NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

MSc THESIS

The semantic data cube system Plato and its cache optimization

Filippos E. Yfantis

Supervisors: **Manolis Koubarakis,** Professor
Dimitris Bilidas, PhD
Georgios Stamoulis, PhD Student

ATHENS

March 2024

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Το σύστημα σημασιολογικού κύβου δεδομένων Plato με τη βελτιστοποίηση κρυφής μνήμης

Φίλιππος Ε. Υφαντής

Επιβλέποντες: **Μανόλης Κουμπαράκης,** Καθηγητής
Δημήτρης Μπηλίδας, Διδάκτωρ
Γεώργιος Σταμούλης, Υποψήφιος Διδάκτωρ

ΑΘΗΝΑ

Μάρτιος 2024

MSc THESIS

The semantic data cube system Plato and its cache optimization

Filippos E. Yfantis
S.N.: CS2190003

SUPERVISORS: **Manolis Koubarakis,** Professor
Dimitris Bilidas, PhD
Georgios Stamoulis, PhD Student

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Το σύστημα σημασιολογικού κύβου δεδομένων Plato με τη βελτιστοποίηση κρυφής μνήμης

Φίλιππος Ε. Υφαντής

Α.Μ.: CS2190003

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Μανόλης Κουμπαράκης,** Καθηγητής
Δημήτρης Μπηλίδας, Διδάκτωρ
Γεώργιος Σταμούλης, Υποψήφιος Διδάκτωρ

ABSTRACT

The DeepCube project provides us with multiple datasets, in the form of data cubes. We can access this data as a foreign table in PostgreSQL, through the development and use of a Multicorn Foreign Data Wrapper. Furthermore, we can use Ontop to query the database as a virtual RDF Graph, using SPARQL. The entire pipeline is implemented through the semantic data cube system Plato. However, the querying process lacks in efficiency and thus, we extend the Ontop plugin by implementing a cache system with a corresponding table in PostgreSQL, for faster access to the requested data. This way a certain portion of the dataset will always be materialized in the database, constantly updating based on user queries.

SUBJECT AREA: Artificial Intelligence, Big Data

KEYWORDS: RDF, OBDA, Database Management, Geospatial Data

ΠΕΡΙΛΗΨΗ

Το έργο DeepCube παρέχει πολλαπλά σύνολα δεδομένων, με τη μορφή κύβων δεδομένων. Μπορούμε να προσπελάσουμε αυτά τα δεδομένα σαν έναν εξωτερικό πίνακα (foreign table) στην PostgreSQL, μέσω της ανάπτυξης και χρήσης ενός Multicorn Foreign Data Wrapper. Επιπλέον, μπορούμε να χρησιμοποιήσουμε το Ontop για να θέσουμε ερωτήματα στη βάση δεδομένων σαν έναν εικονικό RDF γράφο, μέσω της γλώσσας SPARQL. Ολόκληρη η διαδικασία υλοποιείται μέσω του συστήματος σημασιολογικού κύβου δεδομένων Plato. Ωστόσο, η διεργασία των ερωτημάτων στερείται αποτελεσματικότητας και έτσι, επεκτείνουμε το πρόσθετο Ontop υλοποιώντας ένα σύστημα κρυφής μνήμης με αντίστοιχο πίνακα στην PostgreSQL, για ταχύτερη πρόσβαση στα ζητούμενα δεδομένα. Με αυτόν τον τρόπο ένα συγκεκριμένο τμήμα του συνόλου δεδομένων θα υπάρχει πάντα στη βάση δεδομένων και θα ενημερώνεται συνεχώς με βάση τα ερωτήματα των χρηστών.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Τεχνητή Νοημοσύνη, Μεγάλα Δεδομένα

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: RDF, OBDA, Διαχείριση Βάσης Δεδομένων, Γεωχωρικά Δεδομένα

Ευχαριστώ την οικογένειά μου για την ανιδιοτελή της αγάπη.
Είναι ο λόγος που συνεχίζω να προσπαθώ για το καλύτερο.

ACKNOWLEDGEMENTS

I would like to sincerely thank my supervisors, Prof. Manolis Koumparakis, Dr. Dimitris Bilidas and Georgios Stamoulis for their immense help and guidance throughout the planning and development of this thesis. They were always willing to assist me with any issues and obstacles I had to face, and they were always there to motivate me and make sure I stayed on track, every step along the way. Without them I would not have accomplished my end goal.

I am truly grateful to them.

CONTENTS

LIST OF FIGURES

PREFACE

Problem solving has always been one of my favourite activities. Whether that be games, puzzles, math or science it has always intrigued me to break down a problem and figure out the solution, no matter how long it would take. This was further entertained with my introduction to the world of computer science, to the point that I chose to pursue a degree in it. During my Bachelor's degree I was introduced to many of its different fields, but the ones that drew my attention the most was AI and Databases. I proceeded with my academic studies by pursuing a MSc Degree, specializing in Data, Information and Knowledge Management. I was also fortunate enough to be hired as a research assistant in Prof. Koubarakis's team, which allowed me to work in the field that interested me the most. This enabled me to develop this thesis in parallel with my work in the DeepCube project.

1. INTRODUCTION

In this thesis we will try to implement a solution to handle and optimize complex GeoSPARQL queries on large datasets. We will also try to find ways to increase the performance of those queries on given datasets and extract some statistics and metrics. All of this will be presented in the context of Plato, a semantic data cube system implementation which uses ontology-based data access technologies and, in particular, the system Ontop.

A *data cube* is a multidimensional array of values. It is a natural data structure for storing analysis-ready Earth observation (EO) data as well as multidimensional data of all kinds. As a result, a number of data cube infrastructures targeting EO data have recently been developed (e.g., the Open Data Cube infrastructure in Australia, the Euro Data Cube and Earth System Data Cube funded by ESA etc.). These data cube infrastructures offer libraries and APIs (e.g., xarray, YAXArrays) to store and query multidimensional data. However, before data cube infrastructures became a trend, there had already been lots of research and development on *array data base management systems (DBMS)* (e.g., Rasdaman [6], SciDB [22] and MonetDB SciQL [30]), which provide *declarative query languages* for modeling and accessing multidimensional data.

The concept of *semantic EO data cubes* (or *semantic data cubes* for simplicity) was first presented by Augustin et al. in [4]. The term *semantic* was used to distinguish them from regular EO data cubes that contain numbers without high-level meaning for the user (e.g., reflectance values). In semantic data cubes, these values are intricately connected to *symbolic high-level concepts*, allowing users to not only gain insights into the specified concepts but also establish associations with the original values. Beyond imparting knowledge through interpretations, a semantic data cube has the potential to streamline the integration of external knowledge (datasets) and enable the linking of such information with the original values, fostering a comprehensive combined analysis. Utilization of such systems becomes evident in numerous scenarios involving geospatial data. For instance, demographic data released by a governmental organization can be leveraged to pinpoint major cities situated within a specified distance from regions designated as pine forests.

Plato is a pioneering semantic data cube system utilizing geospatial OBDA technologies. OBDA serves as a methodology for connecting an ontology, which captures geospatial knowledge about entity classes and properties within a specific application domain, to underlying data sources. These data sources, managed by specialized systems, exist in various formats and typically reside in pre-existing repositories (e.g., a geospatial relational DBMS or a shapefile). To establish the connection, declarative mappings are employed, enabling the generation of ontology terms based on information extracted from the data sources. Rather than materializing all ontology terms, in OBDA systems users can pose queries directly on the ontology. Subsequently, a process of query transformation takes place, converting the user's query into the native language understood by the underlying data sources (e,g., GeoSPARQL). This transformed query is executed, and the results are then converted back into ontology terms for presentation to the user. This approach, often referred to as the *virtual knowledge graph* approach, offers the advantage of providing users with a familiar vocabulary to articulate queries while abstracting the complexities of the underlying data sources, including intricate schemas and storage nuances. However, the process of transforming the initial query over the ontology into a query over the underlying data sources can occasionally lead to complex and sizable queries.

Due to the *impedance mismatch* between the concepts of an ontology language (directed graphs of classes, instances, properties and values) and the concepts of data cubes (multidimensional arrays of values), our task proved to be very challenging. We show how to face this problem by implementing the semantic data cube system Plato using the well-known ontology-based data access system Ontop [29], Python xarray scripts and PostgreSQL *Foreign Data Wrappers (FDWs)*. During initialization of the Ontop engine, we provide an ontology in the OWL2 QL language and a set of mappings. The mappings define the way that ontology terms are related to the data residing in the backend. After initialization, Ontop is ready to accept GeoSPARQL queries and translate them into SQL enhanced with spatial operators. The PostGIS backend contains virtual tables based on FDWs. The data cubes are stored in ".zarr" directory format or as `.nc` (netCDF) files, and we utilize Python scripts for efficient access. This is possible through the use of the Python xarray package that allows us to work with labelled multi-dimensional arrays conveniently. To handle access to the local or remote data cubes we use Multicorn, a PostgreSQL 9.1+ extension meant to make FDW development easy though the use of Python. Finally, to reduce execution time of queries over large cubes, we implemented a caching mechanism to minimize on-the-fly transformation from GeoSPARQL to SQL, and Raptor Join to efficiently compute spatial join operations.

After a thorough overview of *Plato*, we can proceed with the evaluation of its performance, using data cubes from the use cases of European Horizon 2020 project DeepCube (`https://deepcube-h2020.eu/`). Our experiments demonstrate that the optimizations we have developed, focusing mainly on the caching one, allowing us to process complicated GeoSPARQL queries targeting GBs of data very efficiently.

2. BACKGROUND AND RELATED WORK

In this chapter we will go more in depth regarding the surrounding aspects of Plato and its optimizations. We commence by further explaining data cubes, their infrastructure and their semantic enrichment, followed by array database systems, semantic web technologies and what purpose does an OBDA implementation serve. Other approaches to building semantic data cube frameworks are also highlighted. This chapter has been written jointly with my colleague Anastasios Mantas and, as a result, it also appears verbatim in his MSc thesis [16].

2.1 Data cubes

Starting of, we will present the most well-known infrastructures used to handle and operate with data cubes. In addition, we will highlight the process of semantically enriching a data cube through the characteristics of a different fully-fledged semantic data cube system, besides Plato.

2.1.1 Infrastructures

The Earth System Data Lab (ESDL) data cube provides functionalities for building multivariate data cubes from a variety of sources. The current standard is to store large spatiotemporal datasets in NetCDF format, which is optimized for local filesystems. However, with growing size of the datasets and more demand for cloud-computing there is a growing demand for cloud-optimized storage formats for spatiotemporal datasets. One option here is the zarr format which stores data and metadata in separate objects to ensure quick access to subsets of the data in high-latency file systems or object stores. During the data conversion, the input datasets are read from different sources and ingested into a common data model so that multivariate analyses can be applied on a common spatiotemporal grid. The resulting data cube is stored in the cloud-compatible zarr format and accessible to end users through python or Julia APIs. Users can apply their own complex processing methods and scale them in distributed environments using either xarray+dask (python) or YAXArrays.jl + Distributed.jl (Julia). In addition to applying existing ESDL workflows DeepCube aimed to extend the ESDL capabilities, in particular regarding possible deep learning (DL) applications. More specifically, the aim was the implementation of improvements on efficient data shuffling, storing data in overlapping chunks and thereby facilitating moving-window analyses.

2.1.2 Semantic enrichment

In their work [24], Sudmanns et al. explore the possibilities of an extensive data cube analysis, through an infrastructure which supports users to transform geodata into information. Their system, called *Sen2Cube.at* [26], utilizes computer vision (CV) to automate semantic enrichment on a big EO data scale, while an interactive web-based graphical user interface (GUI) allows users to create, save and share queries in a knowledge base, without the need of technical expertise. More specifically, a user first defines an area and a timeframe of interest for the generic fact-base, where multiple data cubes can be accessed. Then, by combining spectral categories, continuous variables and additional geographic information, they can build a semantic model in the knowledge base (the model is translated into a query against the fact-base using an inference engine [27]). These spectral categories (e.g., types of vegetation, land-forms, water depth, etc.) are generated using the Satellite Image Automatic Mapper (*SIAM*) [5] software, which is responsible for the base-level semantic enrichment by performing categorization of optical multi-spectral EO imagery (Sentinel-2 MSI) from multiple sensors in an automated manner. This methodology is also present in a more recent publication [20] by Sudmanns et al., where the greenness of Austria is measured by combining three information layers (i.e., density, change, and lifespan of vegetation) in one semantic model. The Sen2Cube.at prototype has been successfully transferred to other regions (e.g., North-Western Syria) and datasets (e.g., Sentinel-3) as well, while having been used in a variety of applications, including agricultural monitoring, soil sealing identification and the derivation of essential climate variables [23]. As in our approach, however, open research gaps like the automated instantiation of semantic EO data cubes, the evaluation and transferability of semantic models, or the homogeneity of semantic queries across multiple such data cubes [25], are topics which require further investigation.

2.2 Array databases

In this section we briefly discuss the current state in the area of specialized data management systems for multi-dimensional arrays. Our discussion is based on a recent survey paper by Baumann et al. [7]. Such systems are relevant for the task of developing the semantic data cube technology, as they can be used as backends, where array processing can be efficiently performed. The query languages supported by these systems are of particular interest, as specific fragments of an initial GeoSPARQL query can be translated to corresponding languages. Multi-dimensional arrays (also known as raster data, gridded data or data cubes) are increasingly prominent in science and engineering domains. They are the primary means of spatio-temporal sensor, image, simulation output, or statistical data representation. However, arrays are not sufficiently supported by classic database systems, resulting in information silos and architectures that are unable to keep up with the ever-increasing performance and service quality requirements. In [7] specialized array database management systems are deemed fit to tackle these issues by providing declarative query support for flexible ad-hoc analytics on large n-dimensional arrays. This is analogous to what SQL offers on set-oriented data, XQuery on hierarchical data, as well as SPARQL and CIPHER on graph data. Array Database systems combine the advantage-proven features of a declarative query language for “shipping code to data” with techniques such as parallelization for efficient server-side evaluation. Such systems act as a means to serve massive spatio-temporal data cubes in an analysis-ready manner. Their genuine array support deems them superior to other approaches, with respect to functionality, performance, scalability and data cube standards compliance. Query functionality is independent from the data encoding, and the user can specify the format of data to be delivered. Ultimately, such approaches herald a new level of service quality for data cube services in science, engineering and beyond. Furthermore, massive parallelism and distributed processing is possible with concurrent Petascale Array Database installations. This research is unprecedented, regarding the depth at which technology and available standards are explored, while also providing a comprehensive analysis of: model, query language, architecture, practical usability and performance aspects. Regarding query languages, the two most relevant standards currently are the following:

- *ISO SQL 9075 Part 15*:
  Multi-Dimensional Arrays (MDA) provides support for querying n-D arrays and domainneutral modeling, by extending the SQL query language based on the rasdaman query model. As will be discussed later, the integration of relational and array data is of great significance.
- *OGC Web Coverage Processing Service (WCPS)*:
  This standard defines a geospatial data cube analytics language. While being fundamentally similar to SQL/MDA, two main differences can be found. First, WCPS is based on the Open Geospatial Consortium (OGC) data cube standard, which revolves around the modeling of coverage data. Therefore, it understands geospatial semantics, including spatio-temporal axes and coordinate reference systems (as well as transformations between them). Second, it comes ready for integration with XPath/XQuery (on an experimental level), as XML is the most popular metadata storage format. WPCS has also demonstrated its capabilities on a Petabyte scale [17].

Other notable array query languages proposed in the literature are mentioned below. Details for each one can be found in [18].

- *Array Query Language (AQL)*
- *RasDaMan Query Language (RasQL)*
- *Array Manipulation Language (AML)*
- *Relational Array Mapping (RAM)*
- *Array Functional Language (AFL)*
- *SciQL*

A total of 19 systems are compared in [7] (featuring various Array Databases, command line tools and libraries, together with MapReduce-based tools), four out of which (full-stack array DMBSs–Rasdaman, SciDB, SciQL and EXTASCID) are extensively benchmarked. Some relevant features are selected on a conceptual level, and a comparison is presented in tabular form on the next page. The authors hope that this constitutes a representative overview to any readers willing to immerse into the field, along with being a comprehensible guide to those searching for the best suited data cube tool for their application. On this topic, a few of the main contributions are outlined. Notably, a feature matrix (Table 2.1) is constructed, addressing both abstract concepts (e.g., query language expressiveness) and practicalities (e.g. data formats, support for ingestion tools). This facilitates future testing, as a large matrix is available for further system comparison. Array system designers get a feature list, including relevant standards, along which their own tools can be crafted. In addition, the aforementioned performance benchmark (concerning four Array DBMSs) is made public. It is more rigorous and systematic than existing array benchmarks, and its design prevents tuning a system towards the tests performed, contributing to its general value and reliability. One thing that requires some clarification is the meaning of the first three columns in Table 2.1 (Relational tables, XML stores, RDF stores). According to the authors, they denote the capabilities of each system for data/metadata integration in the corresponding data model. One open question is the competence of each system for efficiently handling and querying data in each of these models. Also, a second open question is relevant to the ability of efficiently performing combined data analysis from a single query that operates both on array and other thematic data in some of these models. Specifically, for the RDF data model, to the best of our knowledge, there is no published system or application that achieves this task. For example, as reported, rasdaman can realize handling of RDF data through the AMOS II mediator system, but to the best of our knowledge there is no developed solution that achieves this. In a similar manner, there are systems that store and query RDF datasets in PostgreSQL. For example, the Strabon [15] system developed by UoA stores data in PostGIS and performs GeoSPARQL query processing. But none of these systems supports array operations with utilization of the PostGIS Raster implementation. The same holds for Oracle GeoRaster. OPeNDAP Hyrax provides RDF descriptions of its data holdings, but lacks the ability to perform combined analysis of semantic and array data through querying. Lastly, the authors highlight the importance of the ISO SQL/MDA standard. It is based on the rasdaman query language, and it integrates multi-dimensional arrays (data) into SQL (metadata). Standalone array stores, even with query capabilities, lead to silo solutions. The integration of array handling into the frequently used metadata paradigms is imperative. If ISO SQL/MDA is standardized as a universal data cube query language, a game change is expected in terms of increased interoperability and cross-domain application manageability.

Table 2.1: Highlighted conceptual features for all the different array systems and tools

	Relational tables	XML stores	RDF stores	Query Language Expressiveness	Data Formats	ISO SQL MDA	OGC/ISO geo data cubes
rasdaman	Yes (SQL/MDA std)	Yes (WCPS std)	Yes (AMOS II)	declarative array QL	CSV, JSON, TIFF, PNG, NetCDF, etc.	Yes	Yes
SciDB	No	No	No	declarative array QL	CSV/text, binary server format	No	No
SciQL	Yes	No	Yes	declarative array QL	FITS, MSEED, VAM, TIFF	No	No
EXTASCID	Yes	No	No	No, function calls	-	No	No
PostGIS Raster	Yes (postgresql)	Yes (postgresql)	Only via postgresql	Array functions with specific microsyntax	Multiple Formats	No	No
Oracle GeoRaster	Yes	Yes	Yes	PL/SQL + object relational functions with sub-language	TIFF, GIF, BMP, PNG	No	No
Teradata Arrays	Yes	Yes	Yes	Array functions with specific microsyntax	-	No	No
OPeNDAP Hyrax	Yes	Yes	Yes	-	Import: CSV, DSP, etc.	Export: ascii,NetCDF,etc.	No
xarray	No	No	No	-	Multiple Formats	No	No
TensorFlow	No	No	No	No, python library	Import/Export: binary checkpoint files + saved model	No	No
Wendelin.core	No	No	No	No, python library	No	No	No
Google Earth Engine	No	No	No	No, function calls, python and JavaScript	GeoTIFF	No	No
OpenDataCube	No	No	No	No, client-side python calls	NetCDF	No	No
xtensor	No	No	No	No, C++ library	No	No	No
Boost::geometry	No	No	No	No, C++ library	Requires external code	No	No
Ophidia	No	No	No	No, client-side command line or python	FITS, NetCDF, JSON	No	No
TileDB	No	No	Key-Value store	No	No	No	No
SciHadoop	No	No	No	Yes, functional	NetCDF, HDF	No	No
SciSpark	No	No	No	No, transformations and actions	NetCDF, HDF, CSV	No	No

2.3 Semantic Web: Languages, Systems and Technologies for Grids and Arrays

The RDF data model [1] holds a prominent position in the context of semantic web, as a simple and easy to use data model, for creating interlinked graph databases, aiming to facilitate data and information exchange on the web. RDF, along with the corresponding query language SPARQL [2], act as official World Wide Web Consortium (W3C) recommendations for exchanging and querying information in the semantic web. Regarding the spatial dimension of RDF datasets, GeoSPARQL [13] is a proposal by the OGC that contains an ontology for modeling spatial objects, coupled with an extension of the SPARQL query language. GeoSPARQL introduces spatial data types in RDF and also both qualitative and quantitative spatial predicates and functions that can be used for querying. In the following subsections we present languages, systems and technologies that extend the aforementioned standards in order to incorporate processing of array and grid datasets in the semantic web.

2.3.1 SciSPARQL

SciSPARQL [3] is an extension of the SPARQL 1.1 query language for representing and querying multi-dimensional arrays. As the underlying data model it uses the “Grid Coverage Ontology”, which is an RDF version of OGC coverage model. In SciSPARQL arrays are incorporated as value nodes into the RDF graph and connected to all their metadata. The authors have implemented their proposal using the rasdaman array DBMS in order to push down SciSPARQL subqueries that involve array processing and manipulation, and they use an in-memory RDF store for thematic processing of the fragments of SciSPARQL queries that do not involve array operations. SciSPARQL also employs user defined functions (UDFs) and second order functions in order to facilitate the definition and application of array manipulation operations. An example of a UDF from [3] is the following one that computes the Normalized Difference Vegetation Index (NDVI) on the fly, considering that the RDF graph contains two aligned arrays storing the NIR and RED components as the properties :nir and :red of the node with id ”mycoverage”.

DEFINE FUNCTION NDVI(?nir ?red ?x) **AS**
SELECT (255 * xsd:**integer**(((?nir − ?red) / (?nir + ?red)) > ?x) **AS** ?result)

Then, this UDF can be used in the following example SciSPARQL query (again, query taken from [3]):

SELECT (MAP(xsd:**integer**, NDVI(*, *, $x), ?nir, ?red) **AS** ?result)
WHERE { ?c :id ”mycoverage” ; ex:nir ?nir ; ex:red ?red }

2.3.2 GeoSPARQL+

GeoSPARQL+ [13] is an extension of the GeoSPARQL model, that has been developed in order to integrate geospatial raster data into the Semantic Web. This proposal contains an ontology that defines a new type of geospatial data type for raster, and it also extends the GeoSPARQL query language with new filter functions based on raster algebra operations, e.g. rasterPlus, rasterSmaller etc. Using GeoSPARQL+, combined vector and raster data analysis can be achieved from a single query. Regarding the implementation, the authors modify the Apache Jena RDF library with extra functions and use java libraries to implement all vector and raster operators. As there is no specialized back-end, the scalability of the implementation remains an open issue. An example GeoSPARQL+ query taken from [13] is the following, which is used to find passable roads which are not flooded by more than 10cm in an emergency flooding event:

```
SELECT ?road
WHERE {
    ?road a ex:Road ;
        geo:hasGeometry ?roadseg .
    ?roadseg geo:asWKT ?roadseg wkt .
        ?floodarea a ex:FloodRiskArea ;
    geo2:asCoverage ?floodarea cov .
    ?floodarea cov geo2:asCoverageJSON ?floodarea covjson .
    BIND(geo2:rasterSmaller(?floodarea covjson,10) AS ?relfloodarea)
    FILTER(geo2:intersects(?roadseg wkt,?relfloodarea))
}
```

This query accesses two different datasets. The first dataset contains the vector geometries of roads in a specific area, whereas the second dataset is a raster that contains a value corresponding to flood altitude for each raster cell. The query uses the raster algebra operator “geo2:rasterSmaller”, in order to filter the specific raster values where the flood altitude is larger than 10 centimetres. Then it uses the GeoSPARQL+ function “geo2:intersects” in order to relate the two datasets and obtain roads that intersect with given raster cells. This specific function is an extended version of the GeoSPARQL function “geo:intersects”, which takes as input both raster and vector geometries.

2.3.3 The RDF Data Cube Vocabulary

The RDF Data Cube Vocabulary [1] offers an ontology for modelling and publishing multi-dimensional data on the web. It has been a W3C recommendation since 16 January 2014. It is compatible with the Statistical data and metadata exchange (SDMX) ISO standard for exchanging and sharing statistical data and metadata among organizations, and it extends well-known existing RDF vocabularies such as SKOS, FOAF, Dublin Core etc. This model does not come with an appropriate query language in order to access and process specific data cells of the data cube.

2.4 Ontop: OBDA in other Semantic Frameworks

Ontop [29] (`https://ontop-vkg.org/`) is one of the first OBDA systems able to perform SPARQL to SQL query translation, given a user defined ontology. The result of the process, is an SQL query that is executed on any database instance that follows the input schema, and provides the complete answers with respect to the ontology axioms. Ontop has been successfully deployed in several demanding use cases and has an active community of users and developers, with 154 forks and 549 stars in Github. It has also been commercialized by the Italian company Ontopic (`https://ontopic.ai/en/`).

Ontop-spatial was developed by the AI team of the National and Kapodistrian University of Athens in 2016 [8, 9] (`https://ontop-spatial.di.uoa.gr/`). It is the first geospatial OBDA system and it was implemented as a geospatial extension of Ontop. In Ontop-spatial, the input GeoSPARQL query is transformed into an intermediate form based on

Datalog, and this query is rewritten by taking into consideration the ontology and the mappings from the ontology-concepts to the data sources. The final result is an SQL query that uses spatial SQL functions, which correspond to the GeoSPARQL functions and operators of the initial query. This SQL query can be executed in a spatially-enabled relational system, like PostGIS (the spatial extension of PostgreSQL) or Spatial-Lite (the spatial extension of SQLite). The functionalities of Ontop-spatial have been integrated fully into Ontop as of version 4.1.0. A semantic framework based on Ontop was recently developed by Hamdani et al [12]. By extending the GeoSPARQL ontology, they built a model that combines the semantics of raster layers with feature-based data that involve geometries as well as spatial/topological relationships. Both raster *and* vector data are materialized in relational tables from their raw formats, and subsequently undergo several conversions (using PostGIS functions) in order to conform to the design of the ontology. Such materializations and conversions become very costly as raster data becomes extensive. However, the evaluation of this system did not involve large EO data cubes, and more importantly, its querying capabilities are limited to the operations currently supported by SPARQL. Lastly, there is another similar approach aiming to extend Ontop in a fashion similar to Ontop-spatial, albeit only for transforming vector data in the relational format [11]. The system architecture resembles that of Plato, in that it features a Python wrapper connecting PostgreSQL to the Rasdaman backend. But this wrapper is actually implemented in PL/Python [1], an untrusted language, meaning database security issues could arise in the process. Among other problems that were observed in the description of their pipeline, two significant ones are the potential pre-processing and post-processing of the data, as well as the string format of query results retrieved as filtered arrays, which prevents any further analysis and visualization. Even if most of these issues were to be resolved in the future (after the integration of Ontop), a single query would still have to be expressed using two languages (i.e., SPARQL+rasql), adding further layers of complexity for the end user.

2.5 Summary

This chapter explored the landscape of infrastructures for handling data cubes, such as the Earth System Data Lab (ESDL) data cube, along with a fully-fledged semantic data cube system, Sen2Cube.at, which utilizes computer vision for semantic enrichment. It highlighted specialized data management systems for multi-dimensional arrays, focusing on their relevance in various domains. It discussed the limitations of traditional database systems in handling arrays effectively and introduced specialized array database management systems as solutions, emphasizing their role in developing semantic data cube technology. Various query languages, such as ISO SQL 9075 Part 15 and OGC Web Coverage Processing Service (WCPS), were examined for their capabilities and standards compliance. The intersection of semantic web technologies with grid and array data processing was also explored, including RDF data modeling and extensions like GeoSPARQL+. Finally, Ontop and its role in ontology-based data access (OBDA) systems, including Ontop-spatial for spatial query translation, were discussed alongside other semantic frameworks contributing to spatial data processing.

[1] `https://www.postgresql.org/docs/current/plpython.html`

3. THE SEMANTIC DATA CUBE SYSTEM PLATO

In this chapter we will highlight the architecture of Plato, going in detail on its two main components: the Ontop OBDA system and the PostGIS backend. We will discuss about the implementation of FDWs to facilitate handling large data cubes as PostgreSQL foreign tables. Finally, we will outline the system's various optimizations, focusing on the implementation of caching raster data for efficient operations. This chapter, apart from the sections regarding the Cache and Raptor Join optimizations, has been written jointly with my colleague Anastasios Mantas and, as a result, it also appears verbatim in his MSc thesis [16].

3.1 Architecture

The architecture of Plato is shown in figure 3.1. To facilitate our discussion, we assume that a data cube consists of analysis ready data in four dimensions: latitude, longitude, time and variable of interest. Further dimensions can be added as a result of an analysis.

The two main components of Plato are the OBDA system Ontop and the PostGIS backend. The PostGIS backend contains virtual tables used to communicate with data cubes (stored locally or remotely). This communication is achieved through Python scripts utilizing the Xarray [14] library and the Multicorn package [10], to implement FDWs.

In Plato, data cubes are stored either in a *.zarr* directory format or as *.nc* (netCDF) files. Even with a compressed format like these two, many data cubes are very large to unpack and materialize in a PostgreSQL database. For that matter, we utilize FDWs and the Xarray package so that we can expedite working with labelled multi-dimensional arrays. Xarray introduces labels in the form of dimensions, coordinates and attributes on top of raw NumPy-like arrays, and functions in a similar manner to the Pandas package. Foreign Data Wrappers are libraries that can communicate with external data sources while hiding connection and data retrieval details. Using FDWs and Xarray can turn out to be an intensive operation, both time and memory-wise, and is by no means a realistic approach for large data cubes. For that reason, we decided to also look into parallelization modules of Python. As the FDW applications are not IO bound, multi-threading was not of much benefit, which was confirmed from brief experimentation. Multiprocessing, on the other hand, resulted in massive speedups, through the exploitation of data chunking and dispatched reading by multiple spawned processes, wherever possible.

Lastly, in favour of testing and a successful deployment of the entire pipeline, we have developed a dockerfile to build an image that installs all the necessary components (PostgreSQL, Python3, Multicorn, Xarray, Zarr) and exposes a port to access the database within the created container. Furthermore, the Ontop component regards multiple endpoints that can also be deployed using Docker, one for every user-provided ontology, so that GeoSPARQL querying may be available online.

3.2 Ontop plugin

Plato is implemented using the OBDA system Ontop [29], one of the pioneering OBDA systems capable of performing query translation from SPARQL to SQL. The inputs required for this process are: (i) an ontology expressed in the OWL2 QL subset of the OWL2

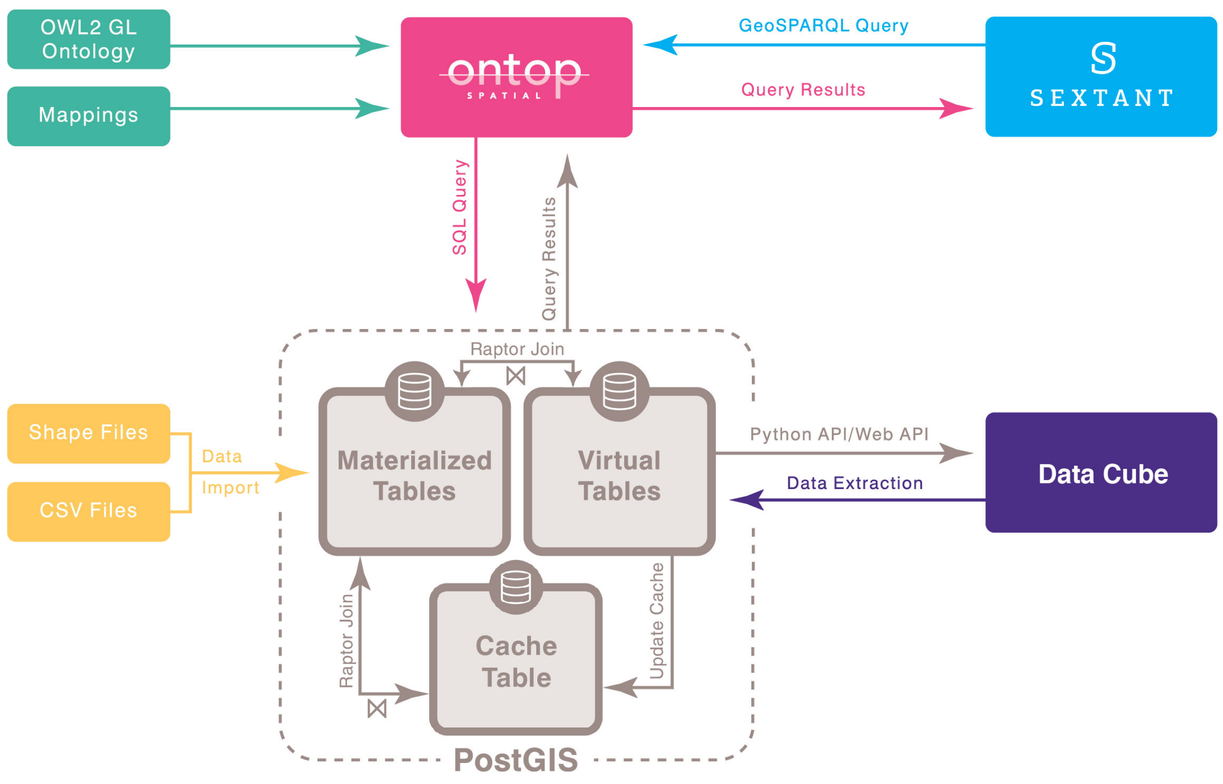

Figure 3.1: The architecture of Plato

ontology language [28], (ii) a database schema, (iii) a set of mapping assertions which generate virtual RDF triples from database records, and (iv) a well-formed GeoSPARQL query targeting the ontology. An SQL query is produced as a result, which can then be executed on any database instance that is in compliance with the input schema, providing comprehensive answers that are consistent with all axioms present in the ontology. In essence, an ontology is defined during the initialization of the Ontop engine, alongside with the specified set of mappings. Ontologies provide a familiar vocabulary for the user in terms of classes and properties, while mappings define the way the ontology terms are related to the data residing in the backend. After initialization, Ontop is ready to accept GeoSPARQL queries and translate them into SQL enhanced with spatial operators.

3.3 PostGIS - Foreign Data Wrappers

Extending the PostgreSQL relational database with PostGIS enables us to store, index, and query geospatial data. This data may or may not be materialized. After creating the necessary extension through the PostgreSQL command line interface, we get access to the geometry data type as well as multiple useful methods to assist us in handling and processing datasets with geospatial information. For instande, we can now utilize various publicly available datasets (*shp, zarr, tiff* files), containing geometries in the form of points or (multi)polygons, in tandem with the data cubes.

The Multicorn package [10] is a PostgreSQL extension which enables us to implement our own FDWs in Python. It assists us in projecting data from the cubes that is requested at query time, in the form of a relational foreign table (connection info is non-transparent). This is achieved through the development of the aforementioned FDWs in Python, which are able to parse the query filters, process the data cubes through Xarray according to those, and retrieve the requested data in the form of tuples. In order to create a FDW, first we need to define a method according to Multicorn's documentation that will be able to access the external data source, in our case a data cube, and return the requested variable based on a given query. The simplest version of that process is as follows:

```
import xarray
from multicorn import ForeignDataWrapper

class CubeForeignDataWrapper(ForeignDataWrapper):

  def __init__(self, options, columns):
    super(CubeForeignDataWrapper, self).__init__(fdw_options, fdw_columns)
      self.columns = fdw_columns
      self.dataset = fdw_options.get('dataset', None)
      self.ds = xarray.open_dataset(self.dataset)

  def execute(self, quals, columns):
    for i in range(self.ds.time.size):
      for j in range(self.ds.y.size):
        for k in range(self.ds.x.size):
          row = {}
          for column in columns:
            if (column == "time"):
              row[column] = self.ds.time.values[i]
            elif (column == "y"):
              row[column] = self.ds.y.values[j]
            elif (column == "x"):
              row[column] = self.ds.x.values[k]
            else:
              row[column] =
                self.ds.get(column).values[i][j][k]
            yield row
```

This was essentially our first approach in handling data cubes as virtual tables in PostgreSQL. Generally speaking, the conversion from multi-dimensional arrays to a tabular format introduces many shortcomings, especially regarding I/O efficiency. Generating rows of the form [*time, y, x, variable*] as shown above means that we have to iterate over the corresponding ndarrays in order to retrieve their values. From the early stages of development, it became apparent that handling large data cubes in this manner was less than desirable. Nevertheless, this problem was mitigated in subsequent versions of each FDW, by implementing multiple optimizations such as handling query filters, slicing the data cube based on user parameters, parallelization of reading and writing, etc. After successfully installing our FDW implementation, we can create a PostgreSQL server that will operate in order to handle any queries directed to the external data source, which in our case will be a data cube. Afterwards, we can create a foreign table in our database to access that data cube, with columns that correspond to the exact same variables found in the data cube. This is done as follows:

```
CREATE SERVER multicorn_cube FOREIGN DATA WRAPPER multicorn
options (
    wrapper 'CubeForeignDataWrapper'
);
```

```
CREATE FOREIGN TABLE public.cubetable (
    ”time” character varying NOT NULL,
    y double precision NOT NULL,
    x double precision NOT NULL,
    ndvi double precision
)
SERVER multicorn_cube
OPTIONS (
    dataset ’/cubes/dataset−greece.nc’
);
```

In this example, we have just created a foreign table for the data cube *dataset-greece.nc*, handling only the three available dimensions as well as its *ndvi* variable. This can obviously be expanded, depending on the use case and dataset requirements.

3.4 Cache

The main idea behind caching raster data in PostGIS by modifying the Ontop plugin, is to efficiently query large volumes of data cubes by having various parts of them be materialized and readily available. In order to achieve this, during Ontop’s query translation from GeoSPARQL to SQL, we can recognize the portions of raster data that need to be accessed and transformed to geometries and save them in an intermediate cache table in the database. This way instead of using on-the-fly transformation and then discarding the data, we can have access to the requested variables and transformations for similar future queries. In order to check if the requested data is already in the cache table, we have implemented data structures as indexes, accessed inside the Ontop plugin. Currently, a hash table is fully implemented for the time dimension of the datasets, with similar functionality for latitude and longitude dimensions being worked on. The following code snippet from the CacheManager class inside the Ontop plugin, performs that check for the time dimension, for a user specified time range and variable (if no range is specified, we have to check all the possible dates for the dataset):

```
//check if data is in time hash (exists in cache table)
String tmp_str;
boolean miss = false;
outerLoop: for (LocalDate tmp_date = min_date; tmp_date.isBefore(max_date);
    tmp_date = tmp_date.plusDays(1)) {
    tmp_str = tmp_date.toString();
    if (!this.timeCache.containsKey(tmp_str)) {
        //if current date is NOT in hash, break
        miss = true;
        break;
    }
    else {
    //if current date IS in hash, check requested variables
        for (String var:variables) {
            //if variable is NOT in hash, break
            if (!this.timeCache.get(tmp_str).contains(var)) {
                miss = true;
                break outerLoop;
                }
            }
        }
}
```

This process might benefit by switching from a simpler structure like the hash table, to a tree structure (R-trees), due to complexity concerns when handling and caching data for multiple dimensions simultaneously. If the requested data is found to be available in the cache table, the query translation process into SQL is modified, in order to access the cache and not a FDW (virtual table). This is performed with the following logic, found in the DataCubeCacheManager class:

```
//modify data node if query can be answered by cache
if (replaceWithCacheTable)
    return DataCubeCacheManager.renameExtensionalDataNode(
        dataNode, coreSingletons, "cache_table");
else
    return dataNode;
...
private static ExtensionalDataNode renameExtensionalDataNode(
        ExtensionalDataNode node, CoreSingletons coreSingletons, String name) {
    //Create an extensional data node to access the cache
    return coreSingletons.getIQFactory().createExtensionalDataNode(new
        DataCubeCacheManager.ValuesRelationDefinition(name, coreSingletons.
        getTypeFactory().getDBTypeFactory().getAbstractRootDBType(), new
        SQLStandardQuotedIDFactory(), node), node.getArgumentMap());
}
```

Although this may look complex, all we do is essentially bypass the default functionality provided by the Ontop plugin, altering the present translation structure in order to create and use a new node that will represent our cache table. When querying for data we are presented with three different cases regarding the cache table: (a) the requested data is not cached at all, (b) the requested data is partially cached, and (c) the requested data is fully cached and we can access it without the need of a FDW. For (a) we can simply update our cache table, by storing the requested data after the process of querying the FDW has returned it. For (b), ideally we would want to use both the cache and a FDW to answer the query, but through testing we found that to be impractical in the context of the Ontop plugin, with its specific pipeline of using a node structure to represent queries. For that reason, we choose to just update the cache with the missing data for future queries, similar to what we did in case (a). An example query to update the cache table can be seen below:

```
ALTER TABLE cache_table
ADD COLUMN IF NOT EXISTS variable double precision;

INSERT INTO cache_table (time, x, y, variable)
SELECT time, x, y, variable FROM foreign_table
WHERE time >= min_date AND time <= max_date +
AND x >= min_x AND x <= max_x
AND y >= min_y AND y <= max_y
AND variable != 'NaN'
ON CONFLICT (time, x, y) DO UPDATE
```

SET variable = EXCLUDED.variable;

Should the cache reach its specified maximum size after a number of user queries, a set replacement policy kicks in, freeing up the necessary space for any newly requested data (e.g. keep the most recent dates). That ease of access to “hot” data is the main benefit of the implementation, as it allows more efficient joins and general operations between dense data cubes and other materialized (non-EO) data. The results of the cache implementation (shown in a later section) are promising for the future of its usage. Furthermore, we can adjust the maximum size and replacement policy accordingly, depending on the specific needs for each use case.

3.5 Raptor Join

After testing various GeoSPARQL queries on large data cubes, we have discovered that accessing large portions of data cubes and transforming each pixel to a vector point (the obvious implementation) created a bottleneck in our system. The idea behind Raptor Join [19] alleviates this problem by reading only parts of the raster that overlap a set of vector geometries (using *scanlines*), without the need for conversions between the two forms in order to perform a join. The Raptor Join method is implemented in Plato as a Python FDW and calculates the result of a spatial operation as output. The requirements are: a set of vector geometries, an EO variable name (raster), an aggregate function name (e.g., sum, max, count, etc.), a spatial relation (intersection currently supported), and a specific time frame as input. Just by adding the properties reflecting these parameters to a given ontology, we are able to create a single mapping to connect Ontop with the FDW operator. Further information on the Raptor Join optimization can be found in [16].

3.6 Summary

The architecture of Plato consists of two main components: the OBDA system Ontop and the PostGIS backend. Ontop facilitates query translation from SPARQL to SQL, utilizing ontology definitions and mapping assertions. PostGIS extends PostgreSQL to handle geospatial data, with Multicorn enabling the implementation of Foreign Data Wrappers (FDWs) in Python. FDWs and Xarray expedite handling large multidimensional arrays, while parallelization, particularly multiprocessing, significantly improves processing speed. Additionally, caching raster data in PostGIS enhances query efficiency. The Raptor Join method optimizes GeoSPARQL queries by reading only relevant portions of data cubes, reducing processing overhead significantly. These components collectively enable efficient querying and analysis of large geospatial datasets in Plato.

4. USING PLATO IN THE DEEPCUBE PROJECT

DeepCube – “Explainable AI pipelines for big Copernicus data” – is a three-year project, funded by the Horizon 2020 program of the European Union under the topic “Big data technologies and Artificial Intelligence for Copernicus”. The project aims to unlock the potential of Copernicus data, leveraging on advances in the fields of Artificial Intelligence and Semantic Web. Plato, which was developed for this project, enables the semantic enrichment of the Earth System Data Cube (ESDC). It allows users to query EO data, other Linked Open Data, and information/knowledge extracted from the data, using a semantic query language, thus creating new value chains. We subsequently present the use cases of DeepCube, where Plato’s contribution is highlighted for each respective approach. Parenthetically, each of the imminent GeoSPARQL queries which are highlighted for the use cases contain predicates of the form <*prefix*>:<property>. For readability, all of the prefixes used are listed beforehand. In a typical environment (endpoint), a query should be preceded by the prefixes involved in it.

```
PREFIX uc2: <http://deepcube−h2020.eu/migration/ontology#>
PREFIX uc3: <http://deepcube−h2020.eu/fire−risk/ontology#>
PREFIX uc5: <http://deepcube−h2020.eu/tourism/ontology#>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
PREFIX uom: <http://www.opengis.net/def/uom/OGC/1.0/>
PREFIX ofn: <http://www.ontotext.com/sparql/functions/>
```

This chapter has been written jointly with my colleague Anastasios Mantas and, as a result, it also appears verbatim in his MSc thesis [16].

4.1 Climate-induced migration in Africa

This use case focuses on how the biosphere and anthroposphere are affected by extreme weather events, such as heatwaves, droughts, and floods, as well as changing climatic circumstances. The climate issue is both caused and impacted by humans, and mitigation and adaptation strategies greatly depend on a knowledge of both effects. The overarching goal of the use case, developed and maintained by the University of Valencia, is to model, anticipate, and understand climate-induced migration flows in Africa from reliable data. The goal is to offer insights into drought-induced displacement trends by utilizing causal time series analysis to identify displacement triggers and quantify their causal relationships and time lags. Additionally, the use case products aim to anticipate and follow-up displacement and provide narratives for these displacements. To achieve this, they need to combine different data sources in order to (i) identify the main environmental and socioeconomic drivers of human mobility and develop models able to reproduce and forecast migration flows, (ii) apply causal discovery methods to gain a deeper understanding of the characteristics of the climate-induced migration flows, and (iii) establish the causal relationships of environmental and socioeconomic drivers with human mobility in sub-Saharan Africa.

To this purpose, they compiled and structured a data cube integrating socioeconomic, environmental and climatic variables and the longest drought displacement time-series existing (covering the 2006-2022 period in Somalia). Through them, causal graphs are derived, applying causal inference at district level. The data we utilized include socioeconomic indexes, displacement data, ERA5 land observations and precipitation from the Climate Hazards Group InfraRed Precipitation with Station.

With the flexibility of FDWs, concepts like the aforementioned variables can be expressed in our ontology [21] as data properties of broader classes, which categorize diverse user requirements. An example query regarding the Correlation class is the following:

```
SELECT ?district ?causal_variable ?causal_link ?mean ?total_mean ?geometry
WHERE {
?corr a geo:Correlation ;
    uc2:hasDistrictName ?district ;
    uc2:causalVar ?causal_variable ;
    uc2:causalLink ?causal_link ;
    uc2:hasMean ?mean ;
    uc2:hasTotalMean ?total_mean ;
    uc2:hasAcquisitionDate ?date ;
    uc2:hasGeometry ?geometry .
FILTER(?causal_variable = ”IDP Drought”)
FILTER(?date > ”2010−01−01 00:00:00” && ?date < ”2013−01−01 00:00:00”)
}
```

In this query we request the districts of Somalia that have a causal link between drought and any other index available. Alongside those, two averages of the drought variable for each district are calculated and returned: ?total_mean, which spans the entire 2006-2022 period, and ?mean, which spans a user-defined 3-year interval (start of 2010 - end of 2012).

Another query which allows us to directly compare metrics between two zones of interest and their variables can be seen below:

```
SELECT ?district_a ?district_b ?diff ?geom_a ?geom_b
WHERE {
?join a uc2:JoinSimilar;
    uc2:hasDistrictA ?district_a ;
    uc2:hasDistrictB ?district_b ;
    uc2:hasVariable ”Water Prices”^^xsd:string ;
    uc2:hasAggregate ”max” ^^xsd:string ;
    uc2:hasTolerance ?tolerance ;
    uc2:hasDiff ?diff ;
    uc2:hasGeomA ?geom_a ;
    uc2:hasGeomB ?geom_b .
    FILTER(?tolerance = 0.25)
}
```

In this instance, we request pairs of Somali districts (and their geometries) having a max water price difference of at most 0.25. The aggregation (e.g., max, min, avg, etc.) for each variable takes all of the available district data into account, spanning from 2006 to 2022. The tolerance filter can be set according to the user's needs.

For this use case, Plato is further used to allow integration of EO (raster) and non-EO (vector) datasets in a unified manner and formulate GeoSPARQL queries that are used to produce thematic maps with the visualization tool Sextant, as narratives for displacements. There are three types of semantic queries that were implemented based on the use case needs: (i) data quality queries to check the quality of a variable's values for each district,

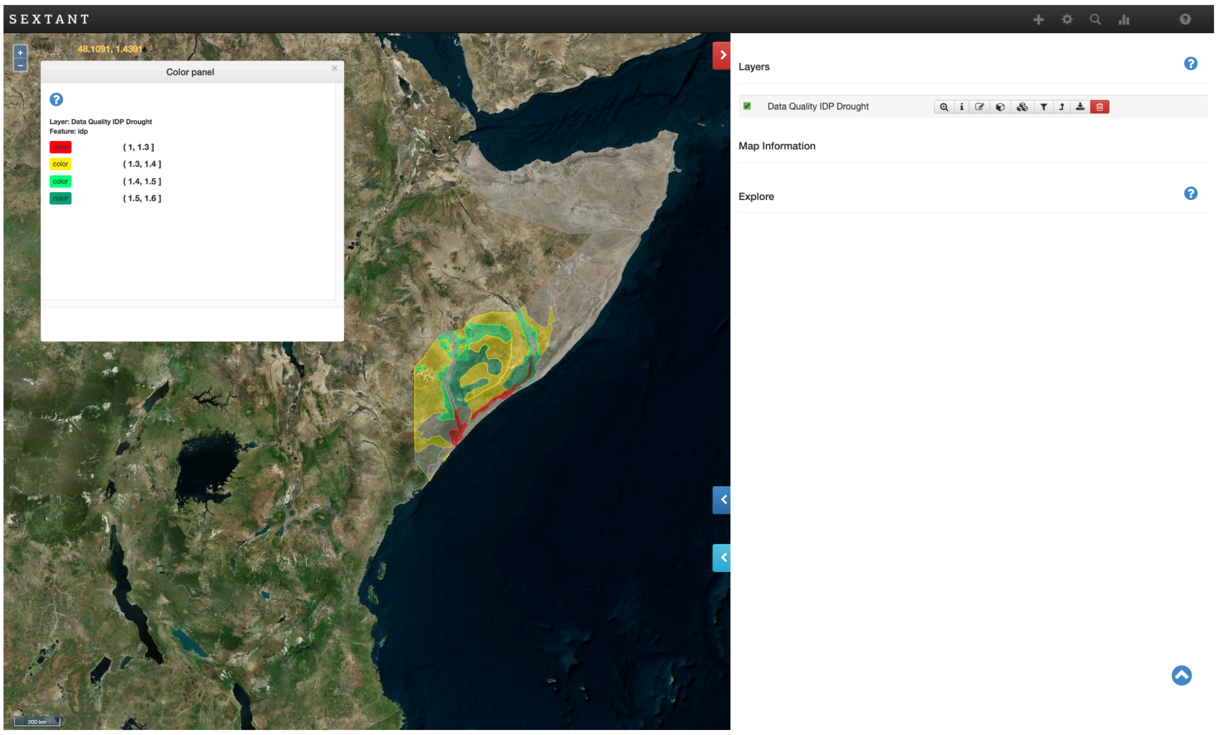

Figure 4.1: Data quality layer in Sextant, for the Internally Displaced Persons (IDP) Drought index. The color map shows districts in Somalia that pass the quality check threshold, along with their quality value.

(ii) early warning queries over the different variables for a specific district and (iii) queries (join) over the causal graphs.

4.2 Fire hazard forecasting

By leveraging AI on large EO and non-EO data, this use case seeks to improve the operational readiness of fire departments and civil protection agencies with fire danger forecasting systems. It looks at a variety of topics, including burned area trends in Mediterranean ecosystems, the anthropogenic drivers that affect fire proneness, the state of the climate and vegetation, modeling of fire hazard to improve future predictions using EO data time-series analysis, and the effect of vegetation recovery dynamics on inter-seasonal fire risk. In order to improve our understanding of the dynamics of fire in complex ecosystems and produce reliable large-scale fire hazard forecasts, hybrid modeling is employed, which combines physical models with data-driven machine learning (ML). The main output products of the use case are: (i) daily fire risk forecasts based on the predictions of the DL models developed by the National Observatory of Athens (NOA), (ii) plots that explain the main reasons behind the model's prediction and (iii) daily fire risk assessment based on asset exposure.

Three types of datasets were provided (along with the corresponding predictions, when available) in the form of data cubes: first, there is a cube for the analysis of wildfires in Greece, containing historical data from 2009 to 2020. Second, the Hellenic Fire Service provided daily EO data cubes (for daytime and night-time) accompanied with their respective fire risk index prediction cubes, for the latter half of 2022. The third and final data cube was compiled by NOA for the modeling and analysis of Wildfires in the entire Mediterranean Sea region (named ”mesogeos”), covering two decades (2002 to 2022). All of the datasets include dynamic variables, such as satellite data from the Moderate Resolution Imaging Spectroradiometer (MODIS), weather variables from ERA5-Land, soil moisture index from JRC European Drought Observatory, as well as static variables, e.g., population count and distance to roads from worldpop.org, land cover from Copernicus Climate Change Service, elevation, aspect, slope and curvature from Copernicus the European Digital Elevation Model (EU-DEM), and burned areas and ignition points from the European Forest Fire Information System (EFFIS).

Plato enhances the pipeline by enabling the semantic enrichment of the available data cubes and allows users to (i) interact with the data cube using higher-level semantic concepts (e.g., land use/land cover classes) instead of only data variables and their values, and (ii) perform combined analysis using the initial data (e.g., observations) and the result of the ML methods (e.g., predictions). An example query on the developed ontology is as follows:

```
SELECT ?name ?fri ?time ?wktPOI ?wktPred
WHERE {
?poi a uc3:PointOfInterest ;
    uc3:hasType ?type ;
    uc3:hasName ?name ;
    geo:asWKT ?wktPOI .
?rastercell a uc3:PredictionRasterCell ;
    geo:asWKT ?wktPred .
?pred a uc3:FirePrediction ;
    uc3:refersToPredictionRC ?rastercell ;
    uc3:hasFireRiskIndex ?fri ;
    uc3:hasAcquisitionDate ?time .
FILTER(geof:distance(?wktPred, ?wktPOI, uom:metre) < 5000)
FILTER(?type = ”archaeological”)
FILTER(?fri > 0.25)
}
```

This query uses the predictions of the historical greek data cube (1km x 1km x 1 daily grid), in order to find if there are any archaeological sites in a distance less than 5 kilometres away from a pixel with a fire risk index above a value of 0.25.

Apart from this instance, the aforementioned raster layers could be combined with other datasets containing geometries in the form of vectors. Plato's database is enriched with many topography layers from OpenStreetMap, as well as protected areas from the Natura 2000 European Ecological Network. A query example, utilizing the Natura areas dataset and a different data cube containing information on the Mediterranean region, can be seen below:

```
SELECT ?result ?label ?wktNaturaGR
WHERE {
?naturaGR a uc3:NaturaArea ;
    rdfs:label ?label ;
    geo:asWKT ?wktNaturaGR .
?join a uc3:Raptor ;
    uc3:hasResult ?result ;
    uc3:hasVariable ”smi” ;
    uc3:hasAggregateFunction ”avg” ;
    uc3:hasSpatialRel ”intersects” ;
    uc3:hasAcquisitionDate ”2022−07−15T00:00:00” ;
    uc3:hasGeometry ?wktNatura .
}
```

This query requests information on the average soil anomaly index of Natura areas for the date July 15th, 2022 and has been enhanced through both the optimizations of the Raptor Join and caching techniques. Combining all the different data sources alongside the non-EO variables present in the larger data cubes, allows users to gather useful statistics for different areas of interest, while Sextant can also be utilized to get specialized visual feedback regarding the same results.

4.3 Copernicus services for sustainable tourism

The main objective of this application is to produce a pricing tool for hotels and package tours, which is independent of the current major booking platforms and which incorporates an environmental and sustainable tourism dimension. Our motivation is to reduce the impact of tourism on planet Earth by implementing a business system based on supply, demand, but also environmental impact. The objectives of this use case are: (i) to characterize the present tourism environmental footprint, tourism demand and tourism offer on the areas of interest, (ii) to develop an engine evaluating the environmental footprint of tourism on a given destination and at a given period, (iii) to develop a pricing engine evaluating downward or upward trend to be applied to a tourism package depending on its destinations and travel periods.

The data cubes used to produce the services, combine air quality information from the Copernicus Atmosphere Service (CAMS) along with weather conditions from the ERA5-Land hourly dataset. We also rely on tourism visit data provided by the Orange FluxVision service and tourism pricing information. FluxVision data is obtained by processing a mix of mobile phone network events and socio-demographic customer data, allowing after data adjustment processes to provide a statistical estimate of a number of people present in an area. The tourism pricing information are extracted from the Amadeus API[1].

For this application, we use Plato to integrate air quality data with tourism data. In order to access the different input sources, we designed an ontology [21] to capture the domain, that allows us to pose GeoSPARQL queries that combine all our sources to analyse the data. A query showcasing links between air quality and tourism frequentation is presented below:

```
SELECT ?area ?avg_no2 ?total_excursionists ?wktAOI
WHERE {
?tf a uc5:Traffic ;
    uc5:hasArea ?area ;
    uc5:hasDate ?date ;
    uc5:hasExcursionists ?total_excursionists .
?aoi a uc5:AreaOfInterest ;
    uc5:hasName ?area ;
    geo:asWKT ?wktAOI .
?join a uc5:Raptor ;
    uc5:hasResult ?avg_no2 ;
```

[1] https://developers.amadeus.com/

```
    uc5:hasVariable ”cams_no2_conc” ;
    uc5:hasAggregateFunction ”avg” ;
    uc5:hasAcquisitionDate ?date ;
    uc5:hasGeometry ?wktAOI .
FILTER(?date = ”2022−01−01T00:00:00”)
}
```

This query utilizes the Raptor Join technique in order to calculate the average nitrogen dioxide value for several areas of France, in a particular day. For the same spatiotemporal input, the total number of day-trippers is also requested. We could alternatively ask for different types of visitors, and/or different CAMS variables, as shown below:

```
SELECT ?co2 ?total_visitors ?wktAOI
WHERE {
?tf a geo:Traffic ;
    uc5:hasArea ”Moissac” ;
    uc5:hasDate ?dateVisit ;
    uc5:hasTotalVisitors ?total_visitors .
?aoi a uc5:AreaOfInterest ;
    uc5:hasName ”Moissac” ;
    uc5:hasGeom ?wktAOI .
?join a uc5:Raptor ;
    uc5:hasResult ?co2 ;
    uc5:hasVariable ”cams_co2_conc” ;
    uc5:hasAggregateFunction ”avg” ;
    uc5:hasSpatialRel ”intersects” ;
    uc5:hasAcquisitionDate ?date ;
    uc5:hasGeom ?wktAOI .
FILTER (?total_visitors > 10000) .
FILTER(
    ?dateVisit >= ”2022−07−01T00:00:00” &&
    ?dateVisit <= ”2022−08−31T00:00:00”
) .
FILTER(ofn:daysBetween(?dateVisit, ?date) < 1)
}
```

In the above query we request the average carbon dioxide concentration, in a specific area (”Moissac”), between the months of July and August of 2022, when the total number of visitors exceeds ten thousand. The last filter is required because the observations from the data cube are hourly, while the tourism frequentation data for each area is daily.

Finally, the returned results of both queries can be viewed as a thematic map with the areas of interest using Sextant, to provide a visual interpretation for our end users.

4.4 Summary

This chapter highlights Plato’s pivotal role within the DeepCube project, demonstrating its application across three diverse use cases. Firstly, in climate-induced migration in Africa, Plato facilitates the understanding of migration flows by combining socioeconomic and environmental variables. Secondly, in fire hazard forecasting, Plato enhances forecasting systems by enabling semantic enrichment and combined analysis of observational and predictive data. Lastly, in Copernicus services for sustainable tourism, Plato supports the research for the development of pricing tools and environmental impact assessments. Throughout these applications, Plato's capabilities in semantic enrichment, spatial analysis, and data integration play a crucial role in deriving actionable insights from complex datasets.

5. EXPERIMENTAL EVALUATION OF PLATO

In this chapter we will present an assessment of query performance on Plato. We begin with an overview of the datasets used for benchmarking, encompassing diverse geospatial and temporal dimensions. The first aspect is the evaluation of the impact of caching mechanisms, while the second one is the exploration of the efficacy of the Raptor Join method, compared to simpler join techniques. This chapter has been written jointly with my colleague Anastasios Mantas and, as a result, it also appears verbatim in his MSc thesis [16].

5.1 Data cubes and datasets

The experiments shown in this section were performed using Ontop 4.2, on a 64-bit machine with 32 logical processors (2.20 GHz), and 128GB of RAM (DDR3, 1600 MHz). Five different data cubes are used in the experiments, with time, latitude and longitude as the primary dimensions, along with several data variables:

- **DC-GR-1 (4314×562×700)**. NDVI data for Greece for the period 2009-2020.
- **DC-GR-2 (1×940×1328)**. Daily relative humidity for Greece for the year 2022.
- **DC-BR (2160×200×200)**. NDVI data for Brazil for the year 2019.
- **DC-SI (8762×150×310)**. Hourly total precipitation data for Slovenia for the year 2021.
- **DC-FR (8760×334×636)**. Hourly total precipitation data for France for the year 2022.

Alongside those, we utilize imported vector data concerning fire prediction data for Greece (point geometries; 2022), Natura-protected areas for Europe (multipolygon geometries), and administrative data for Brazil (polygon geometries). The sizes for all of the datasets are displayed in Table 5.1.

Table 5.1: Raster and Vector datasets

Raster (GB)	
Data cube	Size
DC-GR-1	10
DC-GR-2	0.5
DC-BR	5.9
DC-SI	15.5
DC-FR	35.5

Vector (MB)	
Dataset	Size
Fire Prediction Data	25
Natura Areas (GR)	7
Natura Areas (EU)	78
Brazil Admin. Data	0.5

Since to our knowledge there is no equivalent semantic data cube system to Plato, the evaluation consists of experiments with different query types over the presented data sources, and the results of our optimization techniques. We start with the evaluation of the cache implementation. The different query types selected to evaluate query-execution performance when using a cache table are shown below. These examples are based on an arbitrary ontology, just to provide a template for each query structure. The range of dates for our tests (cases (B) and (D)) was 3-5 days.

- (A) - Requesting variable for one day:

```
SELECT ?wktObs ?value
WHERE {
?rastercellObs a uc:ObservationRasterCell ;
     geo:asWKT ?wktObs .
?var a uc:Variable ;
     uc:refersToObservationRC ?rastercellObs ;
     uc:hasValue ?value ;
     uc:hasAcquisitionDate ?timeObs .
FILTER(?timeObs = ”2023−06−01T00:00:00”) .
FILTER(?value < 0.5)
}
```

- (B) - Requesting variable for a range of dates:

```
SELECT ?wktObs ?value
WHERE {
?rastercellObs a uc:ObservationRasterCell ;
     geo:asWKT ?wktObs .
?var a uc:Variable ;
     uc:refersToObservationRC ?rastercellObs ;
     uc:hasValue ?value ;
     uc:hasAcquisitionDate ?timeObs .
FILTER(
?timeObs >= ”2023−06−01T00:00:00” &&
?timeObs <= ”2023−06−30T00:00:00”
) .
FILTER(?value < 0.5)
}
```

- (C) - Requesting variable to join vector areas with raster predictions (daily), for one day:

```
SELECT ?label ?wktPrediction ?time ?value
WHERE {
?geom a uc:VectorArea ;
     rdfs:label ?label ;
     geo:asWKT ?wktGeometry .
?rastercell a uc:PredictionRasterCell ;
     geo:asWKT ?wktPrediction .
?var a uc:Variable ;
     uc:refersToPredictionRC ?rastercellPred ;
     uc:hasValue ?value ;
     uc:hasAcquisitionDate ?timePred .
FILTER(geof:distance(?wktPrediction, ?wktGeometry, uom:metre) < 1000)
FILTER(?timePred = ”2023−06−01T00:00:00”)
FILTER(?value > 0.5)
}
```

- (D) - Requesting variable to join vector areas with raster predictions (daily), for a range of dates:

```
SELECT ?label ?wktPrediction ?time ?value
WHERE {
?geom a uc:VectorArea ;
    rdfs:label ?label ;
    geo:asWKT ?wktGeometry
?rastercell a uc:PredictionRasterCell ;
    geo:asWKT ?wktPrediction .
?var a uc:Variable ;
    uc:refersToPredictionRC ?rastercellPred ;
    uc:hasValue ?value ;
    uc:hasAcquisitionDate ?timePred .
FILTER(geof:distance(?wktPrediction, ?wktGeometry, uom:metre) < 1000)
FILTER(
?timeObs >= ”2023−06−01T00:00:00” &&
?timeObs <= ”2023−06−30T00:00:00”
)
FILTER(?value > 0.5)
}
```

5.2 Query evaluation

The results for the cache implementation on queries posed to all the data cubes are shown in Table 5.2. We can see that by utilizing a cache table we can overcome the overhead of retrieving the requested data from a foreign table through the use of FDWs. This is more apparent in queries that concern a range of dates (Types B and D), where the cache implementation shows by far the best results. In this way, it is clear that having readily available ”hot” data for multiple requested dates by materializing them, improves the overall user experience.

Table 5.2: Query execution times with cache implementation

Cache (sec)			
Data Cube	Type	Default	Cache
DC-GR-1	A	72.3	20.2
DC-GR-1	B	243	60.6
DC-GR-1	C	1038	953
DC-GR-1	D	>18000	2221
DC-GR-2	A	28.6	1.06
DC-GR-2	B	96.8	7.65
DC-GR-2	C	29.7	1.26
DC-GR-2	D	90.2	2.38
DC-BR	A	6.83	2.22
DC-BR	C	13.6	10.8
DC-SI	A	5.61	2.51
DC-SI	C	72.9	8.31
DC-FR	A	26.5	11.1
DC-FR	C	5121	168

We continue with the evaluation of the join queries. For benchmarking needs, a method simpler than Raptor Join is also implemented, which checks if the pixels within the Minimum Bounding Rectangle (MBR) of input vectors coincide with the actual geometries (requires pixel-to-point transformations). We can also evaluate the performance of our entire pipeline by allowing the Raptor Join implementation to utilize the available data found in the cache table for each of the data cubes and posing the same queries as before. The performance results for those queries for all the different techniques are shown in Table 5.3. The default way to handle a join query is as in case (C) shown in the previous section, while a template for the other two join methods could be written as follows:

```
SELECT ?result ?label ?wktGeometry
WHERE {
?geom a uc:VectorArea ;
    rdfs:label ?label ;
    geo:asWKT ?wktGeometry .
?join a uc:JoinMethod ;
    uc:hasResult ?result ;
    uc:hasVariable ”ndvi” ;
    uc:hasAggregateFunction ”avg” ;
    uc:hasSpatialRel ”intersects” ;
    uc:hasAcquisitionDate ”2022−08−01T00:00:00” ;
    uc:hasGeometry ?wktGeometry .
}
```

Table 5.3: Query execution times with Raptor join and Raptor-Cache combined

Join Results (sec)				
Data Cube	Default	MBR	Raptor	Raptor–Cache
DC-GR-1	1038	139	54	40.2
DC-BR	15.1	21.6	20.9	25.8
DC-SI	72.3	90.4	46.5	17.9
DC-FR	5121	299	137	75.2

The table shows the benefits of using the various join optimizations that we implemented instead of using the default FDWs for data retrieval and letting PostGIS handle the joins by making the necessary pixel-to-point transformations. First of all, the available data (both raster and vector) for Brazil is not very big, so the extra parsing/preprocessing in such cases seems to have an inverse effect on the total time efficiency. Furthermore, the vector data for France has geometries spread around the area of the entire country, while DC-FR is limited to the region of Occitanie. Hence, MBRs are calculated for many non-overlapping geometries, in which case Cache availability allows for the better handling of transformations by PostGIS rather than the use of the MBR technique's FDW. Finally, it is clear that both the simpler MBR method as well as the Raptor Join method generally offer a more efficient approach than handling join queries using standard PostGIS and the default FDWs. Caching data for specific observations and timeframes, when combined with Raptor Join, provide the best speedup (for data of substantial size).

6. CONCLUSIONS AND FUTURE WORK

We presented Plato, a semantic data cube system using OBDA technologies. For the evaluation of Plato, we used data from use cases of the DeepCube project, and showed that the optimizations of caching and Raptor Join allowed us to handle many classes of semantic (GeoSPARQL) queries in an efficient manner. These techniques proved to be significant in alleviating the core issue inherent to the architecture, i.e. projecting data from data cubes onto relational tables, while also reducing I/O load from large datasets. In closing, we discussed other relevant studies/frameworks of semantic systems and underlined the ways that Plato differs from them.

In the future steps of our study, we pinpoint key areas for optimization, specifically focusing on Raptor Join, Caching, and the implementation of a holistic distributed system. For Raptor Join, our exploration into multiprocessing aims to enhance processing speed and efficiency, particularly addressing the effective handling of multiple vector geometries within the system. Simultaneously, in the domain of Caching, the integration of an R-tree index is anticipated to expedite the retrieval of raster data from cubes, optimizing data access. Extending our focus to a holistic distributed system, we aim to address challenges associated with large data cubes, particularly in terms of access and conversion to tabular form. This integrated approach not only refines the performance of individual components but also lays the foundation for a more scalable and efficient system, aligning with the dynamic demands of our application domain.

ABBREVIATIONS - ACRONYMS

AI	Artificial Intelligence
RDF	Resource Description Framework
OBDA	Ontology-based Data Access
EO	Earth Observation
API	Application Programming Interface
DBMS	Data Base Management Systems
FDW	Foreign Data Wrappers
OWL	Web Ontology Language
ESDL	Earth System Data Lab
CV	Computer Vision
MDA	Multi-Dimensional Arrays
OGC	Open Geospatial Consortium
WCPS	Web Coverage Processing Service
XML	Extensive Markup Language
UDF	User Defined Function
SDMX	Statistical Data and Metadata Exchange
IO	Input/Output
MODIS	Moderate Resolution Imaging Spectroradiometer
EU-DEM	European Digital Elevation Model
EFFIS	European Forest Fire Information System
ML	Machine Learning
DL	Deep Learning

BIBLIOGRAPHY

[1] RDF 1.1 concepts and abstract syntax, W3C.

[2] SPARQL 1.1 query language.

[3] Andrej Andrejev, Dimitar Misev, Peter Baumann, and Tore Risch. Spatio-temporal gridded data processing on the semantic web. In *IEEE International Conference on Data Science and Data Intensive Systems, DSDIS 2015, Sydney, Australia, December 11-13, 2015*, pages 38–45. IEEE Computer Society, 2015.

[4] Hannah Augustin, Martin Sudmanns, Dirk Tiede, Stefan Lang, and Andrea Baraldi. Semantic earth observation data cubes. *Data*, 4(3):102, 2019.

[5] Andrea Baraldi. Satellite image automatic mapper - SIAM™, 2001.

[6] Peter Baumann, Paula Furtado, Roland Ritsch, and Norbert Widmann. The RasDaMan approach to multidimensional database management. In Barrett R. Bryant, Janice H. Carroll, Dave Oppenheim, Jim Hightower, and K. M. George, editors, *Proceedings of the 1997 ACM symposium on Applied Computing, SAC'97, San Jose, CA, USA, February 28 - March 1*, pages 166–173. ACM, 1997.

[7] Peter Baumann, Dimitar Misev, Vlad Merticariu, and Bang Pham Huu. Array databases: concepts, standards, implementations. *J. Big Data*, 8(1):28, 2021.

[8] Konstantina Bereta and Manolis Koubarakis. Ontop of geospatial databases. In Paul Groth, Elena Simperl, Alasdair J. G. Gray, Marta Sabou, Markus Krötzsch, Freddy Lécué, Fabian Flöck, and Yolanda Gil, editors, *The Semantic Web - ISWC 2016 - 15th International Semantic Web Conference, Kobe, Japan, October 17-21, 2016, Proceedings, Part I*, volume 9981 of *Lecture Notes in Computer Science*, pages 37–52, 2016.

[9] Konstantina Bereta, Guohui Xiao, and Manolis Koubarakis. Ontop-spatial: Ontop of geospatial databases. *J. Web Semant.*, 58, 2019.

[10] Ronan Dunklau and Florian Mounier. Multicorn - PostgreSQL extension, 2015.

[11] Arka Ghosh, Mantas Simkus, and Diego Calvanese. Semantic querying of integrated raster and relational data: A virtual knowledge graph approach. In Jan Vanthienen, Tomás Kliegr, Paul Fodor, Davide Lanti, Dörthe Arndt, Egor V. Kostylev, Theodoros Mitsikas, and Ahmet Soylu, editors, *Proceedings of the 17th International Rule Challenge and 7th Doctoral Consortium @ RuleML+RR 2023 co-located with 19th Reasoning Web Summer School (RW 2023) and 15th DecisionCAMP 2023 as part of Declarative AI 2023, Oslo, Norway, 18 - 20 September, 2023*, volume 3485 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2023.

[12] Younes Hamdani, Guohui Xiao, Linfang Ding, and Diego Calvanese. An ontology-based framework for geospatial integration and querying of raster data cube using virtual knowledge graphs. *ISPRS Int. J. Geo Inf.*, 12(9):375, 2023.

[13] Timo Homburg, Steffen Staab, and Daniel Janke. GeoSPARQL+: Syntax, semantics and system for integrated querying of graph, raster and vector data. In Jeff Z. Pan, Valentina A. M. Tamma, Claudia d'Amato, Krzysztof Janowicz, Bo Fu, Axel Polleres, Oshani Seneviratne, and Lalana Kagal, editors, *The Semantic Web - ISWC 2020 - 19th International Semantic Web Conference, Athens, Greece, November 2-6, 2020, Proceedings, Part I*, volume 12506 of *Lecture Notes in Computer Science*, pages 258–275. Springer, 2020.

[14] Stephan Hoyer and Joe Hamman. xarray: N-D labeled arrays and datasets in Python. *Open Research Software*, 5(1), 2017.

[15] Kostis Kyzirakos, Manos Karpathiotakis, and Manolis Koubarakis. Strabon: A semantic geospatial DBMS. In Philippe Cudré-Mauroux, Jeff Heflin, Evren Sirin, Tania Tudorache, Jérôme Euzenat, Manfred Hauswirth, Josiane Xavier Parreira, Jim Hendler, Guus Schreiber, Abraham Bernstein, and Eva Blomqvist, editors, *The Semantic Web - ISWC 2012 - 11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part I*, volume 7649 of *Lecture Notes in Computer Science*, pages 295–311. Springer, 2012.

[16] Anastasios Mantas. The semantic data cube system plato and its spatial join optimization, 2024.

[17] Baumann Peter, Rossi Angelo Pio, Bell Brennan, Clements Oliver, Evans Ben, Hoenig Heike, Hogan Patrick, Kakaletris George, Koltsida Panagiota, Mantovani Simone, Marco Figuera Ramiro, Merticariu Vlad, Misev Dimitar, Pham Huu, Siemen Stephan, and Wagemann Julia. Fostering cross-disciplinary earth science through datacube analytics. *Earth Observation Open Science and Innovation*, pages 91–119, 01 2018.

[18] Florin Rusu and Yu Cheng. A survey on array storage, query languages, and systems. *CoRR*, abs/1302.0103, 2013.

[19] Samriddhi Singla. Raptor: Large scale processing of big raster + vector data. In Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava, editors, *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, pages 2905–2907. ACM, 2021.

[20] Pierre Soille, Stefanie Lumnitz, and Sergio Albani. Proceedings of the 2023 conference on Big Data from Space. In *BiDS - 2023*, number KJ-05-23-390-EN-N (online) in Big Data from Space, Luxembourg (Luxembourg), 2023. European Commission, Publications Office of the European Union.

[21] George Stamoulis. Deepcube: Ontologies for semantic data cubes, 02 2023.

[22] Michael Stonebraker, Paul Brown, Alex Poliakov, and Suchi Raman. The architecture of SciDB. In Judith Bayard Cushing, James C. French, and Shawn Bowers, editors, *Scientific and Statistical Database Management - 23rd International Conference, SSDBM 2011, Portland, OR, USA, July 20-22, 2011. Proceedings*, volume 6809 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2011.

[23] Martin Sudmanns, Hannah Augustin, Brian Killough, Gregory Giuliani, Dirk Tiede, Alex Leith, Fang Yuan, and Adam Lewis. Think global, cube local: an earth observation data cube's contribution to the digital earth vision. *Big Earth Data*, 7(3):831–859, 2023.

[24] Martin Sudmanns, Hannah Augustin, Lucas van der Meer, Andrea Baraldi, and Dirk Tiede. The austrian semantic EO data cube infrastructure. *Remote. Sens.*, 13(23):4807, 2021.

[25] Martin Sudmanns, Hannah Augustin, Lucas van der Meer, Christian Werner, Andrea Baraldi, and Dirk Tiede. One GUI to rule them all: Accessing multiple semantic EO data cubes in one graphical user interface. *GI Forum*, Volume 1:53–59, 2021.

[26] Dirk Tiede, Andrea Baraldi, Martin Sudmanns, Mariana Belgiu, and Stefan Lang. Sen2Cube.at: Semantic earth observation data cube analysis, 2021.

[27] Lucas van der Meer, Martin Sudmanns, Hannah Augustin, Andrea Baraldi, and Dirk Tiede. Semantic querying in earth observation data cubes. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLVIII-4/W1-2022:503–510, 2022.

[28] W3C. OWL 2 web ontology language profiles (second edition), 2012.

[29] Guohui Xiao, Davide Lanti, Roman Kontchakov, Sarah Komla-Ebri, Elem Güzel Kalayci, Linfang Ding, Julien Corman, Benjamin Cogrel, Diego Calvanese, and Elena Botoeva. The virtual knowledge graph system Ontop. In Jeff Z. Pan, Valentina A. M. Tamma, Claudia d'Amato, Krzysztof Janowicz, Bo Fu, Axel Polleres, Oshani Seneviratne, and Lalana Kagal, editors, *The Semantic Web - ISWC 2020 - 19th International Semantic Web Conference, Athens, Greece, November 2-6, 2020, Proceedings, Part II*, volume 12507 of *Lecture Notes in Computer Science*, pages 259–277. Springer, 2020.

[30] Ying Zhang, Martin L. Kersten, and Stefan Manegold. SciQL: array data processing inside an RDBMS. In Kenneth A. Ross, Divesh Srivastava, and Dimitris Papadias, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, pages 1049–1052. ACM, 2013.