. Finally, *ExtendedRecord* implements the join mechanism which merges the two data sources with respect to geographical coordinates and then the resulting record is loaded to the DW repository.

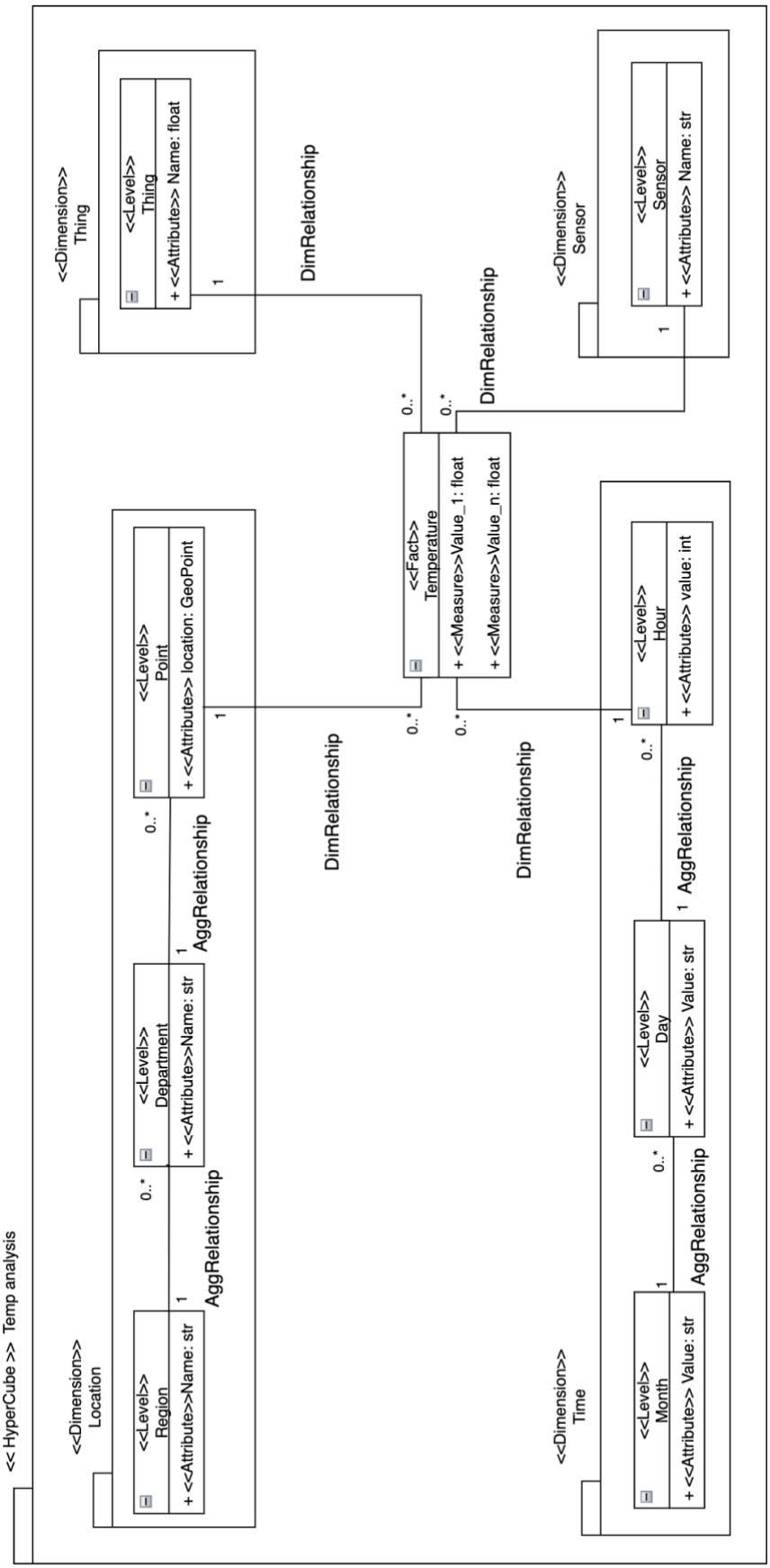


Figure 48 Model for the case study.

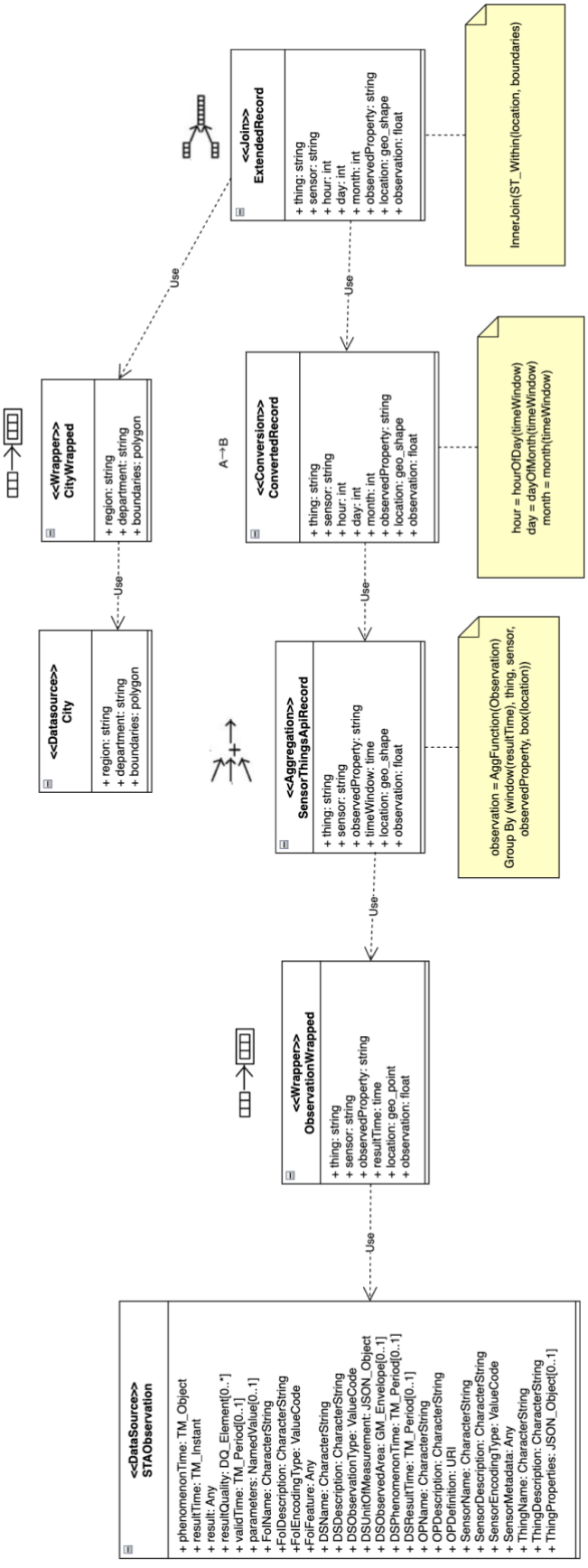


Figure 49 The design of the ETL process for this case study.

4.5 Discussion and Summary

In this chapter, we presented a preliminary study on the development of a generic spatial data warehouse model based on IoT SensorThings API data model. Specifically, we first revised the UML metamodel of (K. Boulil 2012) for spatial data warehouse, and subsequently designed a generic hypercube for the STADW metamodel, which enables allows analysis of observed properties of SensorThings API data model. However, we acknowledge that the current generic design can be further improved to enhance its analytics capabilities. To this end, future research efforts will involve evaluating the generic schema with more complex analytical requirements and data sources. We also aim to implement the case study DW and the corresponding ETL process. Additionally, we aim to investigate the implementation of automated system to develop the ETL process from the conceptual design to physical pipeline.

CONCLUSION AND PERSPECTIVES

This dissertation presents a study of data analysis for geo-referenced sensor data, focusing on several key aspects. The first contribution of this work is the extension of CEBA, a cloud environmental data lake for agriculture, with data analytics capabilities. Our approach involves the collection and analysis of geo-referenced environmental data through the ELK stack, which includes IAT for the ETL process, Elasticsearch for storage and aggregation, and Kibana for visualization. We also provided and implemented multidimensional models in Elasticsearch. Through a series of experiments, we evaluated the viability of our solution, demonstrating that Elasticsearch can handle a large number of dimensions and data, while IAT is highly efficient in processing data streams with high throughput. Our results showed that query execution time is sublinear with respect to the dataset size and increasing the number of bucket queries did not significantly impact query performance. Additionally, we found that Elasticsearch compresses stored data without affecting query performance, making it a suitable option for managing larger datasets and implementing data warehouses. Overall, our findings provide valuable insights into the design and implementation of data analysis systems for geo-referenced sensor data, and especially in environmental context.

In the second part, we present a mediator that has been specifically designed to facilitate the integration of big and streaming data. The mediator is constructed using Apache Spark as its query engine and is intended to simplify the analysis of streaming data types, such as sensor data, for researchers who lack expertise in big data frameworks. Our proposed system comprises three primary components: a query parser, a Spark application writer, and tuner. We describe the proposed SQL grammar that enables expressing continuous queries. We also describe how the mediator manages global and local schemas and mappings using configuration files, and how it transforms user queries into Apache Spark applications. Furthermore, we present empirical evidence indicating that the mediator's optimization techniques are more effective than Apache Spark's native SQL optimizer in improving query execution performance. In conclusion, this mediator provides a solution for researchers working with streaming data, who may

not possess the necessary knowledge of big data frameworks, and enhances the efficiency of data analysis.

In the third part, a preliminary investigation on the modeling of data warehouses for SensorThings data sources was presented. To facilitate the analysis of sensor data, we introduced a generic multidimensional model based on the Unified Modeling Language (UML) profile proposed by (K. Boulil 2012). Furthermore, we present a use case was presented to exemplify the practical application of the proposed approach. Overall, this chapter contributes to the development of a conceptual framework for data warehousing in the context of SensorThings data sources.

The problematic and challenges addressed in this dissertation along with our proposals provide some orientations for future work:

Regarding our proposal in Chapter 2, we have highlighted the robust analytical query capabilities of Elasticsearch concerning spatial data. In an effort to enhance its potential, we aim to further investigate the integration of supplementary spatial query functionalities, such as convex hull. In the field of multidimensional modeling, our objective is to provide generic models that facilitate the representation of various use cases related to environmental monitoring and sensor data analysis. Our preliminary efforts in this regard, as detailed in Chapter 4, entail the creation of a generic multidimensional model tailored to the OGC SensorThingsAPI data model.

In Chapter 3, we focused on the analysis of sensor data using the continuous queries for real-time analysis. As part of our future work, we intend to enhance our system's functionality to accommodate a more extensive range of data sources, both static and streaming. To improve query performance, we employ optimization techniques such as the pushdown of static-static joins and caching of partial results within the Apache Spark application plan. We plan to extend our pushdown capabilities to include local sources, capitalizing on any indexes present within the source databases, thereby minimizing the volume of data ingested by the Spark engine. Additionally, our current system requires the expertise of an integration administrator to establish mappings between local and global schemas. To simplify this process, we propose investigating

dynamic mapping in the context of streaming and spatial data to enable automatic inference of mappings during query execution, as recommended by (X. L. Dong 2009).

In Chapter 4, we introduced a generic multidimensional model for the analysis of SensorThings data. A practical example was provided to illustrate its application. For future work, we aim to evaluate the effectiveness of the proposed schema with more complex analytical applications beyond the environmental domain of this dissertation. We aim to apply the proposed design to various use cases to assess their efficacy for end-users. Furthermore, we intend to investigate the development of an automated system for the ETL process, from conceptual design to physical pipeline, dedicated to multidimensional models based on the proposed generic schema.

The cloud solution CEBA is involved in Project PIA3 TERRA FORMA (2022-2029) with a budget of 9.7 million euros. TERRA FORMA (TerraForma 2022) aims to design and test on site observatories across multiple sites, providing a new multi-messenger vision by integrating sensor data on human, biotic, and abiotic dynamics through five work packages. In the work package 3 (WP3), an innovative and scalable communication infrastructure with computing power will be developed to process data generated by diverse sensors in real-time and feed it into databases.

Chapter 2 highlights the potential use of streaming data integration tool IAT with CEBA. Leveraging the existing tool and CEBA's architecture, it can be repurposed to design the data collection and ingestion infrastructure for TERRA FORMA by deploying it on regional datacenters.

The work conducted in chapter 3 serves as a starting point, as TERRA FORMA involves high-rate data flow and multiple systems deployed across several square kilometers. This work will guide the development of CEBA's architecture to enable on-demand high-rate data processing using containerized Kafka and Spark services. A preliminary test bed, in collaboration with INRAE, is being conducted by a second-year master student to validate the deployment of these tools and queries on top of the CEBA infrastructure using INRAE's data (INRAE 2023) flows from weather stations and farming robots.

The preliminary work in chapter 4 focuses on OGC SensorThingAPI for sensing, which will be proposed to TERRA FORMA partners as a central standard. Additionally, it emphasizes the need for work on the OGC O&M (Observations and Measurements) standard to formalize the TERRA FORMA context. The work accomplished in chapter 4 will guide the development of CEBA's data management to ensure compliance with OGC standards, including the adoption of the OGC WaterML standard (OCGwaterML 2016) for data produced in the University Clermont Auvergne water research center.

REFERENCES

De Mauro, Andrea and Greco, Marco and Grimaldi, Michele. 2015. "What is big data? A consensual definition and a review of key research topics." *AIP conference proceedings.* American Institute of Physics. 97--104.

Jarke, Matthias and Lenzerini, Maurizio and Vassiliou, Yannis and Vassiliadis, Panos. 2002. "Fundamentals of data warehouses." *Springer Science & Business Media.*

Inmon, William H. 2005. "Building the data warehouse." *John wiley & sons.*

Bicevska, Zane and Oditis, Ivo. 2017. "Towards NoSQL-based data warehouse solutions." *Procedia Computer Science.* Elsevier. 104--111.

Sabtu, Adilah and Azmi, Nurulhuda Firdaus Mohd and Sjarif, Nilam Nur Amir and Ismail, Saiful Adli and Yusop, Othman Mohd and Sarkan, Haslina and Chuprat, Suriayati. 2017. "The challenges of extract, transform and loading (etl) system implementation for near real-time environment." *2017 International Conference on Research and Innovation in Information Systems (ICRIIS).* IEEE.

Bajer, Marcin. 2017. "Building an IoT data hub with Elasticsearch, Logstash and Kibana." *5th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW).* IEEE. 63--68.

Alam, Md Mahbub and Torgo, Luis and Bifet, Albert. 2021. "A Survey on Spatio-temporal Data Analytics Systems." *arXiv preprint arXiv:2103.09883.*

Guo, Dongming and Onstein, Erling. 2020. "State-of-the-Art Geospatial Information Processing in NoSQL Databases." *ISPRS International Journal of Geo-Information.* Multidisciplinary Digital Publishing Institute. 331.

Terray, Luca and Royer, Laurent and Sarramia, David and Achard, Cyrille and Bourdeau, Etienne and Chardon, Patrick and Claude, Alexandre and Fuchet, J'erome and Gauthier, Pierre-Jean and Grimbichler, David and others. 2020. "From Sensor to Cloud: An IoT Network of Radon Outdoor Probes to Monitor Active Volcanoes." *Sensors.* Multidisciplinary Digital Publishing Institute. 2755.

Bedard, Yvan and Merrett, Tim and Han, Jiawei. 2001. "Fundamentals of spatial data warehousing for geographic knowledge discovery." *Geographic data mining and knowledge discovery* (Citeseer) 2: 53--73.

Wrembel, Robert. 2006. *Data Warehouses and OLAP: Concepts, Architectures and Solutions: Concepts, Architectures and Solutions.* Igi Global.

Matei, Adriana and Chao, Kuo-Ming and Godwin, Nick. 2014. "OLAP for multidimensional semantic web databases." *Enabling Real-Time Business Intelligence.* Springer. 81--96.

Bimonte, Sandro and Tchounikine, Anne and Miquel, Maryvonne and Pinet, Francois. 2010. "When spatial analysis meets OLAP: multidimensional model and operators." *International Journal of Data Warehousing and Mining (IJDWM).* IGI Global. 33--60.

Nipun Garg, Surabhi Mithal. 2011. "Spatial databases Spatial Data warehouses." https://pdfs.semanticscholar.org/684a/4a2c41360e5965281ee09cabbb621f4400cb.pdf.

Boulil, Kamal and Pinet, Francois and Bimonte, Sandro and Carluer, Nadia and Lauvernet, Claire and Cheviron, Bruno and Miralles, Andre} and Chanet, Jean-Pierre. 2013. "Guaranteeing the quality of multidimensional analysis in data

warehouses of simulation results: Application to pesticide transfer data produced by the MACRO model." *Ecological informatics.* Elsevier. 41--52.

Lenzerini, Maurizio. 2002. "Data integration: A theoretical perspective." *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems* 233--246.

Albrecht, Alexander and Naumann, Felix. 2008. "Managing ETL Processes." *NTII* 8: 12--15.

Bansal, Srividya K and Kagemann, Sebastian. 2015. "Integrating big data: A semantic extract-transform-load framework." *Computer.* IEEE. 42--50.

Tewtia, Hemant Kumar and Singh, Deepti. 2021. "COVID-19 Insightful Data Visualization and Forecasting Using Elasticsearch." *Computational Intelligence Methods in COVID-19: Surveillance, Prevention, Prediction and Diagnosis.* Springer. 191--205.

Dubey, Shishir and Balaii, B and Rao, Dinesh and Rao, Deepak and others. 2018. "Data Visualization on GitHub repository parameters using Elastic search and Kibana." *2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI).* IEEE. 554--558.

Quoc, Hoan Nguyen Mau and Le Phuoc, Danh. 2015. "An elastic and scalable spatiotemporal query processing for linked sensor data." *Proceedings of the 11th International Conference on Semantic Systems.* 17--24.

Bartlett, Rebecca. 2019. "Local geographic information storing and querying using elasticsearch." *Proceedings of the 13th Workshop on Geographic Information Retrieval.* 1--4.

Kramer, Michel. 2020. "GeoRocket: A scalable and cloud-based data store for big geospatial files." *SoftwareX.* Elsevier. 100409.

Barnsteiner, Felix. 2015. *Elasticsearch as a time series data store.* 11. https://www.elastic.co/blog/elasticsearch-as-a-time-series-data-store.

Agarwal, Sarthak and Rajan, KS. 2016. "Performance analysis of MongoDB versus PostGIS/PostGreSQL databases for line intersection and point containment spatial queries." *Spatial Information Research.* Springer. 671--677.

Bartoszewski, Dominik and Piorkowski, Adam and Lupa, Michal. 2019. "The comparison of processing efficiency of spatial data for PostGIS and MongoDB databases." *International Conference: Beyond Databases, Architectures and Structures.* Springer. 291--302.

2014. *ISO 19115-1:2014.* Accessed March 2022. https://www.iso.org/standard/53798.html.

Pilato, David. 2017. *How to fetch data from multiple index using join like sql.* https://discuss.elastic.co/t/how-to-fetch-data-from-multiple-index-using-join-like-sql/106131.

Liang, Steve and Huang, Chih-Yuan and Khalafbeigi, Tania. 2016. "OGC SensorThings API Part 1: Sensing, Version 1.0." Open Geospatial Consortium.

Carbone, Paris and Katsifodimos, Asterios and Ewen, Stephan and Markl, Volker and Haridi, Seif and Tzoumas, Kostas. 2015. "Apache flink: Stream and batch processing in a single engine." *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering.* IEEE Computer Society.

Tatbul, Nesime. 2010. "Streaming data integration: Challenges and opportunities." *2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010).* IEEE. 155--158.

Kreps, Jay and Narkhede, Neha and Rao, Jun and others. 2011. "Kafka: A distributed messaging system for log processing." *Proceedings of the NetDB.* 1--7.

Shaikh, Salman Ahmed and Mariam, Komal and Kitagawa, Hiroyuki and Kim, Kyoung-Sook. 2020. "GeoFlink: A Distributed and Scalable Framework for the Real-time Processing of Spatial Streams." *Proceedings of the 29th ACM International Conference on Information & Knowledge Management.* 3149--3156.

Armbrust, Michael and Xin, Reynold S and Lian, Cheng and Huai, Yin and Liu, Davies and Bradley, Joseph K and Meng, Xiangrui and Kaftan, Tomer and Franklin, Michael J and Ghodsi, Ali and others. 2015. "Spark sql: Relational data processing in spark." *Proceedings of the 2015 ACM SIGMOD international conference on management of data.* 1383--1394.

Karimov, Jeyhun and Rabl, Tilmann and Katsifodimos, Asterios and Samarev, Roman and Heiskanen, Henri and Markl, Volker. 2018. "Benchmarking distributed stream data processing systems." *2018 IEEE 34th International Conference on Data Engineering (ICDE).* IEEE. 1507--1518.

Yu, Jia and Wu, Jinxuan and Sarwat, Mohamed. 2015. "Geospark: A cluster computing framework for processing large-scale spatial data." *Proceedings of the 23rd SIGSPATIAL international conference on advances in geographic information systems.* 1--4.

Pandey, Varun and Kipf, Andreas and Neumann, Thomas and Kemper, Alfons. 2018. "How good are modern spatial analytics systems?" *Proceedings of the VLDB Endowment.* VLDB Endowment. 1661--1673.

You, Simin and Zhang, Jianting and Gruenwald, Le. 2015. "Large-scale spatial join query processing in cloud." *2015 31st IEEE international conference on data engineering workshops.* IEEE . 34--41.

Xie, Dong and Li, Feifei and Yao, Bin and Li, Gefei and Zhou, Liang and Guo, Minyi. 2016. "Simba: Efficient in-memory spatial analytics." *Proceedings of the 2016 International Conference on Management of Data.* 1071--1085.

Baig, Furqan and Vo, Hoang and Kurc, Tahsin and Saltz, Joel and Wang, Fusheng. 2017. "Sparkgis: Resource aware efficient in-memory spatial query processing." *Proceedings of the 25th ACM SIGSPATIAL international conference on advances in geographic information systems.* 1--10.

Tang, Mingjie and Yu, Yongyang and Mahmood, Ahmed R and Malluhi, Qutaibah M and Ouzzani, Mourad and Aref, Walid G. 2020. "Locationspark: in-memory distributed spatial query processing and optimization." *Frontiers in big Data.* Frontiers. 30.

Wiederhold, Gio. 1992. "Mediators in the architecture of future information systems." *Computer.* IEEE. 38--49.

Halevy, Alon Y. 2001. "Answering queries using views: A survey." *The VLDB Journal.* Springer. 270--294.

Boucelma, Omar and Garinet, Jean-Yves and Lacroix, Zo'e. 2003. "The virGIS WFS-based spatial mediation system." *Proceedings of the twelfth international conference on Information and knowledge management.* 370--374.

Eldawy, Ahmed and Mokbel, Mohamed F. 2015. "Spatialhadoop: A mapreduce framework for spatial data." *2015 IEEE 31st international conference on Data Engineering.* IEEE. 1352--1363.

Grégoire, David. 2015. *France Geojson.* https://github.com/gregoiredavid/france-geojson.

Dong, Xin Luna and Halevy, Alon and Yu, Cong. 2009. "Data integration with uncertainty." *The VLDB Journal.* Springer. 469--500.

Dong, Xin Luna and Srivastava, Divesh. 2013. "Big data integration." *2013 IEEE 29th international conference on data engineering (ICDE).* IEEE. 1245--1248.

Patgiri, Ripon and Ahmed, Arif. 2016. "Big data: The v's of the game changer paradigm." *2016 IEEE 18th international conference on high performance computing and communications; IEEE 14th international conference on smart city; IEEE 2nd international conference on data science and systems (HPCC/SmartCity/DSS).* IEEE. 17--24.

Cox, Simon and Daisey, Paul and Lake, Ron and Portele, Clemens and Whiteside, Arliss and Open, GIS. 2003. "geography markup language (GML) implementation specification." *OpenGIS Consortium, version.*

Butler, Howard and Daly, Martin and Doyle, Allan and Gillies, Sean and Hagen, Stefan and Schaub, Tim. 2016. "The geojson format."

Lee, Jae-Gil and Kang, Minseo. 2015. "Geospatial big data: challenges and opportunities." *Big Data Research.* Elsevier. 74--81.

Bartha, Gabor and Kocsis, Sandor. 2011. "Standardization of geographic data: The european inspire directive." *European Journal of Geography* 2: 79--89.

Fritz, Steffen and Scholes, Robert J and Obersteiner, Michael and Bouma, Jetske and Reyers, Belinda. 2008. "A conceptual framework for assessing the benefits of a global earth observation system of systems." *IEEE Systems journal* (IEEE) 2: 338--348.

Na, Arthur and Priest, Mark. 2007. "Sensor observation service." *Implementation Standard OGC* 21.

Pinet, Francois and Schneider, Michel. 2010. "Precise design of environmental data warehouses." *Operational Research.* Springer. 349--369.

Guting, Ralf Hartmut. 1994. "An introduction to spatial database systems." *the VLDB Journal.* Springer. 357--399.

Simonis, Ingo and Echterhoff, Johannes. 2011. "OGC Sensor Planning Service Implementation Standard." *Open Geospatial Consortium Interface Standard: Wayland, MA, USA.*

Toshniwal, Ankit and Taneja, Siddarth and Shukla, Amit and Ramasamy, Karthik and Patel, Jignesh M and Kulkarni, Sanjeev and Jackson, Jason and Gade, Krishna and Fu, Maosong and Donham, Jake and others. 2014. "Storm@ twitter." *Proceedings of the 2014 ACM SIGMOD international conference on Management of data.* 147--156. https://storm.apache.org/.

Sedona, Apache. 2022. *Apache Sedona.* Accessed 11 2022. https://sedona.apache.org/.

Mahmood, Ahmed R and Aly, Ahmed M and Qadah, Thamir and Rezig, El Kindi and Daghistani, Anas and Madkour, Amgad and Abdelhamid, Ahmed S and Hassan, Mohamed S and Aref, Walid G and Basalamah, Saleh. 2015. "Tornado: A distributed spatio-textual stream processing system." *Proceedings of the VLDB Endowment* (VLDB Endowment) 8: 2020--2023.

Chen, Youmin and Lu, Youyou and Fang, Kedong and Wang, Qing and Shu, Jiwu. 2020. "uTree: a persistent B+-tree with low tail latency." *Proceedings of the VLDB Endowment.* VLDB Endowment. 2634--2648.

Chintapalli, Sanket and Dagit, Derek and Evans, Bobby and Farivar, Reza and Graves, Thomas and Holderbaugh, Mark and Liu, Zhuo and Nusbaum, Kyle and Patil, Kishorkumar and Peng, Boyang Jerry and others. 2016. "Benchmarking streaming computation engines: Storm, flink and spark streaming." *2016 IEEE international parallel and distributed processing symposium workshops (IPDPSW).* IEEE. 1789--1792.

Inoubli, Wissem and Aridhi, Sabeur and Mezni, Haithem and Maddouri, Mondher and Nguifo, Engelbert Mephu. 2018. "A comparative study on streaming frameworks for big data." *VLDB 2018-44th International Conference on Very Large Data Bases: Workshop LADaS-Latin American Data Science.* 1--8.

Vavilapalli, Vinod Kumar and Murthy, Arun C and Douglas, Chris and Agarwal, Sharad and Konar, Mahadev and Evans, Robert and Graves, Thomas and Lowe, Jason and Shah, Hitesh and Seth, Siddharth and others. 2013. "Apache hadoop yarn: Yet another resource negotiator." *Proceedings of the 4th annual Symposium on Cloud Computing.* 1--16.

Breton, Vincent. 2019. "CEBA, an Environmental cloud for the benefit of agriculture." *EGI Conference.* https://indico.egi.eu/event/4431/contributions/10134/.

Da Xu, Li and He, Wu and Li, Shancang. 2014. "Internet of things in industries: A survey." *IEEE Transactions on industrial informatics* (IEEE) 10 (4): 2233--2243.

Roshannejad, A Asghar and Kainz, W. 1995. "Handling relations in Spatio-temporal databases." In *ACSM-ASPRS annual conference technical papers Vol. 4. Autocarto 12. 1995.-pp. 119-126*, 119--126. ACSM.

Shekhar, Shashi. 2012. *Spatial big data challenges.* Vol. 14, in *Keynote at ARO/NSF Workshop on Big Data at Large: Applications and Algorithms, Durham, NC*.

Cox, S. 2007. "OGC Implementation Specification 07-022r1: Observations and Measurements-Part 1-Observation schema. Open Geospatial Consortium." Tech. Rep.

Cohn, Raphael J and Coppen, RJ and Banks, Andrew and Gupta, Rahul. 2014. "MQTT version 3.1."

Santhanavanich, Thunyathep and Schneider, Sven and Rodrigues, Preston and Coors, Volker. 2018. "INTEGRATION AND VISUALIZATION OF HETEROGENEOUS SENSOR DATA AND GEOSPATIAL INFORMATION." *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences* 4.

Eldawy, Ahmed and Mokbel, Mohamed F. 2015. "The era of big spatial data." *2015 31st IEEE International Conference on Data Engineering Workshops* (IEEE) 42--49.

Alam, Md Mahbub and Ray, Suprio and Bhavsar, Virendra C. 2018. "A performance study of big spatial data systems." *Proceedings of the 7th ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data* 1--9.

Clementini, Eliseo and Di Felice, Paolino. 1996. "A model for representing topological relationships between complex geometric features in spatial databases." *Information sciences* (Elsevier) 90 (1--4): 121--136.

Han, Jiawei and Kamber, Micheline and Pei, Jian. 2012. "Data warehousing and online analytical processing." *Data Mining* (Morgan Kaufmann Boston, MA, USA) 4: 125--185.

Garcia-Molina, Hector and Papakonstantinou, Yannis and Quass, Dallan and Rajaraman, Anand and Sagiv, Yehoshua and Ullman, Jeffrey and Vassalos, Vasilis and Widom, Jennifer. 1997. "The TSIMMIS approach to mediation: Data models and languages." *Journal of intelligent information systems* (Springer) 8: 117--132.

T. Kirk, A. Y. Levy, Y. Sagiv, and D. Srivastava. 1995. "The Information Manifold." *In C. Knoblock and A. Levy, editors, Information Gathering from Heterogeneous, Distributed Environments, Stanford University, Stanford, California.*

Ray, Suprio and Simion, Bogdan and Brown, Angela Demke. 2011. "Jackpine: A benchmark to evaluate spatial database performance." *2011 IEEE 27th International Conference on Data Engineering* (IEEE) 1139--1150.
Wilke, Claus O. 2019. "Fundamentals of data visualization: a primer on making informative and compelling figures." (O'Reilly Media).
Tomlinson, Roger. 1974. *Geographical Information Systems, Spatial Data Analysis and Decision Making in Government.* Doctoral thesis, University of London.
Haxhibeqiri, Jetmir and De Poorter, Eli and Moerman, Ingrid and Hoebeke, Jeroen. 2018. "A survey of LoRaWAN for IoT: From technology to application." *Sensors* (Mdpi) 18 (11): 3995.
Lott, Roger. 2015. "Geographic information-well-known text representation of coordinate reference systems." (Open Geospatial Consortium).
Augustin, Aloys and Yi, Jiazi and Clausen, Thomas and Townsley, William Mark. 2016. "A study of LoRa: Long range & low power networks for the internet of things." *Sensors* (MDPI) 16 (9): 1466.
Wang, Y-P Eric and Lin, Xingqin and Adhikary, Ansuman and Grovlen, Asbjorn and Sui, Yutao and Blankenship, Yufei and Bergman, Johan and Razaghi, Hazhir S. 2017. "A primer on 3GPP narrowband Internet of Things." *IEEE communications magazine* (IEEE) 55 (3): 117--123.
Moßgraber, Jürgen and Hilbring, Désirée and van der Schaaf, Hylke and Hertweck, Philipp and Kontopoulos, Efstratios and Mitzias, Panagiotis and Kompatsiaris, Ioannis and Vrochidis, Stefanos and Karakostas, Anastasios. 2018. "The sensor to decision chain in crisis management." *Proceedings of the 15th ISCRAM Conference.*
Isah, Haruna and Abughofa, Tariq and Mahfuz, Sazia and Ajerla, Dharmitha and Zulkernine, Farhana and Khan, Shahzad. 2019. "A survey of distributed data stream processing frameworks." *IEEE Access* (IEEE) 7: 154300--154316.
Elasticsearch. 2020. *ELK.* Accessed 6 2021. https://www.elastic.co/elastic-stack.
2022. *ELK stack.* Accessed July 2022. https://www.elastic.co/what-is/elk-stack.
Elasticsearch. 2021. *Creating a Visualization.* Accessed June 2021. https://www.elastic.co/guide/en/kibana/6.8/createvis.html.
—. 2021. *Scalability and resilience: clusters, nodes, and shards.* 4. Accessed June 2021. https://www.elastic.co/guide/en/elasticsearch/reference/current/scalability.html.
2022. *Geonetwork.* Accessed March 2022. https://geonetwork-opensource.org/.
OGC. 2022. *Open Geospatial Consortium.* 10. Accessed October 2022. https://www.ogc.org/.
Sigfox. 2023. *Sigfox—The Global Communications Service Provider for the Internet of Things (IoT).* Accessed January 2023. https://www.sigfox.com/.
Codeluppi, Gaia and Cilfone, Antonio and Davoli, Luca and Ferrari, Gianluigi. 2020. "LoRaFarM: A LoRaWAN-based smart farming modular IoT architecture." *Sensors* (MDPI) 20 (7): 2028.
Yao, Xiaochuang and Li, Guoqing. 2018. "Big spatial vector data management: a review." *Big Earth Data* (Taylor & Francis) 2 (1): 108--129.
Obe, Regina O and Hsu, Leo S. 2015. "PostGIS in Action." (Manning Publications Co.).
2023. *SQL Server.* Accessed January 2023. https://learn.microsoft.com/en-us/sql/relational-databases/spatial/spatial-data-sql-serve.
Amirian, Pouria and Basiri, Anahid and Winstanley, Adam. 2014. "Evaluation of data management systems for geospatial big data." *Computational Science and Its*

Applications--ICCSA 2014: 14th International Conference, Guimaraes, Portugal, June 30--July 3, 2014, Proceedings, Part V 14. Springer. 678--690.

Vatsavai, Ranga Raju and Ganguly, Auroop and Chandola, Varun and Stefanidis, Anthony and Klasky, Scott and Shekhar, Shashi. 2012. "Spatiotemporal data mining in the era of big spatial data: algorithms and applications." *Proceedings of the 1st ACM SIGSPATIAL international workshop on analytics for big geospatial data.* 1--10.

Schmid, Stephan and Galicz, Eszter and Reinhardt, Wolfgang. 2015. "Performance investigation of selected SQL and NoSQL databases." *Proceedings of the AGILE.* 1--5.

Baralis, Elena and Dalla Valle, Andrea and Garza, Paolo and Rossi, Claudio and Scullino, Francesco. 2017. "SQL versus NoSQL databases for geospatial applications." *2017 IEEE International Conference on Big Data (Big Data).* IEEE. 3388--3397.

Patroumpas, Kostas and Giannopoulos, Giorgos and Athanasiou, Spiros. 2014. "Towards geospatial semantic data management: strengths, weaknesses, and challenges ahead." *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems.* 301--310.

Zhen, LIU and Huadong, GUO and Changlin, WANG. 2016. *Considerations on geospatial big data.* Vol. 46, in *IOP Conference Series: Earth and Environmental Science*, 012058. IOP Publishing.

Yang, Jue and Zhang, Chengyang and Li, Xinrong and Huang, Yan and Fu, Shengli and Acevedo, Miguel F. 2010. "Integration of wireless sensor networks in environmental monitoring cyber infrastructure." *Wireless Networks* (Springer) 16: 1091--1108.

Fazio, Maria and Celesti, Antonio and Puliafito, Antonio and Villari, Massimo. 2015. "Big data storage in the cloud for smart environment monitoring." *Procedia Computer Science* (Elsevier) 52: 500--506.

Shvachko, Konstantin and Kuang, Hairong and Radia, Sanjay and Chansler, Robert. 2010. *The hadoop distributed file system.* Vol. 11, in *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, 1--10. Ieee.

Zaharia, Matei and Chowdhury, Mosharaf and Das, Tathagata and Dave, Ankur and Ma, Justin and McCauly, Murphy and Franklin, Michael J and Shenker, Scott and Stoica, Ion. 2012. "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing." *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12).* 15--28.

Ngo, Thi Thu Trang and Sarramia, David and Kang, Myoung-Ah and Pinet, Francois. 2021. "An Analytical Tool for Georeferenced Sensor Data based on ELK Stack." *7th International Conference on Geographical Information Systems Theory, Applications and Management.* SCITEPRESS-Science and Technology Publications. 82--89.

Shekhar, Shashi and Xiong, Hui. 2007. *Encyclopedia of GIS.* Springer Science & Business Media.

Boulil, Kamal. 2012. *Une approche automatisée basée sur des contraintes d'intégrité définies en UML et OCL pour la vérification de la cohérence logique dans les systemes SOLAP: Applications dans le domaine agri-environnemental.* Doctoral dissertation, Université Blaise Pascal-Clermont-Ferrand II.

OMG. 2011. "Unified Modeling Language (OMG UML)-Infrastructure(V2. 1.2)." *Available on: http://www. omg. org/spec/UML/2.4* 1 (2.4): 1.

Armbrust, Michael and Das, Tathagata and Torres, Joseph and Yavuz, Burak and Zhu, Shixiong and Xin, Reynold and Ghodsi, Ali and Stoica, Ion and Zaharia, Matei. 2018. "Structured streaming: A declarative api for real-time applications in apache spark." *Proceedings of the 2018 International Conference on Management of Data* 601--613.

Meng, Xiangrui and Bradley, Joseph and Yavuz, Burak and Sparks, Evan and Venkataraman, Shivaram and Liu, Davies and Freeman, Jeremy and Tsai, DB and Amde, Manish and Owen, Sean and others. 2016. "Mllib: Machine learning in apache spark." *The Journal of Machine Learning Research* (JMLR. org) 17 (1): 1235--1241.

Gonzalez, Joseph E and Xin, Reynold S and Dave, Ankur and Crankshaw, Daniel and Franklin, Michael J and Stoica, Ion. 2014. "Graphx: Graph processing in a distributed dataflow framework." *In 11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)* 599--613.

Storm, Apache. 2014. *ApacheStorm.* Accessed 10 2022. https://storm.apache.org/.

Guttman, Antonin. 1984. "R-trees: A dynamic index structure for spatial searching." *Proceedings of the 1984 ACM SIGMOD international conference on Management of data.* 47--57.

Finkel, Raphael A and Bentley, Jon Louis. 1974. "Quad trees a data structure for retrieval on composite keys." *Acta informatica* (Springer) 4: 1--9.

Trujillo, Juan and Lujan-Mora, Sergio. 2003. "A UML based approach for modeling ETL processes in data warehouses." In *Conceptual Modeling-ER 2003: 22nd International Conference on Conceptual Modeling, Chicago, IL, USA, October 13-16, 2003. Proceedings 22*, 307--320. Springer.

Ngo, Thi Thu Trang and Sarramia, David and Kang, Myoung-Ah and Pinet, Francois. 2023. "A New Approach Based on ELK Stack for the Analysis and Visualisation of Geo-referenced Sensor Data." *Springer Nature Computer Science* (Springer) 4 (3): 241. doi:https://doi.org/10.1007/s42979-022-01628-6.

ConnecSenS, Project. 2015-2020. http://www.lpc-clermont.in2p3.fr/spip.php?article583.

WPS, OGC. 2007. *OpenGIS® Web Processing Service Version: 1.0.0.* Open Geospatial Consortium Inc. Accessed March 2023. https://www.ogc.org/standard/wps/.

WMS, OGC. 2006. *OpenGIS® Web Map Server Implementation Specification Version: 1.3.0.* Open Geospatial Consortium Inc.

CSW, OGC. 2010. *OGC® Catalogue Services Standard 2.0 Extension Package for ebRIM Application Profile: Earth Observation Products Version: 1.0.0* . Open Geospatial Consortium Inc.

Momjian, Bruce. 2001. *PostgreSQL: introduction and concepts.* Vol. 192. Addison-Wesley New York.

ISO, 19162:2019. 2019. Accessed March 2023. https://www.iso.org/obp/ui#iso:std:iso:19162:ed-2:v1:en.

Sivasubramanian, Swaminathan. 2012. *Amazon dynamoDB: a seamlessly scalable non-relational database service.* Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data.

Naqvi, Syeda Noor Zehra and Yfantidou, Sofia and Zimanyi, Esteban. 2017. "Time series databases and influxdb." *Studienarbeit, Universite Libre de Bruxelles* 12.

openTSDB. 2010. *The scalable time series database.* Accessed March 2021. http://opentsdb.net/.

Rajaraman, V. 2016. "Big data analytics." *Resonance* (Springer) 21: 695--716.

Wood, Jordan. 2008. "Filter and Refine Strategy." In *Encyclopedia of GIS*. Springer US.

Miralles, A and Pinet, Francois and Carluer, Nadia and Vernier, Francoise and Bimonte, S and Lauvernet, Claire and Gouy, Veronique}. 2011. "EIS pesticide: an information system for data and knowledge capitalization and analysis." *Euraqua-peer scientific conference.*

Gray, Paul and Watson, Hugh J. 1998. "Decision support in the data warehouse." In *Decision support in the data warehouse*. Prentice Hall.

Kimball, Ralph. 1996. *The data warehouse toolkit: practical techniques for building dimensional data warehouses.* John Wiley & Sons, Inc.

Goasdoue, Francois and Lattes, Veronique and Rousset, Marie-Christine. 2000. "The use of CARIN language and algorithms for information integration: The Picsel system." *International Journal of Cooperative Information Systems* (World Scientific) 9 (04): 383--401.

Jindal, Rajni and Taneja, Shweta. 2012. "Comparative study of data warehouse design approaches: a survey." *International Journal of Database Management Systems* (Academy & Industry Research Collaboration Center (AIRCC)) 4 (1): 33.

Botts, Mike and Robin, Alexandre. 2007. "OpenGIS Sensor Model Language (SensorML) Implementation Specification. Version 1.0. 0." Open Geospatial Consortium.

Boulil, Kamal and Bimonte, Sandro and Pinet, Francois. 2015. "Conceptual model for spatial data cubes: A UML profile and its automatic implementation." *Computer Standards & Interfaces* (Elsevier) 38: 113--132.

Sarramia, David and Claude, Alexandre and Ogereau, Francis and Mezhoud, J'er'emy and Mailhot, Gilles. 2022. "CEBA: A data lake for data sharing and environmental monitoring." *Sensors* (MDPI) 22 (7): 2733.

LeRéseauDesZonesAteliers. 2023. *STSER France Zones Ateliers.* Accessed June 2023. https://www.za-inee.org/fr/reseau.

ERDF. 2023. Accessed June 2023. https://ec.europa.eu/regional_policy/funding/erdf_en.

TerraForma. 2022. Accessed June 2023. https://terra-forma.cnrs.fr/.

OCGwaterML. 2016. Accessed June 2023. https://www.ogc.org/standard/waterml/.

INRAE. 2023. Accessed June 2023. https://www.inrae.fr/en.

Appendix A

Figure 50 Benchmark dashboard – visualization 1.

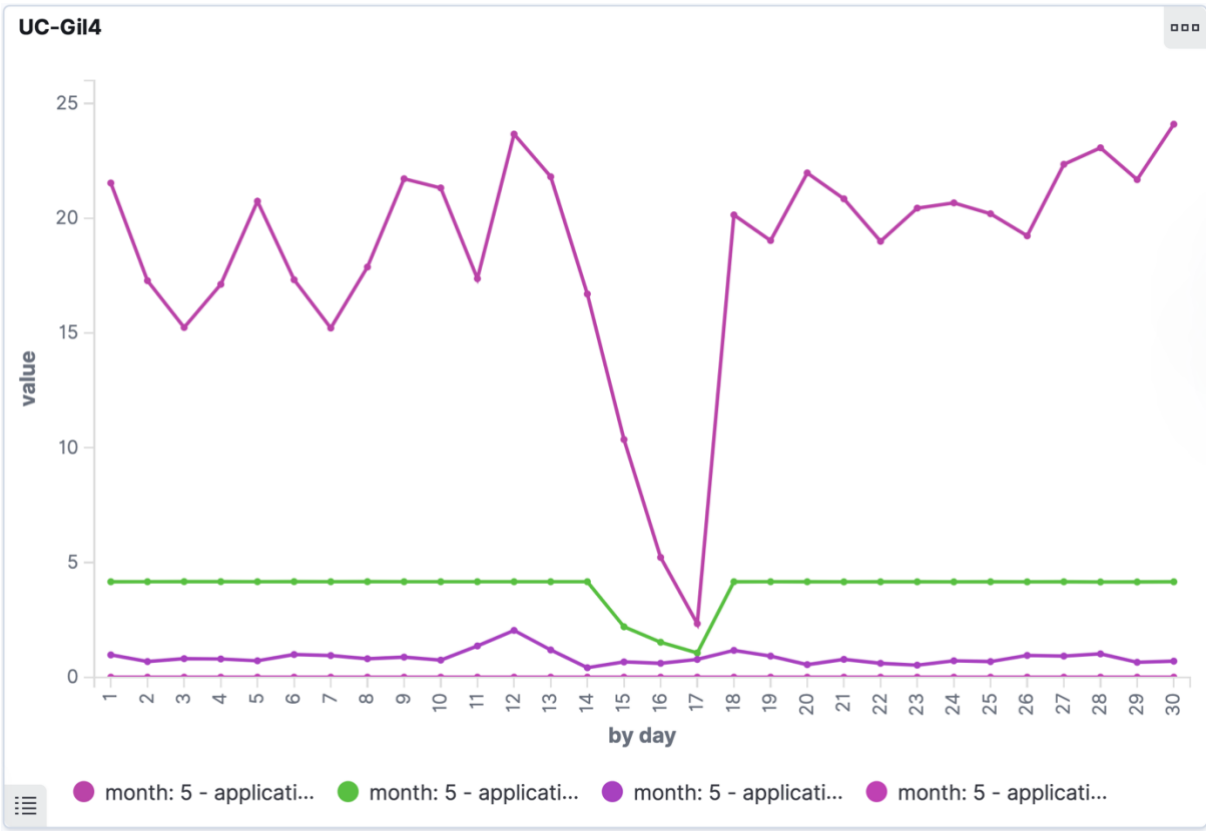


Figure 51 Benchmark dashboard – visualization 2.

Figure 52 Benchmark dashboard – visualization 3.

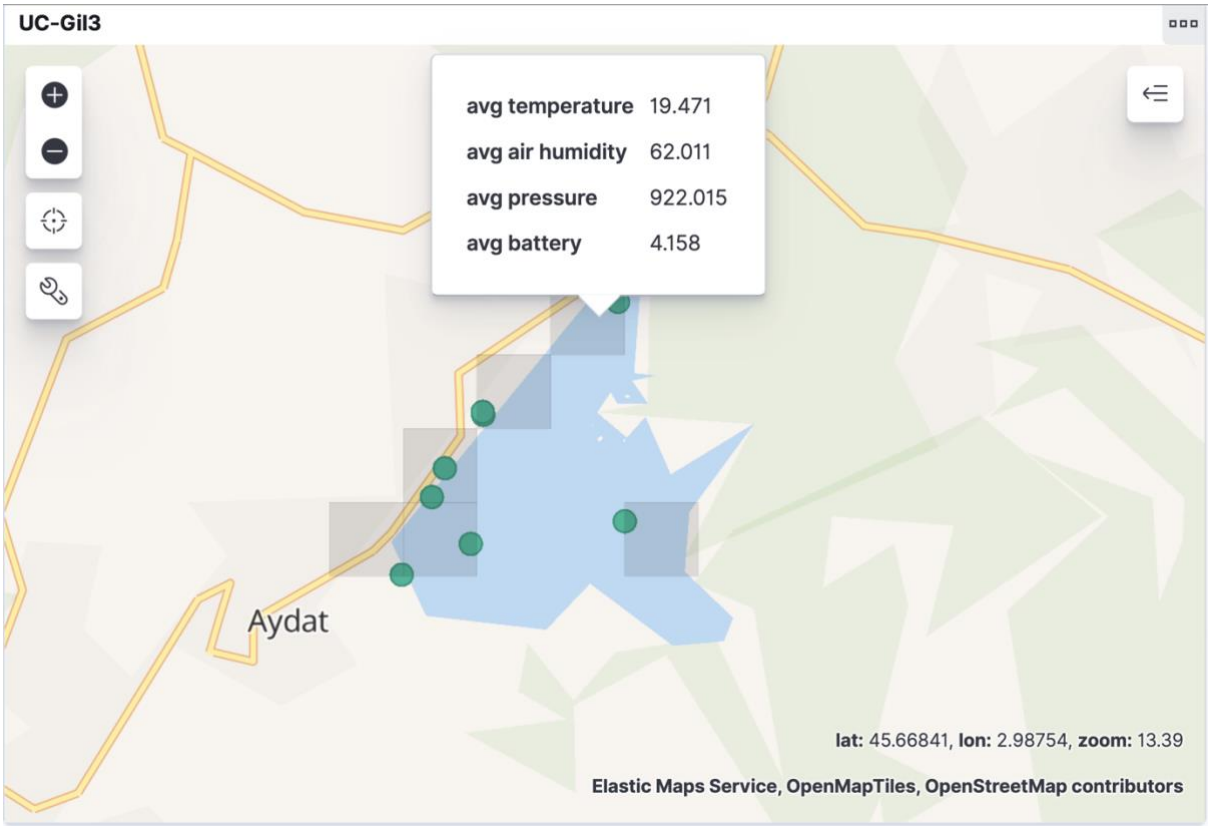


Figure 53 Benchmark dashboard – visualization 4.

APPENDIX B

We can visualise our dataset in many forms, e.g., bar charts, line graphs. In this part, we explain how to produce a visualization on Kibana (Elasticsearch, Creating a Visualization 2021).

- Navigate to the visualization page by clicking on Visualize on the left panel on Kibana home page.
- Select a visualization type, e.g., line, area, maps.
- Select the expected dataset index.

A metric and bucket aggregation query panel will be displayed by default as Figure 54.

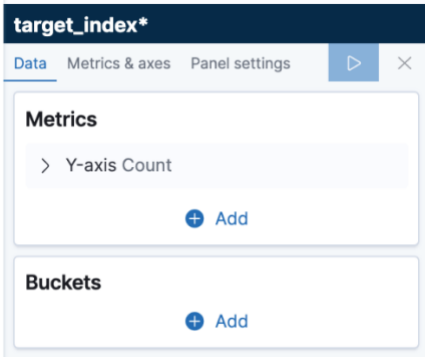


Figure 54 Kibana – Visualization controller in default mode.

In the visualization controller, the metrics in Figure 55 are represented for the visualization of Y axis and the buckets in Figure 56 are represented for the visualization of the X axis. Consider the visualization 3 (see Figure 52) as a use case example to monitor air humidity measurement by devices and hours: the Y axis shows the value of air humidity received from each sensor, the device name and hours are displayed across the X axis.

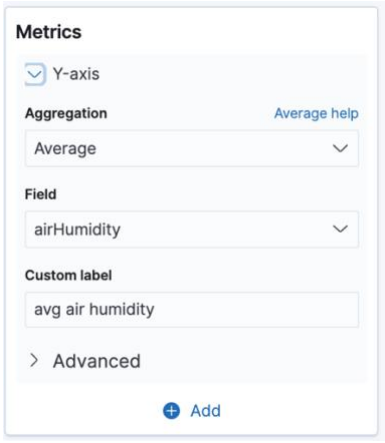


Figure 55 Kibana – Visualization metric controller.

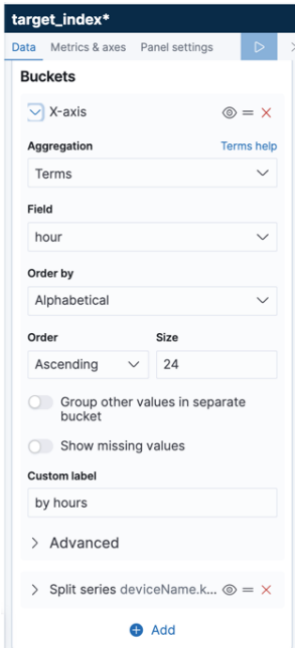


Figure 56 Kibana – Visualization bucket controller.

Appendix C

```
{
    "s1":
    {
        "id":"s1",
        "source":{
            "type":"mongo",
            "details":{
                "connection":"X.X.X.X:27017",
            }
        },
        "schema":{
            "department":
            [
                {"name":"departmentCode","type":"string"},
                {"name":"departmentName","type":"string"},
                {"name":"regionName","type":"string"}
            ],
            "commune":
            [
                {"name":"communeCode","type":"string"},
                {"name":"communeName","type":"string"},
                {"name":"communeShape","type":"geo-object"},
                {"name":"regionName","type":"string"},
                {"name":"regionShape","type":"geo-object"},
                {"name":"departementCode","type":"string"}
            ]
        }
    }
...
    "s4":
    {
        "id":"s4",
        "source":{
            "type":"kafka",
            "details":{
                "topic":"observation",
                "server":"X.X.X.X:9092"
            }
        },
...
}
```

Figure 57 Snippet of the local schema.

```
{
    "commune":
    {
        "id":"commune",
        "schema":[
            {"column":"zipcode","type":"string"},
            {"column":"name","type":"string"},
            {"column":"geometry","type":"geo-object"}
        ],
        "sources":[
            {
                "name":"s1"
                "transformation":"SELECT communeShape AS geometry,
                  communeCode AS zipcode, communeName AS name FROM commune"
            }
        ]
    },
    "sensor":
    {
        "id":"sensor",
        "schema":[
            {"column":"deviceID","type":"string"},
            {"column":"sensorName","type":"string"},
            {"column":"applicationId","type":"string"},
            {"column":"applicationName","type":"string"},
            {"column":"data_temperature","type":"float"},
            {"column":"data_airHumidity","type":"float"},
            {"column":"timestamp","type":"timestamp"},
            {"column":"location","type":"geo-object"}
        ],
        "sources":[
            {
                "name":"s4",
                "transformation":"SELECT deviceID, deviceName AS sensorName,
                applicationID, applicationName, location, measureTime AS Timestamp,
                value AS data_airHumidity WHERE measureName="air_humidity"
                FROM observation UNION SELECT deviceID, deviceName AS sensorName, applicationID,
                applicationName, location, measureTime AS Timestamp, value AS data_temperature
                WHERE measureName="temperature" FROM observation"
            },
            ...
        ],
    }
...
}
```

Figure 58 Snippet of the global schema.

```
<Query>    ::= SELECT   <SelectList>
                FROM     <FromList>
                WHERE    <WCondition>
                GROUP BY <GroupByExpressionList>
                WINDOW   <WindowList>
                HAVING   <HavingCondition>
                ORDER BY <OrderByList> |
                SELECT   <SelectList>
                FROM     <FromList>
                WHERE    <WCondition>
                GROUP BY <GroupByExpressionList>
                WINDOW   <WindowList>
                HAVING   <HavingCondition>   |
                SELECT   <SelectList>
                FROM     <FromList>
                WHERE    <WCondition>
                GROUP BY <GroupByExpressionList>
                WINDOW   <WindowList>  |
                SELECT   <SelectList>
                FROM     <FromList>
                WHERE    <WCondition>
                GROUP BY <GroupByExpressionList>
                WINDOW    <WindowList>
                ORDER BY <OrderByList>   |

<SelectList>   ::= <AttributeList>, <AggAttributeList> |
                   <AttributeList> |
                   <AggAttributeList>, <SelectList>

<AttributeList> ::= <Attribute> |
                    <Attribute>, <AttributeList>

<Attribute> ::= LITERAL

<AggAttributeList> ::= <AggFunction>(<MeasurementAttribute>)|
                       <AggFunction>(<MeasurementAttribute>), <AggAttributeList>

<AggFunction> ::= MIN|
                  MAX|
                  AVG|
                  MEDIAN|
                  ...

<FromList>     ::= <Relation> |
                   <Relation> , <FromList>

<MeasurementAttribute> ::= <Relation>.<Attribute> |
                           <Attribute>,  <MeasurementAttribute>
```

Figure 59 Dedicated SQL grammar – part 1.

```
<WCondition>  ::=        <WCondition> AND <WCondition> |
                        (<GeoAttributeList> <Comparator> <GeoAttributeList>) |
                        (<GeoAttributeList> <Comparator> NUMERICAL) |
                        (<Attribute> LIKE LITERAL ) |
                        (<SpatialFunctionCall> <Comparator> NUMERICAL ) |
                        <SpatialPredicateCall>, <WCondition>


<SpatialFunctionCall> ::= <SpatialFunction>(<GeoAttributeList>,<GeoAttributeList>) |
                          <SpatialFunction>(<GeoAttributeList>,GEO_OBJECT)

<SpatialPredicateCall> ::= <SpatialPredicate>(<GeoAttributeList>,<GeoAttributeList>) |
                           <SpatialPredicate>(<GeoAttributeList>,GEO_OBJECT)

<SpatialPredicate> ::= ST_Contains  |
                       ST_Within    |
                       ST_Intersects|
                       ...

<SpatialFunction> ::= ST_Distance |
                      ...

<GeoAttributeList> ::= (<Relation>,<Attribute>) |
                       <Attribute>,  <GeoAttributeList>

<Comparator> ::= <  |
                 >  |
                 =  |
                 <= |
                 >= |

<GroupByExpressionList>  ::=               |
                             <Attribute>  |
                             <Attribute>, <GroupByExpressionList>

<WindowList>   ::= <Amount> <Unit>,<Amount> <Unit>

<Amount> ::= NUMBER
<Unit> :: minutes|
          hours

<HavingCondition> ::=  <HavingCondition> AND <HavingCondition> |
                       <HavingCondition> OR <HavingCondition> |
                       <Attribute> LIKE LITERAL  |
                       <Attribute> <Comparator> NUMERICAL

<OrderByList>  ::=   <Attribute> <keyword>

<keyword>  ::=  ASC |
                DESC
```

Figure 60 Dedicated SQL grammar – part 2.