ExtremeEarth

H2020 - 825258

Deliverable

D3.5

Evaluation framework for linked geospatial data systems

Antonis Troumpoukis, Nefeli Prokopaki-Kostopoulou, Giannis Mouchakis, Babis Kostopoulos, Angelos Charalambidis, Stasinos Konstantopoulos, Dimitris Bilidas, Theofilos Ioannidis, Michail Mitsios, George Smyris and Manolis Koubarakis

December 31, 2021

Status: FINAL
Scheduled Delivery Date: 31 December 2021

Executive Summary

This deliverable is developed in the context of WP3, objective of which is to develop a set of tools for querying, integration and extreme analytics for the big information and knowledge that was mined from Copernicus data and other auxiliary data sources using the techniques developed in WP2. This information and knowledge was encoded as linked geospatial data and was integrated with other open linked data sources to be demonstrated in the two use cases of ExtremeEarth.

Deliverable D3.5 concerns Task 3.5 *evaluation framework for big linked geospatial data systems* (months 12-36). In this task we develop an evaluation platform for the techniques and implemented systems developed in Tasks 3.1-3.4. The framework that we develop is based on the benchmark Geographica of partner UoA, which is the state-of-the-art benchmark for geospatial RDF stores that takes into account the most recent advances in this area. We extend Geographica so that it can be used to evaluate the performance of the transformation, interlinking, querying and integration systems of Tasks 3.1-3.4. Parts of the data sources and scenarios of the extension of Geographica come from the two use cases of the project. The new version of Geographica is made available in the new version of the KOBE Benchmarking Engine (partner NCSR participates in H2020 project HOBBIT and has developed KOBE).

As mentioned in both D3.7 and D3.8, Deliverable D3.5 also contains the final experimental evaluation of Strabo2 (developed in T3.3) and Semagrow (developed in T3.4).

Document Information

Contract Number	H2020 - 825258	**Acronym**	ExtremeEarth
Full title	ExtremeEarth		
Project URL	`http://earthanalytics.eu/`		
EU Project Officer	Riku Leppänen		

Deliverable	**Number**	D3.5	**Name**	Evaluation framework for linked geospatial data systems			
Task	**Number**	T3.5	**Name**	Evaluation framework for big linked geospatial data systems			
Work package	**Number**			WP3			
Date of delivery	**Contract**	31 December 2021		**Actual**	31 December 2021		
Status	Draft ☐ Final ☑						
Nature	Prototype ☑ Report ☐						
Distribution Type	Public ☑ Restricted ☐						
Responsible Partner	NCSR-D						
QA Partner	UoA						
Contact Person	Stasinos Konstantopoulos						
	Email	konstant@iit.demokritos.gr		**Phone**	+30 210 650 3194	**Fax**	n/a

Project Information

This document is part of a research project funded by the IST Programme of the Commission of the European Communities as project number H2020-825258. The beneficiaries in this project are the following:

Partner	Acronym	Contact
National and Kapodistrian University of Athens Department of Informatics and Telecommunications (Coordinator)	UoA	Prof. Manolis Koubarakis National and Kapodistrian University of Athens Dept. of Informatics and Telecommunications Panepistimiopolis, Ilissia, GR-15784 Athens, Greece Email: (koubarak@di.uoa.gr) Tel: +30 210 7275213, Fax: +30 210 7275214
VISTA Geowissenschaftliche Fernerkundung GmbH	VISTA	Heike Bach Email: (bach@vista-geo.de)
The Arctic University of Norway Deptartment of Physics and Technology	UiT	Torbjørn Eltoft Email: (torbjorn.eltoft@uit.no)
University of Trento Department of Information Engineering and Computer Science	UNITN	Lorenzo Bruzzone Email: (lorenzo.bruzzone@unitn.it)
Royal Institute of Technology	KTH	Seif Haridi Email: (haridi@kth.se)
National Center for Scientific Research - Demokritos	NCSR-D	Vangelis Karkaletsis Email: (vangelis@iit.demokritos.gr)
Deutsches Zentrum für Luft-und Raumfahrt e. V.	DLR	Corneliu Octavian Dumitru Email: (corneliu.dumitru@dlr.de)
Polar View Earth Observation Ltd.	PolarView	David Arthurs Email: (david.arthurs@polarview.org)
METEOROLOGISK INSTITUTT	METNO	Nick Hughes Email: (nick.hughes@met.no)
Logical Clocks AB	LC	Jim Dowling Email: (jim@logicalclocks.com)
United Kingdom Research and Innovation - British Antarctic Survey	UKRI-BAS	Andrew Fleming Email: (ahf@bas.ac.uk)

Contents

List of Figures

List of Tables

1. Introduction

This deliverable is developed in the context of WP3, objective of which is to develop a set of tools for querying, integration and extreme analytics for the big information and knowledge that was mined from Copernicus data and other auxiliary data sources using the techniques developed in WP2. This information and knowledge was encoded as linked geospatial data and was integrated with other open linked data sources to be demonstrated in the two use cases of ExtremeEarth.

Deliverable D3.5 concerns Task 3.5 *evaluation framework for big linked geospatial data systems* (months 12-36). In this task we develop an evaluation platform for the techniques and implemented systems developed in Tasks 3.1-3.4. The framework that we develop is based on the benchmark Geographica of partner UoA, which is the state-of-the-art benchmark for geospatial RDF stores that takes into account the most recent advances in this area. We extend Geographica so that it can be used to evaluate the performance of the transformation, interlinking, querying and integration systems of Tasks 3.1-3.4. Parts of the data sources and scenarios of the extension of Geographica come from the two use cases of the project. The new version of Geographica is made available in the new version of the KOBE Benchmarking Engine (partner NCSR participates in H2020 project HOBBIT and has developed KOBE).

As mentioned in both D3.7 and D3.8, Deliverable D3.5 also contains the final experimental evaluation of Strabo2 (used for querying big linked geospatial data and developed in T3.3) and Semagrow (used for federating big linked geospatial data and developed in T3.4). In these experiments, we combine the cluster-level scalability offered by Strabo2 with Semagrow's ability to transparently federate multiple such clusters. The aim is to prove that the combination of these key ExtremeEarth technologies can bring geospatial linked data query processing to the order of magnitude of petabytes.

The rest of the deliverable is organized as follows:

- In Chapter 2 we present the new version of the KOBE Benchmarking Engine, which is the first part of the evaluation framework of Task T3.5. KOBE uses modern containerization and Cloud computing technologies for automating the process of deployment, initialization, experiment execution, and results presentation.
- In Chapter 3 we present a series of new benchmarks that extend Geographica and comprise the second part of the evaluation framework of Task T3.5. These experiments are Geographica2 (for single node linked geospatial stores), Geographica3 (for distributed big linked geospatial stores), and GeoFedBench (for federated linked geospatial data).
- In Chapter 4 we present the final experimental evaluation of Strabo2 using the Geographica3 benchmark. Apart from this, we evaluate specific aspects of the system, and we present the execution times for real world queries and datasets from the use cases.
- In Chapter 5 we present the final experimental evaluation of Semagrow using the GeoFedBench benchmark (which contains datasets and queries from the Food Security use case).
- In Chapter 6 we perform an experiment with data and queries from Geographica2 and a federation of a large number of endpoints to demonstrate scalability to the PB level.
- Finally, in Chapter 7 we conclude the deliverable.

2. The KOBE Benchmarking Engine

In the SPARQL query processing community, as well as in the wider databases community, benchmark reproducibility is based on releasing datasets and query workloads. However, this paradigm breaks down for federated query processors, as these systems do not manage the data they serve to their clients but provide a data-integration abstraction over the actual query processors that are in direct contact with the data. As a consequence, benchmark results can be greatly affected by the performance and characteristics of the underlying data services. This is further aggravated when one considers benchmarking in more realistic conditions, where internet latency and throughput between the federator and the federated data sources is also a key factor.

In this chapter we present KOBE [8, 9], a benchmarking system that leverages modern containerization and Cloud computing technologies in order to reproduce collections of data sources. In KOBE, data sources are formally described in more detail than what is conventionally provided, covering not only the data served but also the specific software that serves it and its configuration as well as the characteristics of the network that connects them. KOBE provides a specification formalism and a command-line interface that completely hides from the user the mechanics of provisioning and orchestrating the benchmarking process on Kubernetes-based infrastructures; and of simulating network latency. Finally, KOBE automates the process of collecting and comprehending logs, and extracting and visualizing evaluation metrics from these logs.

2.1 Introduction

Data federation and distributed querying are key technologies for the efficient and scalable consuming of data in the decentralized and dynamic environment of the Semantic Web. Several federation systems have been proposed [18, 5, 3], each with their own characteristics, strengths, and limitations. Naturally, consistent and reproducible benchmarking is a key enabler of the relevant research, as it allows these characteristics, strengths, and limitations to be studied and understood.

There are several benchmarks that aim to achieve this, but, similarly to the wider databases community, to release a benchmark amounts to releasing datasets, query workloads, and, at most, a benchmark-specific evaluation engine for executing the query load [6, 17, 16]. Research articles using these benchmarks need to specify what software has been used to implement the SPARQL endpoints, how it has been configured and distributed among hardware nodes, and the characteristics of these nodes and of the network that connects them to the federation system. Reproducing an experiment from such a description is a challenging and tedious task. Based on our own experience with federated query processing research we have been looking for ways to minimize the effort required and the uncertainty involved in replicating experimental setups from the federated querying literature. Our first step in that direction was to complement a benchmark we previously proposed [19] with Docker images of the populated triple store installations and of the federation systems used for that work.

In this chapter we present KOBE, an open-source[1] benchmarking engine that reads benchmark definitions and handles the distributed deployment of the data sources and the actual execution of the experiment. This includes instantiating a data source from dataset files, configuring and initializing the federation engine, connecting them into a virtual network with controlled characteristics, executing the experiment, and collecting the evaluation results. The main objective of KOBE is to provide a generic and controlled benchmarking framework where any combination of datasets, query loads, querying scenarios, and federation engines can be tested. To meet this goal, KOBE leverages modern Cloud-native technologies for the containerization and orchestration of different components.

[1] See `https://github.com/semagrow/kobe`

In this chapter we will first introduce the core concepts of a federated query processing experiment and the requirements for consistently and reproducibly carrying out such experiments (Section 2.2) and then present KOBE, its system components and how experiments are provisioned and orchestrated (Section 2.3). We then discuss how logs are collected and evaluation metrics visualized (Section 2.4), and how users can extend the library of benchmarks and federation engines to prepare their own experiments (Section 2.5). We close with a comparison to related systems (Section 2.6), conclusions and future work (Section 2.7).

2.2 Benchmarking Concepts and Requirements

We start by discussing the requirements for a benchmarking experiment of a federated query processor. First, we briefly introduce the main concepts of a federated query processing experiment:

Data source: An endpoint that processes queries. A data source is characterized by a dataset label, with data sources characterized by the same dataset serving the exact same data.

Benchmark: A collection of data sources, the latency and throughput of these data sources, and a list of query strings. Benchmarks are defined independently of the federator that is being benchmarked.

Federator: A federated query processor that provides a single endpoint to achieve uniform and integrated access to the data sources.

Experiment run: A specific experiment, where (a) a specific federator has been configured to be able to connect to the data sources foreseen by the benchmark; and (b) the query load foreseen by the benchmark has been applied to the federator.

Experiment: The repetition of multiple runs of the same benchmark. An experiment is stateful, in the sense that the federator and data source instances are not terminated and maintain their caches and, in general, their state between runs.

Having these elements in place allows for the following tests, commonly used to evaluate query processing systems in general and federated query processing systems in particular:

- Comparing the first run for a query against subsequent runs; to understand the effect of caching.
- Observing if performance degrades for large numbers of runs by comparison to smaller numbers of runs; to understand if there are memory leaks and other instabilities.
- Observing if performance degrades for large numbers of experiments executed concurrently; to perform stress-testing.
- Comparing the performance of the same federation engine, on the same datasets, over different data sources; to understand the effect of current load, implicit response size limits, allocated memory, and other specifics of the query processing engines that implement the data sources.
- Comparing the performance of different federation engines on the same experiment; to evaluate federation engines.

Based on the above, we will now proceed to define the requirements for a benchmarking system that supports automating the benchmarking process.

2.2.1 Data Source Provisioning

In order to reliably reproduce evaluation results, there are several parameters of the data source implementation that need to be controlled as they affect evaluation metrics. These include the software used to implement the SPARQL endpoint and its configuration, the memory, processing power, disk speed of the server where it executes, the quality of the network connection between the data server and the federation engine, etc.

Replicating a specific software stack and its configuration can be captured by virtualization and containerization technologies, so we require that a benchmarking engine use recipes (such as a Dockerfile for Docker containers) that prepare each endpoint's execution environment.

The characteristics of the computing infrastructure where the data service executes and of the network connection between the data service and the federation engine can be naturally aggregated as the latency and throughput at which the federation engine receives data from it. So, one requirement from benchmarking engines is that latency and throughput can be throttled to a maximum, although other conditions might make a data service even less responsive than these maxima: e.g., a data source might be processing an extremely demanding query or might be serving many clients in a stress test scenario.

Based on this observation, we require that benchmarking engines allow the experiment description to include the latency and throughput between the data sources and the federation. And, in fact, that these parameters are specific to each data source. Technically, this requires that the architecture foresees a configurable proxy between the federator and each data source, so that each experiment can set this parameter to simulate the real behaviour of SPARQL query processors.

Naturally, this is in addition to the obvious requirement to control the data served and the way that data is distributed between data services.

2.2.2 Sequential and Concurrent Application of Query Workload

The benchmarking engine should automate the process of applying a query load to the federation engine. The queries that make up the query load should be applied either sequentially to evaluate performance on different queries or concurrently to stress-test the system.

Technically, a benchmarking system should include an orchestrator that can read such operational parameters from the experiment definition and apply them when serving as a client application for the federation engine.

2.2.3 Logs Collection and Analysis

One important requirement of a benchmarking system is that the experimenter can have easy access on several statistics and *key performance indicators* of each conducted experiment. An effective presentation of such indicators can offer to the experimenter the ability to compare the performance of different setups of the same benchmark (e.g., different federators or data sources) and to draw conclusions for a specific setup by examining time measurements for each phase of the query processing and several other metrics.

Metrics that are important for the experimenter to analyze the effectiveness of a federator in a specific benchmark, include the following:

- The *number of returned results* can be used to validate the correctness of the query processing by verifying that the federator returns the expected number of results. Naturally, this validation is incomplete as the results might have the correct cardinality and still be different from the correct ones. However, many errors can be very efficiently caught by simply comparing cardinalities before proceeding to the detailed comparison.
- The *total time to receive the complete result set* indicates how the engine performs overall from the perspective of the client. This is the most common key indicator that most benchmarks consider.
- Although different federated query processing architectures have been proposed, there is some convergence on *source selection*, *query planning*, and *query execution* as beeing the main query processing phases. Regardless of whether these phases execute sequentially or are adaptive and their execution is interwined, the *breakdown of the query processing time into phases* provides the experimenter with insights regarding the efficiency of the federation engine and how it can be improved.
- The *number of sources accessed* during processing a specific query can be used to evaluate the effectiveness of source selection in terms of excluding redundant sources from the execution plan.

The aforementioned key performance indicators can be computed by different pieces of software during an experiment execution. For instance, the first two metrics of the above list should be computed by the evaluator (i.e., the software that poses the queries to the federator), while the last two metrics can be computed only by the federation engine itself. In order for these metrics to be available to the experimenter, the benchmarking system must collect and process the log lines emitted by the federation engine and the other components. This will produce an additional requirement on the compatible format of the log lines of the systems under test.

2.3 The KOBE System

The *KOBE Benchmarking Engine (KOBE)* is a system that aims to provide an extensible platform to facilitate benchmarking on federated query processing. It was designed with the following objectives in mind:

1. to ease the deployment of complex benchmarking experiments by automating the tedious tasks of initialization and execution;
2. to allow for benchmark and experiment specifications to be reproduced in different environments and be able to produce comparable and reliable results;
3. to provide to the experimenter the reporting that is identified by the requirements in Section 2.2.

In the following sections we will present the architecture and components of KOBE and its key features.

2.3.1 Deployment Automation

One of the major tasks that KOBE undertakes is the deployment, distribution and resource allocation of the various systems (i.e., the database systems, the federator and others) that participate

on a specific experiment. In order to achieve this task, KOBE employs Cloud-native technologies to facilitate the deployment on cloud infrastructures. Each system is deployed in an isolated environment with user-defined computational resources and network bandwidth. In particular, KOBE leverages containerization technologies to support the deployment of systems with different environments and installation requirements. An immediate consequence of employing those technologies is that KOBE is open and can be extended with arbitrary federators and database systems.

KOBE consists of three main subsystems that control three aspects of the benchmarking process:

- The *deployment subsystem* that is responsible for deploying and initializing the components required by an experiment. This subsystem handles the allocation of computational resources for each component.
- The *networking subsystem* that is responsible for connecting the different components of an experiment and imposes the throughput and latency limitations described by the benchmark.
- The *logging subsystem* that manages the logs produced by the several components (i.e, the data sources, federators and evaluators) and produces meaningful diagrams and graphs about the benchmarking process.

KOBE relies on Kubernetes[2] to allocate cluster resources for the benchmark execution. It deploys ephemeral containers with the individual components of a benchmarking experiment. The orchestration of that deployment and the communication with the underlying Kubernetes cluster is performed by the *KOBE operator*. The KOBE operator runs as a daemon and continuously monitors the progress of each running experiment in the cluster. This controller is also responsible for the interpretation of the experiment specifications (see Subsection 2.3.2) to complete deployment commands of the components of the experiment.

The network subsystem is controlled by Istio[3], a Cloud-native controller that tightly integrates with Kubernetes to provide a service mesh layer. The KOBE operator utilizes the functionality of Istio to setup the network connections between the data sources and the federating engine. The quality of those network connections can be controlled by the KOBE operator to provide the simulated behavior specified by the specific experiment. It is worth noting that those network links are established in the service mesh layer of the cluster and as a result one can have multiple experiments with different networking topologies running at the same time in the cluster.

The logging subsystem of KOBE is implemented as an EFK stack, a popular solution for a centralized, cluster-level logging environment in a Kubernetes cluster. EFK stack consists of (a) Elasticsearch[4], an object store where all logs are stored in a structured form, used for log searching, (b) Fluentd[5], a data collector which gathers logs from all containers in the cluster and feeds them into Elasticsearch, and (c) Kibana[6], a web UI for Elasticsearch, used for log visualization. Since the metrics of our interest are produced from the federator and the evaluator, and, as we will see in Section 2.4, these logs are of a specific form, Fluentd is configured to parse and to keep only the logs of these containers using a set of regular expression patterns for each type of KOBE-specific logs.

Figure 2.1 illustrates the relationships between the individual components and the information flow through this architecture. In a typical workflow, the user uses kobectl (the KOBE command-line client) to send commands to the KOBE operator. The operator, itself deployed as a container in the Kubernetes cluster, communicates with the Kubernetes API and with Istio in order to deploy

[2] cf. `https://kubernetes.io`
[3] cf. `https://istio.io`
[4] cf. `https://www.elastic.co/elasticsearch`
[5] cf. `https://www.fluentd.org`
[6] cf. `https://www.elastic.co/kibana`

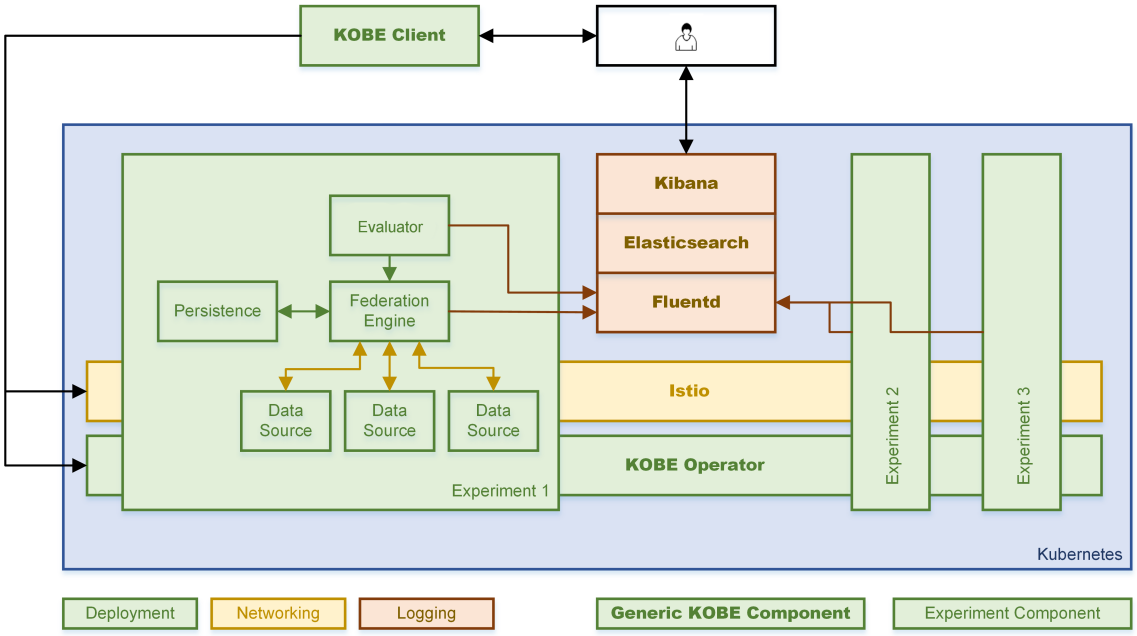

Figure 2.1: Information flow through a KOBE deployment: The user edits configuration files and uses kobectl (the KOBE command-line client) to deploy and execute the benchmarking experiments, at a level that abstracts away from Kubernetes specifics. Experimental results are automatically collected and visualized using the EFK stack.

the corresponding containers and establish the network between them. Moreover, a Fluentd logging agent is attached to each related container in order to collect the respective log output. The user also uses kobectl to provide a query load to the evaluator. The query evaluator is also deployed as a containerized application and is responsible for applying the query load to the federator and for measuring the latter's response.

During the execution of the experiment, Fluentd collects the log output from the evaluator, and parses it to extract evaluation metrics which are stored in Elasticsearch. If the federation engine is KOBE-aware, then it also produces log lines following the syntax understood by Fluentd so that fine-grained metrics about the different stages of the overall query processing are also computed and stored in Elasticsearch. The user connects to Kibana to see visualizations of these metrics, where we have prepared a variety of panels specifically relevant to benchmarking federated query processors.

2.3.2 Benchmark and Experiment Specifications

An important aspect of benchmarking is the ability to reproduce the experimental results of a benchmark. KOBE tackles this important issue by defining declarative specifications of the benchmarks and the experiments. Those descriptions can be serialized in a human-readable format (we use YAML as the markup language) and shared and distributed as artifacts.

These specifications are grouped around the various components of an experiment including the benchmark, the evaluator, the data source systems, the data federator and the network topology. Typically, those specifications are partitioned in a series of files; each file includes informations about different elements of the experiment. For example, one specification describes a specific federator and a different specification includes information about the set of datasets and querysets.

The main idea of this organization is that each specification can be provided by a different role. For example, the federator (resp. dataset server) specification should be provided by the *implementor of the federator* (resp. *dataset server*). These specifications include, for example, details about

the correct initialization of a federation engine. Moreover, the benchmark specification should be provided by the *benchmark designer* and the more specific details such as the computational resources and the network topology by the *experimenter*. The relevant pages of the online KOBE manual[7] give details about these parameters.

It is worth noting that the specifications are declarative in the sense that they describe the desired outcome rather than the actual steps one needs to follow to reproduce the experiment. The KOBE operator interprets these specifications as the necessary interactions with Kubernetes and Istio to deploy an experiment.

2.3.3 Experiment Orchestration

The KOBE operator is continuously monitoring for new experiment specifications that are submitted to KOBE by the user via a command-line client application. Upon a new experiment submission, the KOBE operator compiles new deployments for the data sources. The data sources consists of a list of dataset files, that is the serializable content of the dataset, and specifications about the database system that will serve this dataset. The deployment of a data source is performed in two phases: in the first phase the data files are downloaded and imported into the database system and in the second phase the system is configured and started for serving.

When all data sources are ready for serving, the federating engine is started. Similarly, the federating engine is deployed in two phases. In the first phase, the federation of the specific instances of data sources is established. This includes the specific initialization process that a federation engine might need. For example, some engines need the generation of a set of metadata that depend on the specific datasets that they federate. The second phase start the actual federation service. After that, the network connections are established and the network quality characteristics are configured.

In that stage the experiment is ready to proceed with querying the federation. This is accomplished by an evaluator component that reads the query set from the benchmark specification and starts sending the queries to the endpoint of the federator. The evaluator is just another container that is deployed in the cluster. During the query evaluation, potential logs that are produced by the federation engine and the evaluator are collected and visualized to the user. The experiment completes when the evaluator finished with all the queries.

2.4 Collecting and Analysing Evaluation Metrics

In Section 2.2.3 we stipulated that benchmarking engines should include a mechanism that collects and analyzes the logs from multiple containers in order to compute evaluation metrics, and to present them to the experimenter in an intuitive way.

2.4.1 Collecting the Evaluation Metrics

In KOBE, the following benchmarking metrics are treated: the duration of the query processing phases (source selection, planning, and execution); the number of sources accessed during a query evaluation from the federator; the total time to receive the complete result set of a query; and the number of the returned results of a query. We assume that the federator and the evaluator calculate these metrics and produce a corresponding log message for each metric.

[7] `https://semagrow.github.io/kobe/references/api`

Notice, though, that many executions of several experiments can result in multiple query evaluations. As a result, many log messages that contain the same metric can appear. In order to differentiate between these query evaluations and to collect all logs that refer to the same query that belongs to a specific run of an experiment, each log message should also provide the following information:

Experiment name: This information is used to identify in which experiment the given query evaluation belongs.

Start time of the experiment: Since one experiment can be executed several times, this information is used to link to the given query evaluation with a specific experiment execution.

Query name: Each query has a unique identification name in an experiment. This information is used to refer to the name of the query in the experiment.

Run: Each experiment has several runs, meaning that the evaluation of a query happens multiple times in a specific experiment execution. This information identifies in which run of the experiment the given query evaluation belongs.

An important problem that arises is that this information is only available to the evaluator and cannot be accessed by the federator directly. Any heuristic workarounds that try to connect the evaluator log to the federator log using, for instance, the query strings would not work, as query strings are not unique. Especially in stress-testing scenarios, the exact same query string might be simultaneously executed multiple times, so that a combination of query strings and timestamps would not be guaranteed to work either. To work around this problem, the KOBE evaluator uses SPARQL comments to pass the query *experiment* id to the federator, and the latter includes those in its logs. Then, the federator can retrieve this information by parsing this comment. This approach has the advantage that even if a federation engine has not been modified to produce log lines that provide this information, the query string is still in a valid, standard syntax and the comment is ignored. The fine-grained time to complete each step in the typical federated query processing pipeline cannot be retrieved, but the experiment can proceed with the end-to-end query processing measurements provided by the evaluator.

2.4.2 Visualizing the Evaluation Metrics

In this subsection, we describe the visualization component of KOBE. In particular, we present the three available dashboards. For every dashboard we provide some screenshots of the graphs produced for some experiment runs.

Details of a specific experiment execution

The dashboard of Figure 2.2 focuses on a specific experiment execution. It comprises:

1. Time of each phase of the query processing for each query of the experiment.
2. Total time to receive the complete result set for each query of the experiment.
3. Number of sources accessed for each query of the experiment.
4. Number of returned results for each query of the experiment.

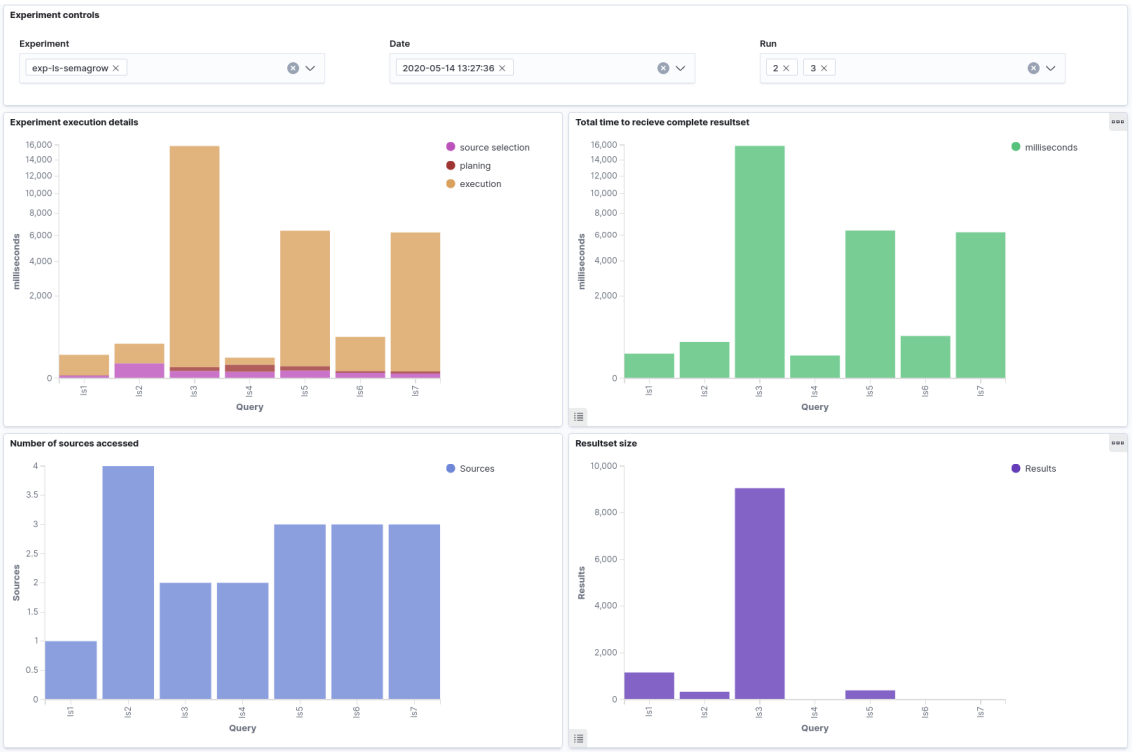

Figure 2.2: Details of a specific experiment execution

The first and the third visualizations are obtained from the logs of the federator engine, if available. The second and the fourth visualizations are obtained from the logs of the evaluator, so they are available even for federators that do not provide KOBE-specific logs. The values in each visualization can be also exported in a CSV file for further processing.

As an example, we consider an experiment execution for the life-science (ls) query set of the FedBench benchmark for a development version of the Semagrow federation engine. This visualization can help us, for instance, to observe that the query execution phase of the federation engine dominates the overall query processing time in all queries of the benchmark except ls4.

Comparisons of experiment runs

The dashboards depicted in Figure 2.3 and Figure 2.4 can be used to draw comparisons between several runs in order to directly compare different configurations of a benchmark. The dashboard of Figure 2.3 can be used for comparing several experiment executions. It consists of two visualizations:

1. Total time to receive the complete result set for each experiment execution.
2. Number of returned results for each specified experiment execution.

These visualizations are obtained from the logs of the evaluator. Each bar refers to a single query of the experiments presented. The dashboard of Figure 2.4 displays the same metrics. The main difference is that it focuses on a specific query and compare all runs of this query for several experiment executions. Contrary to the visualizations of the other two dashboards, each bar refers to a single experiment run, and all runs are grouped according to the experiment execution they belong to.

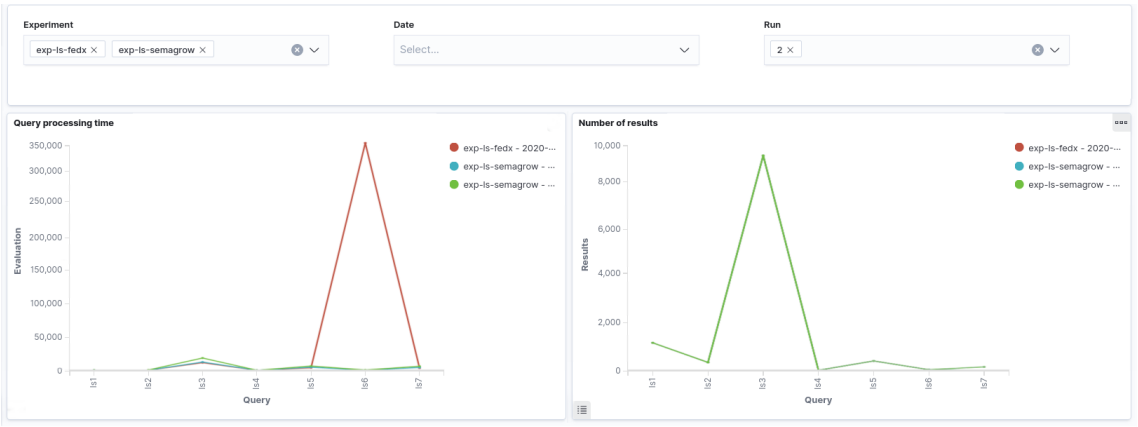

Figure 2.3: Comparison of three experiment executions

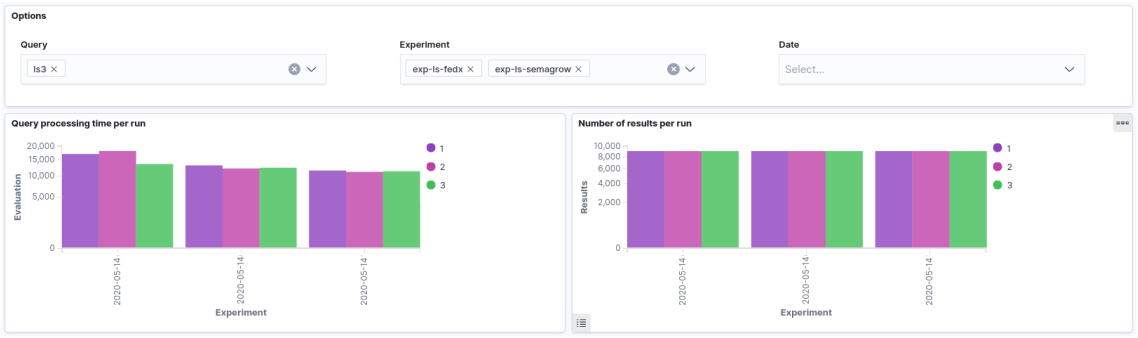

Figure 2.4: Comparison of all runs of the ls3 query for three experiment executions

Continuing the previous example, we consider three experiment executions that refer to for the life-science queryset of FedBench; one for the FedX federator and two for the Semagrow federator. In Figure 2.3 we can observe that all executions return the same number of results for each query, and that the processing times are similar, with the exception of the ls6 query for the FedX experiment. Moreover, we can observe that all runs return same number of results, and that the processing times for each run are similar; therefore any caching used by the federators does not play any significant role in speeding up this query.

2.5 KOBE Extensibility

It is apparent that a well-designed and well-executed benchmarking experiment needs contributions from different actors. For example, a benchmark designer may provide a benchmark that is designed to compare a particular aspect of different federators. On the other hand, the specifications of each federator should ideally be provided by their respective implementors.

KOBE provides various extensibility opportunities and by design welcomes contributions from the community. In particular, KOBE can be extended with respect to the database systems, federators, query evaluators and benchmarks that comprise an experiment.

We currently provide specifications for two database systems, namely for Virtuoso[8] and Strabo2[9] and for two federators, FedX [18] and Semagrow [3]. These systems have very different requirements

[8] cf. `https://virtuoso.openlinksw.com`
[9] cf. `http://strabon.di.uoa.gr`

in terms of deployment, providing strong evidence that extending the list of supported RDF stores will be straightforwrd.

We also provide a range of benchmark and experiment specifications for existing federated SPARQL benchmarks. Currently, the benchmarks that are already bundled with KOBE include the most widely used LUBM [6] and FedBench [17] benchmark. Moreover, we also include big RDF data benchmarks LargeRDFBench [16] and OPFBench [19] and geospatial benchmarks GeoFedBench [20], Geographica [4] and Geographica2 [7].

In the following, we briefly discuss the process of defining these specifications and give links to the more detailed walk-throughs provided in the online KOBE documentation.

2.5.1 Benchmarks and Experiments

Benchmarks are defined independently of the federator and comprise a set of datasets and a list of queries. Datasets are described in terms of the data and the system that should serve them. Data can be provided as a data dump to be imported in the database systems. For example, RDF data can be redistributed in the N-Triples format. Each dataset is characterized by its name and is parameterized by the URL where the data dump can be accessed. Queries of the benchmark are typically described as strings and annotated with the query language in which they are expressed; supporting heterogeneous benchmarks where not all data is served by SPARQL endpoints. A benchmark specification can also include network parameters, such as a fixed delay, or a percentage on which delay will be introduced as part of fault injection. The online KOBE manual provides walk-throughs for defining a new benchmark[10] and for tuning network parameters.[11]

An experiment that evaluates the performance of a federator over a given benchmark is defined using a strategy for applying the query load to the federator and the number of runs for each query of the experiment. The experimenter specifies an experiment by providing a new unique name for the experiment, the unique name of the benchmark and the federator specification. Moreover, an experiment includes a specific query evaluator, and the number of runs of the experiment. The query evaluator applies the query load to the federator. The one currently bundled with KOBE performs sequential querying, meaning that the queries of the benchmark are evaluated in a sequential manner. The online KOBE manual provides walk-throughs for defining a new experiment[12] and for extending KOBE with a new evaluator.[13] Furthermore, the manual also provides a walk-through for defining and visualizing new metrics.[14]

2.5.2 Dataset Servers and Federators

Dataset servers can be also integrated in KOBE. The dataset server specification contains a set of initialization scripts and a Docker image for the actual dataset server. The initialization scripts are also wrapped on isolated Docker containers and are used for properly initializing the database system. Typically, it includes the import of the data dump and indexing of the database. The dataset server specification may also include other parameters for network connectivity such as the port and the path to the listening SPARQL endpoint. A walk-through for adding a new dataset server is provided in the online KOBE manual.[15]

Federators can also be added to the KOBE system by providing the appropriate specification. That specification resembles the specification of a ordinary dataset server. The main difference is

[10] https://semagrow.github.io/kobe/use/create_benchmark
[11] https://semagrow.github.io/kobe/use/tune_network
[12] https://semagrow.github.io/kobe/use/create_experiment
[13] https://semagrow.github.io/kobe/extend/add_evaluator
[14] https://semagrow.github.io/kobe/extend/add_metrics
[15] https://semagrow.github.io/kobe/extend/add_dataset_server

on the initialization phase of a federator. Typically, the initialization of a federator may involve the creation of histograms from the underlying datasets. Thus, in KOBE, the federator initialization is performed in two steps: the first step extracts needed information from each dataset and the second step consolidates that information and properly initializes the federator. As in the dataset server, the initialization processes are provided as containerized Docker images by the implementor of the federator. A walk-through for adding a federator is provided in the online KOBE manual.[16]

Federator implementors should also consider a tighter integration in order to benefit from the detailed log collection features for reporting measurements that can only be extracted by collecting information internal to the federator (Section 2.4). Therefore, a log line from a federator should be enhanced to include the evaluation metrics and the query parameters discussed in Section 2.2.3. More details about how a federator should be extended to provide detailed logs are given in the online KOBE manual.[17] This tighter integration is not a requirement, in the sense that the overall end-to-end time to evaluate the query and the number of returned results are provided without modifying the source code of the federation engine (as we have done in the case of FedX).

2.6 Comparison to Related Systems

To the best of our knowledge, the only benchmark orchestrator that directly targets federated query processors is the orchestrator distributed with the FedBench suite [17]. As also stated in the introduction, it is in fact the limitations of the FedBench orchestrator that originally motivated the work described here. Specifically, FedBench does not support the user with either container-based deployment or collecting federator logs to compute detailed metrics.

HOBBIT [12], on the other hand, is a Docker-based system aiming at benchmarking the complete lifecycle of Linked Data generation and consumption. Although HOBBIT tooling can support with collecting logs and visualizing metrics, HOBBIT as a whole is not directly comparable to KOBE. In the HOBBIT architecture, the benchmarked system is perceived as an opaque container that the system tasks and measures. KOBE exploits the premise that the benchmarked system comprises multiple containers one of which (the federator) is tasked and that this one container communicates with the rest (the data sources). By exploiting these premises, KOBE goes further than HOBBIT could have gone to automate the deployment of the modules of an experiment and the control of their connectivity. In other words, KOBE aims at the federated query processing niche and trades off generality of purpose for increased support for its particular purpose.

A similar conclusion is also reached when comparing KOBE with scientific workflow orchestrators. Although (unlike HOBBIT and like KOBE) scientific workflow orchestrators are designed to orchestrate complex systems of containers, they focus on the results of the processing rather on benchmarking the processors. As such, they lack features such as controlling network latency.

Finally, another unique KOBE feature is the mechanism described in Section 2.4.1 for separating the logs of the different runs of an experiment. This especially useful in stress-testing scenarios where the same query is executed multiple times, so that the query string alone would not be sufficient to separate log lines of the different runs.

2.7 Conclusions

We have presented the architecture and implementation of the KOBE open benchmarking engine for federation systems. KOBE is both open-source software and an open architecture, leveraging

[16] `https://semagrow.github.io/kobe/extend/add_federator`

[17] Specifically, see the first step of the walk-through for adding a new federator. See also details about collecting logs to compute evaluation metrics `https://semagrow.github.io/kobe/extend/support_metrics`

containerization to allow the future inclusion of any federation engine. KOBE also uses Elasticsearch as a log server and Kibana as the visualization layer for presenting evaluation metrics extracted from these logs, again emphasizing openness by supporting user-defined ingestion patterns to allow flexibility in how evaluation metrics are to be extracted from each federator's log format. Deployment depends on Kubernetes, which is ubiquitous among the currently prevalent Cloud infrastructures. These features allow experiment publishers the flexibility needed for sharing federated query processing experiments that can be consistently reproduced with minimal effort by the experiment consumers.

Although originally developed for our own experiments, we feel that the federated querying community can extract great value from the abstractions it offers, as it allows releasing a benchmark as a complete, fully configured, automatically deployable testing environment.

As a next step, we are planning to expand the library of federators bundled with the KOBE distribution, and especially with systems that will verify that KOBE operates at the appropriate level of abstraction away from the specifics of particular federators. For instance, adding Triple Pattern Fragments [22] will verify that adaptive source selection and planning can operate within the KOBE framework.

Another interesting future extension would be support for the detailed evaluation of systems that stream results before the complete result set has been obtained. This requires adding support for calculating the relevant metrics, such as the diefficiency metric [1].

3. Linked Geospatial Data Benchmarks

In this chapter we presents the linked geospatial benchmarks that are contained in evaluation framework for big linked geospatial data systems, that was developed during Task T3.5. These benchmarks are Geographica2 (Section 3.1), Geographica3 (Section 3.2), and GeoFedBench (Section 3.3).

3.1 Geographica2

Geographica2 [1] [7] is the second iteration of the single node geospatial RDF store benchmark [4] Geographica[2] . It is more comprehensive, because it now includes new geospatial RDF stores and frameworks, big real world datasets of many hundred million triples with up to fifty million features of complex geometries, new tests and queries that reveal the scalability of these systems. The augmented and revised real world workload of Geographica2 tests the efficiency of primitive spatial functions in RDF stores, their performance in the geocoding scenario against the new Census dataset in addition to many other real use case scenarios and finally includes computation of statistics for geospatial datasets. A more detailed and systematic evaluation is performed using the synthetic workload. The new scalability workload aims at discovering the limits of centralized geospatial RDF stores of various architectures. It employs a set of six well balanced real world datasets with highly complex geometries covering many European countries and compares three RDF stores in terms of storage space, bulk loading and query response time. In addition, a special version of the benchmark has been created for systems with limited geospatial functionality and two more systems of this category are introduced along the six systems of the main benchmark, all stressed against point-only subsets of the workloads. Three out of the eight systems use an RDBMS for the persistence layer, while some of them offer a variety of persistence options. The benchmark is publicly available[3] as part of the KOBE benchmarking engine.

3.1.1 A benchmarking framework

Geographica2 is both a benchmark and a benchmarking framework. Apart from the workloads, datasets and querysets presented above, it also exposes two sets of APIs which allow the researcher to easily integrate and test new systems. The first runtime API is for RDF stores which are compliant with the OpenRDF Sesame API. The second runtime allows easy integration of systems that are compliant with the newer Eclipse RDF4J[4] API. Another important aspect of Geograhica2, as a framework, is the convenience it provides to modify existing workloads or creating new ones to meet the user's needs. There are class hierarchies of dataset and querysets which allow easy sub-classing to introduce with minimal coding effort the desired changes that meet one's needs.

3.1.2 Real world workload

The benchmark contains a real world workload that uses publicly available linked geospatial data, covering a wide range of geometry types (e.g., points, lines, polygons), some of which are highly complex. Each dataset is loaded in a different named graph and therefore allows for named graph query patterns. Figure 3.1 shows the comprised datasets with their basic characteristics such as data size, number of triples and the distribution of various geometry types they contain.

[1] http://geographica2.di.uoa.gr/
[2] http://geographica.di.uoa.gr/
[3] https://github.com/semagrow/benchmark-geographica
[4] https://rdf4j.org/

Datasets	Size	Triples	# of Points	# of Lines (max/min/avg # of points per line)	# of Polygons (max/min/avg # of points per polygon)
GAG	33MB	4K	-	-	325 (14K/4/192)
CLC	401MB	630K	-	-	45K (15K/4/171)
OSM (only ways)	29MB	150K	-	12K (1.6K/2/21)	-
GeoNames	45MB	400K	22K	-	-
DBpedia	89MB	430K	8K	-	-
Hotspots	90MB	450K	-	-	37K (4/4/4)
Census	3.3GB	23M	-	894K (262/2/6)	-

Figure 3.1: Real world dataset characteristics

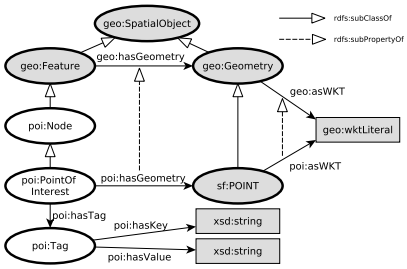

Figure 3.2: Ontology for Points of Interest

The real world workload follows the approach of the benchmark Jackpine [15] and defines both a micro and a macro benchmark. The micro benchmark tests primitive spatial functions. The spatial component of a system is tested with queries that use non-topological functions, distance, spatial selections, spatial joins and spatial aggregate functions. The macro benchmark tests performance of selected RDF stores in typical application scenarios like geocoding, reverse geocoding, map search and browsing and a real world use case from the EO domain. In these scenarios, a mix of spatial nearest neighbor, spatial selections and spatial joins are tested over multiple named graphs.

3.1.3 Synthetic workload

The benchmark also features a synthetic workload, which is based primarily on VESPA [14] and [10, 2]. It uses a generator that produces synthetic datasets of various sizes and generates queries of predefined thematic and spatial selectivity combinations. In this way, performance of geospatial RDF stores can be studied in a closely controlled environment.

3.1.4 Synthetic generator - StdSynthGen

The synthetic workload of Geographica2 relies on a generator that produces synthetic datasets of arbitrary size and instantiates query templates as queries with varying thematic and spatial selectivity. The synthetic generator `StdSynthGen` is a component of Geographica 2 and is distributed freely as open-source software.

The produced dataset comprises a collection of N-Triples files each one containing instances of features on a map. As in VESPA [14], they model the following geographic features: `country states`, `land ownership`, `roads` and `points of interest`. The features, follow a small ontology that takes after a general version of the schema of OSM and uses GeoSPARQL ontologies and vocabularies. In Figure 3.2 the developed ontology for representing points of interest is presented. As in [2, 10], every feature (i.e., point of interest) is assigned a number of thematic tags each of which consists of a key-value pair of strings. Each feature is tagged with key 1, every other feature with key 2, every fourth feature with key 4, etc. up to key $2^k, k \in \mathbb{N}$. This tagging makes it possible to select different parts of the entire dataset in a uniform way, and perform queries of

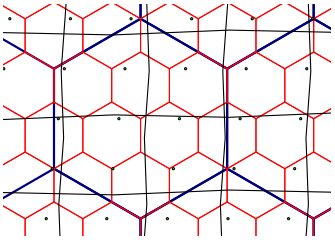

Figure 3.3: Visualization of the geometric part of the synthetic dataset

various thematic selectivities. For example, if we selected all points of interest tagged with key 1, we would retrieve all available points of interest (100% selectivity), if we selected all points of interest tagged with key 2, we would retrieve half of them (50% selectivity), etc.

Every feature has a spatial extent that is modelled using the GeoSPARQL vocabulary. The spatial extent of the land ownership dataset constitutes a uniform grid of $n \times n$ hexagons. The land ownership dataset forms the basis for the spatial extent of all generated datasets since the size of each dataset is given relatively to the number n (also mentioned as `N` from now on). In Figure 3.3 a sample of the spatial extent of all features is presented. By modifying the number of hexagons along an axis, datasets of arbitrary size can be produced. The same parameter is also used in order to create the corresponding queryset to run against the dataset.

By default, the synthetic generator produces the maximum (100%) and minimum (100/N%) thematic tags in order to keep the dataset size to reasonable sizes to be handled by single node geospatial RDF stores. The emphasis is given on testing mainly the spatial component of the systems under test and for this reason the produced queryset includes several instantiations of each query template with many difference spatial selectivities, each one combined with the min and max thematic selectivities (thematic tags) only.

The dataset produced for `N=512` is **778.5MB**. This is the value used for the synthetic workload of Geographica2 benchmark for single node systems. Every time the N doubles the dataset size quadruples. The dataset produced for `N=4096` is **49.2GB**. Since 4096 is 8*512, the resulting size is approximately 64 times the size of 512 dataset. The limitations of this generator are discussed in detail in section 3.2.2.

3.2 Geographica3 CL

3.2.1 General

In the context of ExtremeEarth project it was required to create a geospatial semantic benchmark that would be appropriate to test systems such as Strabo2, the distributed scalable GeoSPARQL query execution engine. The differentiating factors from the previously presented benchmark, Geographica2, are: (i) the datasets should scale well beyond the sizes of the single node benchmark, hundreds of TBs instead of hundreds of GBs, (ii) the deployment environment is HopsWorks, a contemporary multi-node HDFS-derivative cluster and not single node server and (iii) it is difficult to find real world RDF geospatial datasets that exceed the few TBs limit.

3.2.2 A distributed geospatial benchmark

The sizes of the geospatial semantic datasets required by ExtremeEarth project (hundreds of TB) for tasks 3-3 and 3-5 cannot be matched by real world datasets. A major issue with locating or

assembling real world geospatial datasets at this size is that they need to be well interlinked in order to make reasonable queries over them. For example, finding geospatial semantic datasets with overlapping spatial extents is relatively easy, since the desired spatial association in most cases is done at query time. However, in order to be able to run meaningful real world use case queries against this collection of datasets we need to have dense associations in their thematic aspects as well. Our best efforts produced OSM and Corine Land Cover based datasets of 1TB size. These efforts were actually based on the scalability workload of Geographica2 which was extended in order to go from 500 million triples to approximately 1.2 billion triples. The conclusion is that at the time of writing the desired dataset sizes can only be achieved by a synthetic generator.

Limitations of the Standard Synthetic Generator

The Geographica2 `StdSynthGen` was initially considered because it produces an ontology based dataset and the corresponding queryset, with a scaling factor `N` as a parameter. By appropriately increasing `N` we could, in theory, create an as large as desired geospatial N-Triples dataset. However when faced with the task of scaling to large values for `N`, a number of limitations of the specific implementation were revealed, which did not allow it to scale to the desired ExtremeEarth requirements. These limitations are described below:

- It is a single-threaded application with sequential processing logic and time complexity O(N^2). On an Intel i7-7700HQ CPU the N=4096 dataset (49.2GB) takes 20 minutes to complete and the N=8192 dataset (196.7GB) requires 80 minutes.
- It is also a memory intensive application with space/memory complexity approximately O(N^2). On a 24GB RAM node the N=8192 dataset (196.7GB) is the max scaling factor before a memory problem occurs. So for a N>=16384 dataset the memory requirement for a single node system would be approximately 128GB Ram which is unreasonable.
- The output file format is uncompressed text with storage requirements following the O(N^2) rule. A list of serious side-effects contains the following:
  - N=4096 dataset is made up of 5 files with a total of 50 GB size
  - datasets are created externally of the target cluster and need to be uploaded to the cluster file system
  - datasets this size require compression before uploading independent of the connection bandwidth
  - unreasonable expected upload times for N>=8192 datasets with high probability of corrupting the data
  - loss of project data on the cluster for whatever reason (i.e. platform upgrades) requires new uploads
  - large storage footprint punishes target cluster resources since production clusters use a replication factor >1, usually 3.
- Queryset spatial and thematic selectivities are fixed and require code modification to match specific requirements
- Dataset includes only the maximum and minimum thematic tags described in the ontology. As a consequence queries cannot use other tag values to achieve different thematic selectivities.
- Dataset and Queryset are produced by the same class which allows for less flexibility as far as dataset or queryset specific parameters.

These important limitations obliged us to create a new distributed Spark application that would remedy all of the above problems. The specifications and features of this distributed synthetic generator `DistSynthGen` are presented below.

3.2.3 Requirements of DistSynthGen

`DistSynthGen` follows the same ontology and logic as the standard synthetic generator `StdSynthGen` and produces similar datasets. However in view of the limitations of `StdSynthGen`, in combination with the WP3 demand for cluster platforms, we set the following requirements for the new distributed synthetic generator:

- The application needs to run in a cluster environment and employ a more parallel logic for generating datasets in order to achieve high horizontal scalability. While ExtremeEarth requires specifically the HopsWorks plarform, it is desirable to target standard Hadoop clusters as well in order to increase the software acceptance by the user community, which is also desirable by EE.
- Output datasets should have a more compact size and try to use more cluster friendly storage options in addition to plain text.
- Try to optimize memory data structures in order to minimize memory footprint and reduce network traffic between nodes.
- Datasets must include the additional triples that correspond to all tag values according to the ontology in order to allow for queries with various thematic selectivities to take effect.
- Queryset generation should be more parametric and allow the user to define the desired lists of thematic (tag values) and spatial selectivities.
- The output datasets should be validated against the corresponding datasets generated by `StdSynthGen`, in order to make sure that the produced datasets are valid.

3.2.4 Main features of DistSynthGen

The most efficient and appropriate distributed execution engine for Hadoop based clusters is Apache Spark and we used the Spark RDD API for most part of the logic implementation. The features of the new distributed synthetic generator are explained in detail in the following list:

- `DistSynthGen` was implemented as a Github Maven Java project[5] with 2 profiles that target the following platforms:
  - The standard Hadoop cluster is targeted by the **hdfs** profile, which is the `default` one and the specific releases of the main components are: Hadoop v2.10.0 and Apache Spark v2.4.5
  - The `HopsWorks` cluster platform is targeted by the **hops** profile and the specific releases of the main components are: Hadoop v3.2.0.2 and Apache Spark v2.4.3.2
- Optimized data structures to minimize memory footprint on the driver, each executor and network communication.
- To minimize the storage footprint the **Parquet+Snappy** file format was added to the output format options.
- To increase parallelism by dataset distributed consumers, such as Strabo 2 Loaders, the number of optimal partitions can be provided as a parameter in order to have a specific number of partitions for each one of the 5 dataset files or automatic partition calculation can be used by specifying a value of **0**.

[5] `https://github.com/tioannid/dist-semantic-geospatial-synthetic-generator`

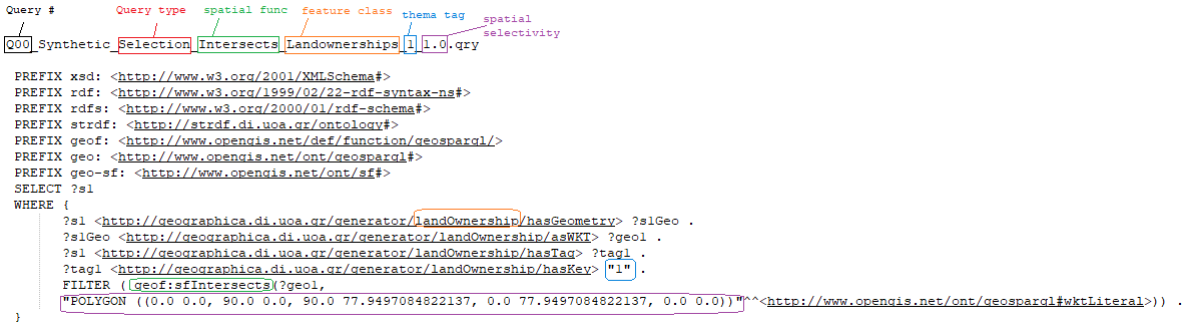

Figure 3.4: Spatial Selection query on Landownerships

- Dynamic Spatial Selectivities for Querysets is provided by the user and allows for better steering of query loads towards testing the scalability of spatial behaviour.
- Dynamic Thematic tag list (thematic selectivities) for Querysets is provided by the user and allows better control of the overall data engaged in heavy query joins.
- Dataset and Queryset are produced by different Java classes each one with its own set of parameters. This allows for more clarity and flexibility as far as specific parameters are concerned for each type of operation (dataset or queryset generation).
- Verified that produced datasets and querysets of `DistSynthGen` match the output datasets and querysets of the `StdSynthGen` for scaling factors (N=4,..,32, .., 256, 512, 1024, 2048, 4096).
- The latest release of the `DistSynthGen` is **2.4.5-SNAPSHOT**

3.2.5 DistSynthGen Queries explained

`DistSynthGen` uses 2 GeoSPARQL query templates: *(i) spatial selection* and *(ii) spatial join*. Each template is further customized by the choice of the spatial function, feature(s) involved, thematic and spatial selectivity. Each GeoSPARQL query file has a distinctive name which combines all the above information in order to make it as descriptive as possible.

Spatial Selection template

The spatial selection template has four different parameters `(sf, fc, ts, ss)`:

- Spatial function **(sf)** : sfIntersects or sfWithin
- Feature class **(fc)** used for spatial and thematic selection : Landownership, State, POIs (points of interest)
- Thematic tag/selectivity **(ts)** : 2^0=1,..,2^k=N, where N is the dataset scale factor
- Spatial selectivity **(ss)** : s <= 1, where 1 means 100%

Figure 3.4 shows query `Q00_Synthetic_Selection_Intersects_Landownerships_1_1.0.qry` which is an instantiation of the spatial selection template. This query requests all **Landownership** features which have a tag value **1** and spatially **intersect** with a bounding box that covers **1.0** = 100% of the ontology's map spatial extent. The template is instantiated with the following set of values:

```
(sf, fc, ts, ss) = (sf_Intersects, Landownerships, 1, 1.0)
```

Figure 3.5: Spatial Join query between Landownerships and States

Spatial Join template

The spatial join template has five different parameters `(sf, fc1, ts1, fc2, ts2)`:

- Spatial function **(sf)** : sfIntersects, sfTouches or sfWithin
- Feature class 1 **(fc1)** : first feature to participate in the spatial join
- Thematic tag/selectivity 1 **(ts1)** : 2^0=1,..,2^k=N, where N is the dataset scale factor
- Feature class 2 **(fc2)** : second feature to participate in the spatial join
- Thematic tag/selectivity 2 **(ts2)** : 2^0=1,..,2^k=N, where N is the dataset scale factor

Figure 3.5 shows query `Q10_Synthetic_Join_Intersects_Landownerships_States_1_2.qry` which is an instantiation of the spatial join template. This query requests all **Landownership** and **State** features which spatially **intersect**, with the additional thematic constraints that all (tag=1 means 100%) `Landownership` instances should be considered but only half (tag=2 means 50%) of the `State` instances. The template is instantiated with the following set of values:

```
(sf, fc1, ts1, fc2, ts2) = (sf_Intersects, Landownerships, 1, States, 2)
```

3.2.6 Usage of DistSynthGen

In the below sections we start by giving instructions on how to build a platform specific version of the software. We proceed with examples of dataset and queryset generation for both target cluster platforms and elaborate on the options provided to the user and the obtained results.

Platform specific build

To user initially needs to clone the DistSynthGen the github repository[6]. For the standard Hadoop cluster the user builds (from the local repo root directory) with maven using the **hdfs** profile or no profile at all, since **hdfs** is the default one:

```
$ mvn clean package -DskipTests [-Phdfs]
```

[6] `https://github.com/tioannid/dist-semantic-geospatial-synthetic-generator`

The generated uber-jar is named **SyntheticGenerator-2.4.5-SNAPSHOT_hdfs.jar**.

To build the DistSynthGen for the HopsWorks cluster the user builds (from the local repo root directory) with maven using the **hops** profile:

```
$ mvn clean package -DskipTests -Phops
```

The generated uber-jar **SyntheticGenerator-2.4.5-SNAPSHOT_hops.jar** needs to be uploaded through the `HopsWorks` UI into a project in order to be available for execution.

Dataset generation

The main class for the dataset generation is `generator.DistDataSyntheticGenerator` and its syntax is shown below:

```
DistDataSyntheticGenerator <FileFormat> <DstDir> <N> <P> {<ALL_THEMA>}
<FileFormat> : spark output file format {text | parquet}
<DstDir> : destination folder in HDFS
<N> : dataset scale factor, a value (preferably 2^k) which scales the size of the dataset
<P> : number of partitions, to be used for the generation of the 5 data files (0=automatic)
{<ALL_THEMA>} : default value=false, produce all thematic tag values
```

The following command uses the 'hdfs' jar to create a N=256 scaled dataset comprising 5 parquet snappy-compressed files each one in 1 partition with all thematic tag values present:

```
$ $SPARK_HOME/bin/spark-submit --class generator.DistDataSyntheticGenerator --master spark://localhost:7077 \
--conf spark.sql.parquet.compression.codec=snappy target/SyntheticGenerator-2.4.5-SNAPSHOT_hdfs.jar \
parquet hdfs://localhost:9000/user/tioannid/Resources/Synthetic/256/data/ 256 1 ALL_THEMA

$ hdfs dfs -ls Resources/Synthetic/256/data/*
Found 2 items
-rw-r--r--   1 tioannid supergroup          0 ./data/HEXAGON_LARGE/_SUCCESS
-rw-r--r--   1 tioannid supergroup    1150557  ./256/data/HEXAGON_LARGE/part-00000-7bd028db-93ad-4982-ace0-c8b95b02ac6f-c000.snappy.parquet
Found 2 items
-rw-r--r--   1 tioannid supergroup          0 ./data/HEXAGON_LARGE_CENTER/_SUCCESS
-rw-r--r--   1 tioannid supergroup     862458  ./256/data/HEXAGON_LARGE_CENTER/part-00000-14a7306c-c243-46ae-8ff3-e89a86f1f084-c000.snappy.parquet
Found 2 items
-rw-r--r--   1 tioannid supergroup          0 ./data/HEXAGON_SMALL/_SUCCESS
-rw-r--r--   1 tioannid supergroup   11826896  ./256/data/HEXAGON_SMALL/part-00000-be2d7fde-b7b2-4327-bb1e-689f93627082-c000.snappy.parquet
Found 2 items
-rw-r--r--   1 tioannid supergroup          0 ./data/LINESTRING/_SUCCESS
-rw-r--r--   1 tioannid supergroup     616819  ./256/data/LINESTRING/part-00000-aee735c3-3327-4bf3-93a0-5ffc12df1068-c000.snappy.parquet
Found 2 items
-rw-r--r--   1 tioannid supergroup          0 ./256/data/POINT/_SUCCESS
-rw-r--r--   1 tioannid supergroup    9444757  ./256/data/POINT/part-00000-4825de83-c3cf-46dd-8eab-d89fe8552b0a-c000.snappy.parquet
```

Figure 3.6 shows the HopsWorks user interface job arguments to create a N=4096 scaled dataset in the `hdfs:///Projects/DistSynthGen/Synthetic244_ALLTHEMA/4096/unitext/` location, comprising 5 text files each one in 1 partition with all thematic tag values present. Figure 3.7 shows the 1 partition file generated for the LandOwnership feature class which corresponds to `HEXAGON_SMALL` geometry.

Queryset generation

The main class for the queryset generation is `generator.DistQuerySyntheticGenerator` and its syntax is shown below:

```
DistQuerySyntheticGenerator <DstDir> <N> <S> <T>
<DstDir> : destination folder in HDFS
<N> : dataset scale factor, the queries will be used for the corresponding scaled dataset
<S> : selectivities list, eg. "1,0.5,0.1,0.01" for 100%, 50%, 10%, 1%, 0.1% selectivities
<T> : thematic tag list (comma separated within double-quotes of 2^i values <= N)
```

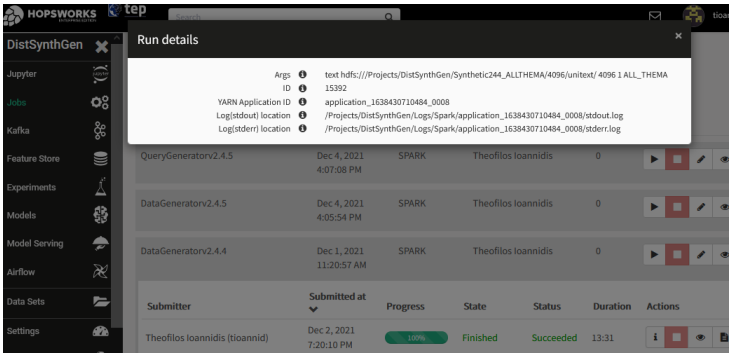

Figure 3.6: Dataset creation on the HopsWorks platform

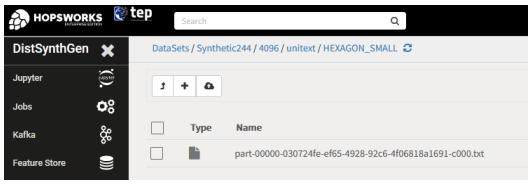

Figure 3.7: Single partition file for LandOwnerships (small hex) on HopsWorks

The following command uses the 'hdfs' jar to create a N=512 scaled queryset to be used with the corresponding N=512 scaled dataset, using spatial selectivities (100%, 10%, 1%) and thematic tag list "1,2,512" (1-> 100%, 2-> 50%, 512-> 1/512*100 = 0,19%):

```
$ $SPARK_HOME/bin/spark-submit --class generator.DistQuerySyntheticGenerator --master spark://localhost:7077 \
target/SyntheticGenerator-2.4.5-SNAPSHOT_hdfs.jar hdfs://localhost:9000/user/tioannid/Resources/Synthetic/512/qrytest/ \
512 "1,0.1,0.01" "1,2,512"

$ hdfs dfs -ls hdfs://localhost:9000/user/tioannid/Resources/Synthetic/512/qrytest
Found 72 items
-rw-r--r-- 1 tioannid supergroup  928 hdfs://localhost:9000/../qrytest/Q00_Synthetic_Selection_Intersects_Landownerships_1_1.0.qry
-rw-r--r-- 1 tioannid supergroup  928 hdfs://localhost:9000/../qrytest/Q01_Synthetic_Selection_Intersects_Landownerships_2_1.0.qry
-rw-r--r-- 1 tioannid supergroup  930 hdfs://localhost:9000/../qrytest/Q02_Synthetic_Selection_Intersects_Landownerships_512_1.0.qry
-rw-r--r-- 1 tioannid supergroup  960 hdfs://localhost:9000/../qrytest/Q03_Synthetic_Selection_Intersects_Landownerships_1_0.1.qry
-rw-r--r-- 1 tioannid supergroup  960 hdfs://localhost:9000/../qrytest/Q04_Synthetic_Selection_Intersects_Landownerships_2_0.1.qry
-rw-r--r-- 1 tioannid supergroup  962 hdfs://localhost:9000/../qrytest/Q05_Synthetic_Selection_Intersects_Landownerships_512_0.1.qry
-rw-r--r-- 1 tioannid supergroup  958 hdfs://localhost:9000/../qrytest/Q06_Synthetic_Selection_Intersects_Landownerships_1_0.01.qry
-rw-r--r-- 1 tioannid supergroup  958 hdfs://localhost:9000/../qrytest/Q07_Synthetic_Selection_Intersects_Landownerships_2_0.01.qry
-rw-r--r-- 1 tioannid supergroup  960 hdfs://localhost:9000/../qrytest/Q08_Synthetic_Selection_Intersects_Landownerships_512_0.01.qry
-rw-r--r-- 1 tioannid supergroup 1106 hdfs://localhost:9000/../qrytest/Q09_Synthetic_Join_Intersects_Landownerships_States_1_1.qry
-rw-r--r-- 1 tioannid supergroup 1106 hdfs://localhost:9000/../qrytest/Q10_Synthetic_Join_Intersects_Landownerships_States_1_2.qry
-rw-r--r-- 1 tioannid supergroup 1108 hdfs://localhost:9000/../qrytest/Q11_Synthetic_Join_Intersects_Landownerships_States_1_512.qry
-rw-r--r-- 1 tioannid supergroup 1106 hdfs://localhost:9000/../qrytest/Q12_Synthetic_Join_Intersects_States_Landownerships_1_1.qry
-rw-r--r-- 1 tioannid supergroup 1106 hdfs://localhost:9000/../qrytest/Q13_Synthetic_Join_Intersects_States_Landownerships_1_2.qry
-rw-r--r-- 1 tioannid supergroup 1108 hdfs://localhost:9000/../qrytest/Q14_Synthetic_Join_Intersects_States_Landownerships_1_512.qry
-rw-r--r-- 1 tioannid supergroup 1106 hdfs://localhost:9000/../qrytest/Q15_Synthetic_Join_Intersects_Landownerships_States_2_1.qry
-rw-r--r-- 1 tioannid supergroup 1106 hdfs://localhost:9000/../qrytest/Q16_Synthetic_Join_Intersects_Landownerships_States_2_2.qry
-rw-r--r-- 1 tioannid supergroup 1108 hdfs://localhost:9000/../qrytest/Q17_Synthetic_Join_Intersects_Landownerships_States_2_512.qry
-rw-r--r-- 1 tioannid supergroup 1106 hdfs://localhost:9000/../qrytest/Q18_Synthetic_Join_Intersects_States_Landownerships_2_1.qry
-rw-r--r-- 1 tioannid supergroup 1106 hdfs://localhost:9000/../qrytest/Q19_Synthetic_Join_Intersects_States_Landownerships_2_2.qry
-rw-r--r-- 1 tioannid supergroup 1108 hdfs://localhost:9000/../qrytest/Q20_Synthetic_Join_Intersects_States_Landownerships_2_512.qry
-rw-r--r-- 1 tioannid supergroup 1108 hdfs://localhost:9000/../qrytest/Q21_Synthetic_Join_Intersects_Landownerships_States_512_1.qry
-rw-r--r-- 1 tioannid supergroup 1108 hdfs://localhost:9000/../qrytest/Q22_Synthetic_Join_Intersects_Landownerships_States_512_2.qry
-rw-r--r-- 1 tioannid supergroup 1110 hdfs://localhost:9000/../qrytest/Q23_Synthetic_Join_Intersects_Landownerships_States_512_512.qry
-rw-r--r-- 1 tioannid supergroup 1108 hdfs://localhost:9000/../qrytest/Q24_Synthetic_Join_Intersects_States_Landownerships_512_1.qry
-rw-r--r-- 1 tioannid supergroup 1108 hdfs://localhost:9000/../qrytest/Q25_Synthetic_Join_Intersects_States_Landownerships_512_2.qry
-rw-r--r-- 1 tioannid supergroup 1110 hdfs://localhost:9000/../qrytest/Q26_Synthetic_Join_Intersects_States_Landownerships_512_512.qry
-rw-r--r-- 1 tioannid supergroup 1071 hdfs://localhost:9000/../qrytest/Q27_Synthetic_Join_Touches_States_States_1_1.qry
-rw-r--r-- 1 tioannid supergroup 1071 hdfs://localhost:9000/../qrytest/Q28_Synthetic_Join_Touches_States_States_1_2.qry
-rw-r--r-- 1 tioannid supergroup 1073 hdfs://localhost:9000/../qrytest/Q29_Synthetic_Join_Touches_States_States_1_512.qry
-rw-r--r-- 1 tioannid supergroup 1071 hdfs://localhost:9000/../qrytest/Q30_Synthetic_Join_Touches_States_States_1_1.qry
-rw-r--r-- 1 tioannid supergroup 1071 hdfs://localhost:9000/../qrytest/Q31_Synthetic_Join_Touches_States_States_1_2.qry
-rw-r--r-- 1 tioannid supergroup 1073 hdfs://localhost:9000/../qrytest/Q32_Synthetic_Join_Touches_States_States_1_512.qry
-rw-r--r-- 1 tioannid supergroup 1071 hdfs://localhost:9000/../qrytest/Q33_Synthetic_Join_Touches_States_States_2_1.qry
-rw-r--r-- 1 tioannid supergroup 1071 hdfs://localhost:9000/../qrytest/Q34_Synthetic_Join_Touches_States_States_2_2.qry
-rw-r--r-- 1 tioannid supergroup 1073 hdfs://localhost:9000/../qrytest/Q35_Synthetic_Join_Touches_States_States_2_512.qry
-rw-r--r-- 1 tioannid supergroup 1071 hdfs://localhost:9000/../qrytest/Q36_Synthetic_Join_Touches_States_States_2_1.qry
-rw-r--r-- 1 tioannid supergroup 1071 hdfs://localhost:9000/../qrytest/Q37_Synthetic_Join_Touches_States_States_2_2.qry
-rw-r--r-- 1 tioannid supergroup 1073 hdfs://localhost:9000/../qrytest/Q38_Synthetic_Join_Touches_States_States_2_512.qry
```

```
-rw-r--r-- 1 tioannid supergroup 1073 hdfs://localhost:9000/../qrytest/Q39_Synthetic_Join_Touches_States_States_512_1.qry
-rw-r--r-- 1 tioannid supergroup 1073 hdfs://localhost:9000/../qrytest/Q40_Synthetic_Join_Touches_States_States_512_2.qry
-rw-r--r-- 1 tioannid supergroup 1075 hdfs://localhost:9000/../qrytest/Q41_Synthetic_Join_Touches_States_States_512_512.qry
-rw-r--r-- 1 tioannid supergroup 1073 hdfs://localhost:9000/../qrytest/Q42_Synthetic_Join_Touches_States_States_512_1.qry
-rw-r--r-- 1 tioannid supergroup 1073 hdfs://localhost:9000/../qrytest/Q43_Synthetic_Join_Touches_States_States_512_2.qry
-rw-r--r-- 1 tioannid supergroup 1075 hdfs://localhost:9000/../qrytest/Q44_Synthetic_Join_Touches_States_States_512_512.qry
-rw-r--r-- 1 tioannid supergroup  908 hdfs://localhost:9000/../qrytest/Q45_Synthetic_Selection_Within_Pois_1_1.0.qry
-rw-r--r-- 1 tioannid supergroup  908 hdfs://localhost:9000/../qrytest/Q46_Synthetic_Selection_Within_Pois_2_1.0.qry
-rw-r--r-- 1 tioannid supergroup  910 hdfs://localhost:9000/../qrytest/Q47_Synthetic_Selection_Within_Pois_512_1.0.qry
-rw-r--r-- 1 tioannid supergroup  934 hdfs://localhost:9000/../qrytest/Q48_Synthetic_Selection_Within_Pois_1_0.1.qry
-rw-r--r-- 1 tioannid supergroup  934 hdfs://localhost:9000/../qrytest/Q49_Synthetic_Selection_Within_Pois_2_0.1.qry
-rw-r--r-- 1 tioannid supergroup  936 hdfs://localhost:9000/../qrytest/Q50_Synthetic_Selection_Within_Pois_512_0.1.qry
-rw-r--r-- 1 tioannid supergroup  936 hdfs://localhost:9000/../qrytest/Q51_Synthetic_Selection_Within_Pois_1_0.01.qry
-rw-r--r-- 1 tioannid supergroup  936 hdfs://localhost:9000/../qrytest/Q52_Synthetic_Selection_Within_Pois_2_0.01.qry
-rw-r--r-- 1 tioannid supergroup  938 hdfs://localhost:9000/../qrytest/Q53_Synthetic_Selection_Within_Pois_512_0.01.qry
-rw-r--r-- 1 tioannid supergroup 1110 hdfs://localhost:9000/../qrytest/Q54_Synthetic_Join_Within_Pois_States_1_1.qry
-rw-r--r-- 1 tioannid supergroup 1110 hdfs://localhost:9000/../qrytest/Q55_Synthetic_Join_Within_Pois_States_1_2.qry
-rw-r--r-- 1 tioannid supergroup 1112 hdfs://localhost:9000/../qrytest/Q56_Synthetic_Join_Within_Pois_States_1_512.qry
-rw-r--r-- 1 tioannid supergroup 1110 hdfs://localhost:9000/../qrytest/Q57_Synthetic_Join_Within_States_Pois_1_1.qry
-rw-r--r-- 1 tioannid supergroup 1110 hdfs://localhost:9000/../qrytest/Q58_Synthetic_Join_Within_States_Pois_1_2.qry
-rw-r--r-- 1 tioannid supergroup 1112 hdfs://localhost:9000/../qrytest/Q59_Synthetic_Join_Within_States_Pois_1_512.qry
-rw-r--r-- 1 tioannid supergroup 1110 hdfs://localhost:9000/../qrytest/Q60_Synthetic_Join_Within_Pois_States_2_1.qry
-rw-r--r-- 1 tioannid supergroup 1110 hdfs://localhost:9000/../qrytest/Q61_Synthetic_Join_Within_Pois_States_2_2.qry
-rw-r--r-- 1 tioannid supergroup 1112 hdfs://localhost:9000/../qrytest/Q62_Synthetic_Join_Within_Pois_States_2_512.qry
-rw-r--r-- 1 tioannid supergroup 1110 hdfs://localhost:9000/../qrytest/Q63_Synthetic_Join_Within_States_Pois_2_1.qry
-rw-r--r-- 1 tioannid supergroup 1110 hdfs://localhost:9000/../qrytest/Q64_Synthetic_Join_Within_States_Pois_2_2.qry
-rw-r--r-- 1 tioannid supergroup 1112 hdfs://localhost:9000/../qrytest/Q65_Synthetic_Join_Within_States_Pois_2_512.qry
-rw-r--r-- 1 tioannid supergroup 1112 hdfs://localhost:9000/../qrytest/Q66_Synthetic_Join_Within_Pois_States_512_1.qry
-rw-r--r-- 1 tioannid supergroup 1112 hdfs://localhost:9000/../qrytest/Q67_Synthetic_Join_Within_Pois_States_512_2.qry
-rw-r--r-- 1 tioannid supergroup 1114 hdfs://localhost:9000/../qrytest/Q68_Synthetic_Join_Within_Pois_States_512_512.qry
-rw-r--r-- 1 tioannid supergroup 1112 hdfs://localhost:9000/../qrytest/Q69_Synthetic_Join_Within_States_Pois_512_1.qry
-rw-r--r-- 1 tioannid supergroup 1112 hdfs://localhost:9000/../qrytest/Q70_Synthetic_Join_Within_States_Pois_512_2.qry
-rw-r--r-- 1 tioannid supergroup 1114 hdfs://localhost:9000/../qrytest/Q71_Synthetic_Join_Within_States_Pois_512_512.qry
```

The names of the generated queries include the query number, the spatial query type (join or selection), the spatial operator used, the feature(s) that participate, the thematic tag(s) and the spatial selectivity used.

3.2.7 Triples scaling

In this section, we present how the triples per dataset feature class scale with respect to the dataset scaling factor. Scaling factor N=512 is the `baseline` because this is the factor used in the Synthetic workload of Geographica2 benchmark. Scaling factor N=4096 is the threshold from which onward the `StdSynthGen` starts to raise problems on a single node server. Figure 3.8 shows that each time a scaling factor doubles, the number of triples approximately quadruple. The scaling factors up to N=32768 have been successfully created in the HopsWorks software platform on PolarTEP with a small hardware configuration (2 physical servers, max 10-12 executors, file replication=1). All results presented in this section have been generated with the default value of false for the ALL_THEMA parameter, and as a result only the minimum and maximum thematic tags are generated(for values 1 and scale factor respectively).

3.2.8 Storage/size scaling

Below, we present the storage scaling with respect to the dataset scaling factor. Figure 3.9 presents the storage size per feature file in `Text` format and the total storage required for one copy of the data. The ratio by which the total storage size increments from one scaling factor to the next is presented in red color and is approximately 4. Figure 3.10 presents the storage size per feature file in `Parquet+Snappy` format and the total storage required for one copy of the data. The ratio by which the total storage size increments from one scaling factor to the next is presented in red color and is approximately 4.

The `Parquet+Snappy` file format storage requirements are on average 13% of the `Text` file format requirements and therefore it is by far a more cluster-friendly choice. As we will see later, this benefit comes with a small time generation penalty.

TEXT / PARQUET	N=512 (baseline)	1024	2048	4096	8192	16384	32768
HEXAGON_LARGE (states)	202.524	814.403	3.256.764	13.044.367	52.173.916	208.764.915	835.045.124
HEXAGON_LARGE_ CENTER (state centers)	202.524	814.403	3.256.764	13.044.367	52.173.916	208.764.915	835.045.124
HEXAGON_SMALL (land ownerships)	1.837.056	7.344.128	29.368.320	117.456.896	469.794.816	1.879.113.728	7.516.323.840
LINESTRING (roads)	3.588	7.172	14.340	28.676	57.348	114.692	229.380
POINT (points of interest)	1.837.056	7.344.128	29.368.320	117.456.896	469.794.816	1.879.113.728	7.516.323.840
TOTAL	**4.082.748**	**16.324.234**	**65.264.508**	**261.031.202**	**1.043.994.812**	**4.175.871.978**	**16.702.967.308**

Figure 3.8: Scaling of Triples per Feature Class

TEXT	N=512 (baseline)	1024	2048	4096	8192	16384	32768
HEXAGON_LARGE (states)	37.3M	150.5M	604.8M	2.4G	9.5G	38.1G	153.1G
HEXAGON_LARGE_ CENTER (state centers)	33.9M	136.6M	549.0M	2.2G	8.6G	34.6G	139.1G
HEXAGON_SMALL (land ownerships)	374.7M	1.5G	5.9G	23.7G	94.6G	379.5G	1.5T
LINESTRING (roads)	6.4M	24.4M	95.4M	377.5M	1.5G	5.8G	23.4G
POINT (points of interest)	326.2M	1.3G	5.1G	20.6G	82.5G	331.0G	1.3T
TOTAL	**778.5M**	**3.0G** (x3.95)	**12.2G** (x4.06)	**49.2G** (x4.03)	**196.7G** (x4.00)	**789.1G** (x4.01)	**3.1T** (x4.02)

Figure 3.9: Text Storage Scaling

PARQUET	N=512 (baseline)	1024	2048	4096	8192	16384	32768
HEXAGON_LARGE (states)	4.7M	20.5M	86.5M	350.8M	1.4G	5.6G	23.0G
HEXAGON_LARGE_ CENTER (state centers)	3.4M	14.5M	59.4M	241.0M	978.8M	3.9G	15.8G
HEXAGON_SMALL (land ownerships)	46.9M	198.3M	835.0M	3.3G	13.3G	53.8G	218.8G
LINESTRING (roads)	2.3M	12.4M	60.1M	237.1M	920.9M	3.5G	14.3G
POINT (points of interest)	36.9M	149.6M	611.4M	2.4G	9.8G	39.6G	160.8G
TOTAL	**94.3M**	**395.2M** (x4.19)	**1.6G** (x4.14)	**6.5G** (x4.06)	**26.3G** (x4.04)	**106.4G** (x4.04)	**432.6G** (x4.06)

Figure 3.10: Parquet + Snappy compression Storage Scaling

Format	512	1024	2048	4096
Text	0m 36s	1m 0s (x1.66)	2m 09s (x2.15)	11m 00s (x5.11)
Parquet	0m 31s	0m 58s (x1.87)	3m 54s (x4.03)	14m 00s (x3.58)

Figure 3.11: PolarTEP Baseline Configuration: Driver(2GB, 1vCore), 1 x Executor(4GB, 1vCore)

3.2.9 Time/generation scaling

The time scalability of the `DistSynthGen` data generation is presented and analyzed in the below paragraphs. Both output file formats were tested. In order to check the time scalability we employed three different `HopsWorks` job configurations, by increasing the statically allocated hardware resources from the `PolarTEP` cluster. These are: *(i) Baseline*, *(ii) Medium*, and *(iii) High*.

Baseline configuration

The **Baseline** job configuration represents an as similar as possible cluster configuration to that of a single node system. For the `Driver` node it statically allocates 2GB of RAM and 1vCore. There is 1 `Executor` node which statically allocates 4GB of RAM and 1vCore. With this setup we intend to draw conclusions about the following aspects of `DistSynthGen` performance:

- Compare the `DistSynthGen` performance (Text format) against the `StdSynthGen` performance on the same scaling factors that `StdSynthGen` can produce on a single node server.
- Check the `DistSynthGen` time scalability for small scaling factors.
- Evaluate the time penalty of using `Parquet+Snappy` against the `Text` file format.

Figure 3.11 presents the `Spark Job Total Uptime` with the **Baseline** job configuration in order to complete the dataset generation for scaling factors N=512,..,4096.

In red color you see the execution time scaling factor as the dataset scale factor doubles. Below 4 is very good. For N=4096 it takes less time in any of the output formats than the 20min execution time of the `StdSynthGen` in a single node machine. For the same scaling factor, `Parquet+Snappy` increaces the time by 27%.

Medium configuration

The **Medium** job configuration scales in two directions. It basically raises the driver memory to 4GB and the number of executors to 4. This is a modest setup that does not affect even a small cluster such as PolarTEP. The scaling factor range under test is: `2048, 4096, 8192, 16384`. The two smaller values of this range intentionally overlap with the two largest scaling factors tested with the `Baseline` configuration and extends it with 2 extra steps which are the maximum the `StdSynthGen` could generate on a single node with ample resources.

Figure 3.12 presents the `Spark Job Total Uptime` with the **Medium** job configuration in order to complete the dataset generation for scaling factors N=2048,..,16384. In red color it is depicted that the execution time scaling factor remains below 4 as the dataset scale factor doubles. For the

Format	2048	4096	8192	16384
Text	1m 12s	3m 24s (x2.83)	13m 00s (x3.82)	51m 00s (x3.92)
Parquet	1m 24s	4m 06s (x2.92)	16m 00s *(15m 00s 12prt)* (x3.65)	66m 00s *(60m 00s 12prt)* (x4.00)

Figure 3.12: PolarTEP Medium Configuration: Driver(4GB, 1vCore), 4 x Executor(4GB, 1vCore)

Format	4096	8192	16384	32768
Text	2m 18s	7m 48s (x3.39)	30m 00s (x3.84)	2h 5m (x4.16)
Parquet	2m 48s	9m 18s (x3.32)	34m 00s (x3.65)	2h 18m (x4.05)

Figure 3.13: PolarTEP High Configuration: Driver(8GB, 2vCore), 8 x Executor(4GB, 1vCore)

N=4096 dataset we also notice that the time is reduced more than x3.2 compared to the `Baseline` configuration, and for the parquet dataset the reduction is even bigger, x3.4.

For the 2 larger datasets the time scaling factors are below or near x4 which means linear horizontal scalability or better. We also had tested the effect of specifying the *ideal*[7] number of partitions for each one of the generated files, using the Apache Spark suggestion of *2-3 times the number of total executor vCores*. In this case we used **3** times 4 vCores which results in 12 partitions. The specific tests were run for the parquet format only and yielded a 5-10% improvement in response time. This is a small improvement compared to the expected one. However an important thing to take into account is that the PolarTEP cluster has only one datanode (file replication factor 1) and it doesn't help in achieving high parallelization in disk I/O when the executors read or write partitions.

High configuration

The **High** job configuration doubles the `Medium` configuration's allocated resources. It raises the driver memory to 8GB, the number of driver vCores increases to 2 and the number of executors to 8. This is a decent setup for the PolarTEP cluster size, since it statically allocates approximately 43% of its resources. The scaling factor range under test is: `4096, 8192, 16384, 32768`. The three smaller values of this range intentionally overlap with the three largest scaling factors tested with the `Medium` configuration and extends by 1 extra step which is the maximum storage limit of the PolarTEP datanode.

Figure 3.13 presents the `Spark Job Total Uptime` with the **High** job configuration in order to complete the dataset generation for scaling factors N=4096,..,32768. In red color it is depicted that the execution time scaling factor remains below 4 (for most cases) as the dataset scale factor doubles. For the N=32768 dataset the `Text` format time scaling factor x4.16 exceeds the ideal horizontal scalability while the `Parquet+Snappy` format time scaling factor is approximately x4. We also notice that for the two largest datasets in `Text` format, the time is reduced more than x1.6 compared to the `Medium` configuration, and for the parquet dataset the reduction is even bigger, x1.76 which is close to the ideally expected improvement of x2, since we have doubled the allocated resources.

[7] `https://spark.apache.org/docs/2.4.8/tuning.html`

3.3 GeoFedBench

Performance benchmarks are invaluable for evaluating and comparing federated query processing systems, but it is hard to design benchmarks that are both realistic and informative about the systems being tested. In this section we present GeoFedBench [20], a benchmark that has been obtained from the domain of food security and uses GeoSPARQL constructs that challenge all phases of federated query processing. The benchmark is publicly available[8] as part of the KOBE benchmarking engine.

3.3.1 Introduction and Motivation

Performance benchmarks are invaluable for evaluating and comparing systems, but designing benchmarks is subject to considerations that are difficult to satisfy simultaneously. One potential tension is the creation of a realistic benchmark that accurately reflects how the benchmarked systems will behave in real-world use cases against the design of a benchmark that is informative with respect to system characteristics we know in advance that we need to test and measure.

Given the above, we are excited to present a benchmark that has been obtained from an actual, practical applications of linked geospatial data querying. The data and the queries derived from practical use cases in the context of ExtremeEarth; In particular:

1. linking land usage data with water availability data provided for the Food Security Use Case
2. linking land usage data with ground observations for the purpose of estimating crop type accuracy

Besides being extracted from a real workflow in the Earth Observation domain, the benchmark queries use GeoSPARQL constructs that challenge all phases of federated query processing, from source selection to query planing and execution. Besides a detailed presentation of the datasets and the queries (Subsection 3.3.2 and Subsection 3.3.3), we also present an analysis of the basic query characteristics of the benchmark and the challenges that presents to federated GeoSPARQL query engines (Subsection 3.3.4).

3.3.2 The GSSBench Suite

The first part of the benchmark is the *GSSBench suite*. In this suite we have 3 data layers that cover Austria (i.e., administrative, snow cover, land usage) and each layer is partitioned both thematically and geospatially, thus resulting in a federation with more than 25 datasets. Each dataset contains a single thematic layer and refers to a specific polygonal area. The queries are provided by a real-world use case, namely the combination of snow cover data with land usage data (these queries were compiled with the help of VISTA).

Use-case: Combining snow data with crop-type data for Food Security

Food security is one of the most challenging issues of this century, mainly due to population growth, increased food consumption, and climate change. The goal is to minimize the risks of yield loss while making sure not to damage the available resources. Of great importance is the study of irrigation, which requires reliable water resources in the area being farmed. Considering

[8] `https://github.com/semagrow/benchmark-geofedbench`

the fact that a large portion of the world's freshwater is linked to snowfall and snow storage, a promising way for providing an indication of water availability for irrigation is to study the snow cover areas in conjunction with the field boundaries and their crop-type information. The queries that are most relevant for this analysis are spatial within queries, spatial intersection queries, and within-distance queries: that is, retrieving the land parcels with a given crop type that are within, intersecting, or within a given maximum distance (without requiring the exact distance) from any snow-covered area. Moreover, sometimes we need to reduce our focus either on a smaller polygonal area with given coordinates, or on a specific administrative region.

Datasets

We use the following data sources:[9]

1. The Austrian Land Parcel Identification System (INVEKOS)[10], which contains the geo-locations of all crop parcels in Austria and the owners' self-declaration about the crops grown in each parcel.
2. A snow cover map[11], which contains thematic and geospatial snow data within Austria from February to April of 2018.
3. The Database of Global Administrative Areas (GADM) for Austria[12], which contains all administrative divisions of Austria up to Level-3.

We envisage that Austrian state governments publish crop data for their own area of responsibility; and a further (possibly different) entity publishes snow cover datasets for the same area. As the datasets described previously refer to the whole region of Austria, we partinioned them for the purposes of our benchmark to datasets that refer to smaller areas. Regarding the administrative and crop datasets, we partition them into smaller datasets according to the polygons of the states of Austria. For the snow cover dataset, we create two different partitions; one partition using a canonical geographical grid (which reflects a scenario where the snow cover data provider ignores administrative areas) and one partition that follows the administrative regions (which reflects a scenario where snow cover data are also published by the state governments). The polygons of the grids for the former partition are obtained by dividing the minimum bounding box of Austria into 8 parts in a 4×2 grid, and the polygons of the states of Austria (used for the partitioning of all three datasets) are obtained from the GADM dataset.

The datasets and the code that we use for partitioning the data is publicly available.[13] The partitioning of a given input dataset according to a set of boundaries is executed as follows: First, we populate each member of the partition with all features that their geometry intersects with the corresponding boundary. Second, we substitute each shape of each member of the partition with its intersection with the corresponding boundary. By doing this, for all features that their geometry intersects with more than one partitioning boundary, we split the original shape into several parts so that each part fits entirely in a single member of the partition. Finally, we modify the URIs of all resources so that all resources that appear in the same output dataset share a common prefix, which is unique among the prefixes of all datasets of the benchmark.

Table 3.1 illustrates the statistics for the datasets of the benchmark; in the table, we group the datasets by type and we display statistics about the sum of the group, as-well-as average and standard deviation for each dataset in the group (notice that the datasets are unequal in size due

[9] All datasets described here are publicly available from `http://rdf.iit.demokritos.gr/dumps/`
[10] `http://www.data.gv.at/katalog/dataset/f7691988-e57c-4ee9-bbd0-e361d3811641`
[11] `http://earthanalytics.eu/food-security-use-case.html`
[12] `https://gadm.org/maps/AUT.html`
[13] The code that we use for partitioning the data are publicly available in `https://github.com/semagrow/semagrow-geotools`.

Table 3.1: GSSBench suite: Dataset statistics.

datasets	data type	boundary	#datasets		#triples total	#triples geosp.	#triples them.	#prop.
gadm1-9	administrative divisions	state polygon	9	total	57 K	2 K	55 K	35
				avg	6 K	250	6 K	35
				stdev	4 K	200	4 K	0
crops1-9	crop types and field boundaries	state polygon	9	total	14 M	2 M	12 M	7
				avg	2 M	223 K	1 M	7
				stdev	1 M	179 K	1 M	0
snowS1-9	snow cover areas	state polygon	9	total	331 K	66 K	265 K	5
				avg	37 K	7 K	29 K	5
				stdev	27 K	5 K	21 K	0
snowG1-7	snow cover areas	4×2 grid	7	total	335 K	67 K	268 K	5
				average	48 K	9 K	38 K	5
				stdev	34 K	6 K	27 K	0

to the large standard deviation for each group). Apart from the statistics, boundary type for each dataset and the number of datasets per group. The number of the snowG datasets is 7 (and not 8, as expected) because the north-west part of the 4×2 grid does not contain any data due to the shape of Austria.

The datasets are organized into two possible federation setups for the three available data layers (i.e., administrative, crops, and snow). The first federation setup comprises 27 source endpoints, namely gadm1-9, crops1-9, and snowS1-9; the second one comprises 25 source endpoints, namely gadm1-9, crops1-9, and snowG1-7. In the first setup, we have three datasets for each Austrian state, i.e., all three data layers are split according to the same set of geographical boundaries, while in the second one, the snow cover is divided in a canonical geographical grid, thus we have two data layers that are (by nature) aligned on an uneven geographical split and one that is not aligned.

Queries

In Table 3.2, we summarize the queries of the benchmark. The query workload is produced by 7 query templates (Q1-7); each query template has a single parameter, which is either a WKT literal (Q1-3) or a Municipality name (Q4-7). We generate a set of 100 municipality names and a set of 100 WKT literals; this makes a total of 700 queries.

For the municipality names, we select 100 random municipalities from the GADM shapefile using the PostgreSQL `random()` function. We ignore the municipalities whose names contain characters not in the English alphabet in order to avoid possible string encoding conflicts. For the WKT literals, we create 100 random polygons; We first generate 100 random points within the border of Austria; then, we extend each point by a few meters in each direction, by using the PostGIS `ST_Expand()` function, in order to form rectangles covering approximately an area of 25 square kilometres each. We prune all polygons that are not completely within Austria and repeat the steps above until the random polygons reach 100.

For every query, we define its *administrative part* as the triple patterns that refer to administrative data (i.e., datasets gadm1-9), its *crop part* as the triple patterns that refer to crop data (i.e., datasets crops1-9), and its *snow part* as the triple patterns that refer to snow data (i.e., datasets snowS1-9 or snowG1-7). Q1 comprises only an administrative part, Q2 and Q3 comprise a snow

Table 3.2: GSSBench suite: Queries.

	Parameter	Query	#tp	#geo selec	#geo joins	Data layers	#r
Q1	Polygon in WKT	Municipalities intersecting a given polygon	6	1	0	gadm	3.7
Q2	Polygon in WKT	Snow-covered potato fields intersecting a given polygon	10	2	1	crops, snow	2.1
Q3	Polygon in WKT	Potato fields within 5 km from snow cover and intersecting a given polygon	10	2	1	crops, snow	15.6
Q4	Municipality name	Snow cover areas within 5 km from a given municipality	9	0	1	gadm, snow	12.5
Q5	Municipality name	Potato fields within a given municipality	9	0	1	gadm, crops	9.7
Q6	Municipality name	Snow-covered potato fields within a given municipality	14	0	3	gadm, crops, snow	0.5
Q7	Municipality name	Potato fields within 5 km from snow cover and within a given municipality	14	0	3	gadm, crops, snow	6.7

and a crop part, Q4 (resp. Q5) comprises an administrative and a snow (resp. crop) part, Q6 and Q7 contain all three types of parts. Q1-Q3, Q6-7 use the `geof:sfIntersects` function; Q3, Q4, and Q7 use the `geof:distance` function; and finally, Q5-7 use the `geof:sfWithin` function.

For each query template, we also illustrate the number of triple patterns of the query (#tp), the number of geospatial selection filters, i.e., geospatial filters with one free variable (#geoselec), the number of geospatial join filters, i.e., geospatial filters with two free variables (#geojoins), and the relevant data layers for each query template. Notice that the queries that are parameterized with a WKT have geospatial selection filters for each data layer of the query, while the remaning queries do not have geospatial selection filters. Moreover, all queries that make use of two or three data layers (i.e., all queries apart from Q1) have geospatial join filters for combining data from the corresponding layers.

3.3.3 The GDOBench Suite

The second part of the benchmark is the *GDOBench suite*. The benchmark federates a database of Earth Observation data about land usage and a database of ground observations about land usage, to search for pairs between them that simultaneously satisfy geospatial and thematic (land usage) constraints (these queries were compiled with the help of UNITN).

Use-case: Validating land usage data using ground observations

Detailed land usage data is crucial in many applications, ranging from formulating agricultural policy and monitoring its execution, to conducting research on climate change resilience and future food security. Land usage can be inferred from Earth Observation images or collected through self-declaration, but in either case it is important for such data to be validated against land

Table 3.3: GDOBench suite: Dataset statistics.

dataset	#shapes	#triples total	geospatial	thematic	#properties
INVEKOS	2,008,137	14,056,959	4,016,274	10,040,685	7
LUCAS	4,325	30,379	8,650	21,729	11

surveys. Ground observations are geo-referenced to a point on the road adjacent to a field (and not inside a field), which is often ambiguous in agricultural areas with several adjacent parcels; further exacerbated by GPS accuracy. Therefore, we can estimate the error rate of the land usage data as follows: first, all ground observation is irrelevant to the analysis if it is more than 10 meters from any crop parcel. For the remaining ground observations, we find the nearest land parcel, and if the crop types match, the GPS point provides a positive validation; otherwise it provides a negative one. This process is challenging not only because it is computationally demanding (since it involves quadratic many distance calculations), but because the crop types between different datasets usually make use of different code names.

Datasets

We use the following data sources:

1. The Austrian Land Parcel Identification System (INVEKOS)[14], which contains the geo-locations of all crop parcels in Austria and the owners' self-declaration about the crops grown in each parcel.
2. The EUROSTAT's Land Use and Cover Area frame Survey (LUCAS)[15], which contains agro-environmental and soil data by field observation of geographically referenced points.

The datasets are publicly available as shapefiles, and are transformed into RDF with the GeoTriples tool [11]. The LUCAS dataset was extended with a set of mappings [13] between LUCAS land cover codes and INVEKOS crop types. Table 3.3 gives more details about these datasets. We illustrate the number of shapes that appear the original shapefiles, as-well-as the number of triples, entities, and properties that appear in the transformed RDF datasets.

Unlike GSSBench suite, here the datasets are not further partitioned.

Queries

In Table 3.4 we summarize the queries of the benchmark. Each row of the table corresponds with a query template; the parameter is shown in the "parameter" column of the table.

Q1-3 are derived from the data validation use case, and are used to estimate the reliability of the INVEKOS dataset. For each LUCAS instance URI, we check if: (Q1) the closest INVEKOS instance is under 10 meters away and their crop labels match (positive validation); (Q2) the closest INVEKOS instance is under 10 meters away and their crop labels do not match (negative validation); or (Q3) there is no INVEKOS instance within 10 meters (neutral validation). Instead of simply providing a boolean result, we formulated the queries as SELECT queries, so that the data analyst can get more information regarding the corresponding instances. Thus, each query either returns a single result (if the LUCAS instance has the desired property) or an empty result

[14] cf. `http://www.data.gv.at/katalog/dataset/f7691988-e57c-4ee9-bbd0-e361d3811641`
[15] cf. `https://esdac.jrc.ec.europa.eu/projects/lucas`

Table 3.4: GDOBench suite: Queries.

	parameter	query	#tp	characteristics
Q1	LUCAS URI	return the nearest INVEKOS instance if it is within 10 meters and their crop types match	10	Subquery, ORDER, LIMIT 1
Q2	LUCAS URI	return the nearest INVEKOS instance if it is within 10 meters and their crop types do not match	10	Subquery, ORDER, LIMIT 1, FILTER NOT EXISTS
Q3	LUCAS URI	return the LUCAS instance if there is no INVEKOS instance within 10 meters	10	Subquery, ORDER, LIMIT 1, FILTER NOT EXISTS
Q4	distance	return all INVEKOS instances within D meters from a LUCAS instance	5	-

set (otherwise). Notice that these queries have several characteristics apart from federated within-distance geospatial joins.

In order to focus on the behavior of the within-distance geospatial join, we have also included Q4. This query returns all INVEKOS instances that are within a given distance from a specific LUCAS instance. The query is parameterized with various distances ranging from 10 meters to 100 kilometers. As for the LUCAS instance, we used the one that is closest to the center of the minimum bounding box of Austria.

3.3.4 Benchmark characteristics

Since linked geospatial data vocabularies link instances with a geometry object (which then has as an attribute the actual shape), these queries (and GeoSPARQL queries in general) challenge `FILTER` optimizers because it presents them with comparisons between variable groundings (as opposed to constant values), and because these comparisons are non-standard extensions (the geospatial extensions of GeoSPARQL).

In most benchmarks, filters are either not present at all [6] or only have unary functions or comparisons against constants [17] that can always be pushed into one data source. LargeRDFBench [16] includes multi-variable filters that compare values from different repositories, challenging the optimizer to select the correct strategy: to fetch the left-hand side values and push the filter into the right-hand side source or to fetch both sides and apply the filter locally. Both approaches are valid, but can vary dramatically in terms of efficiency. Our benchmark presents the same challenge in a geospatial context; the federator is tested not only on correctly selecting the best strategy but also on the efficiency of its local implementation of the GeoSPARQL extension.

Properties of standard vocabularies, which can appear possibly in all sources of a federation, present another challenge in the efficient evaluation of a query. When evaluating a triple pattern that contains a property such as `rdf:type` or `owl:sameAs` the source selector is prone to overestimate the set of relevant sources, thus increasing both network traffic and the overall query processing time. Current benchmarks already contain such commonly used properties. But GeoFedBench stresses source selections more on this direction by exploiting a query characteristic that appears frequently

in Geospatial data; a resource `?x` is linked with its geometry representation `?wkt` using *chains of known properties* of the form `?x geo:hasGeometry ?g . ?g geo:asWKT ?wkt`, where all members of the chain usually appear in the same dataset. The federation engine is tested on distinguishing which geospatial triple patterns refer to which dataset, thus avoiding to fetch redundant bindings for the variable in the middle of the chain.

GeoFedBench's GSSBench suite contains many endpoints (27 or 25, depending on which federation setup is used), which is much higher than other benchmarks in the literature (e.g., FedBench [17] uses 9 and LargeRDFBench [16] uses 14). This challenges the source selector of a federator w.r.t. the total number of endpoints of the federation. In addition, notice that each of the endpoints refer to a specific polygonal area; to the best of our knowledge no federated linked geospatial data benchmark uses such a partitioning. A federation engine that is aware of the boundaries of the sources it federates should perform better on these queries, we notice that our benchmark tests whether a federator can exploit the geospatial nature of the source endpoints.

Finally, the complex nature of the queries of GeoFedBench's GDOBench suite challenges query planning. Current benchmarks usually contain simple queries consisting only joins between triple patterns and `FILTER` operations, or some additional operators such as `UNION`, `ORDER`, `LIMIT`, etc. In GDOBench, Q1 and Q2 use a *subquery* for discovering the *closest* INVEKOS instance. Also, Q2 and Q3 use *negation*, in the form of the `FILTER NOT EXISTS` operator to check that there does not exist and matching Invekos instance. Both subqueries and negation are not present in any of the currently existing federated SPARQL benchmarks.

4. Strabo2 Experiments

In this chapter we present the experimental evaluation of Strabo2 on the Hopsworks cluster that has been set up in CREODIAS. Strabo2 has been presented in Deliverable D3.7. Specifically we have presented the overall architecture of the system and the query translation process. We have also presented several improvements in the two different modules of the Strabo2 (loader and query executor). These improvements include caching of the thematic tables, creation of a persistent spatial index and partitioning through hybrid translation into both SQL and Java code, and using JedAI spatial to precompute qualitative spatial relations in order to avoid computations that involve geometries during query execution. Finally, in D3.7 we include a description of the query optimization, the SPARQL endpoint and a set of preliminary experiments.

In this chapter we extend the experiments presented in D3.7 with two main objectives. First, evaluate the system as a whole, including the ability to scale to a PB of initial geospatial RDF data. Second, evaluate several aspects of the system, and specifically the impact of the improvements that have been presented in D3.7.

4.1 Query Execution Results

In this section we present the query execution times for the synthetic dataset of the Geographica 3 CL generator to generate a dataset with scale factor 16384 and with the ALL_THEMA set to true, in order to generate all thematic tags. We have also generated 72 queries with spatial selectivities of 1%, 0.1% and 0.01% and thematic selectivities corresponding to values 4096, 8192 and 16384. The execution times for all 72 queries are shown in Table 4.1. We have used 22 executors with 6120 MB of memory and 2 virtual cores per executor. The size of the dataset is 156 GB in compressed parquet format, which corresponds to an initial size of 1.16 TB in NTriples text format. Total execution time for the 72 queries is 7711 seconds, which gives an average execution time of 107 seconds per query.

Query	Execution Time (ms)	Spatial Operator
Q01	232044	Selection
Q02	146899	Selection
Q03	147170	Selection
Q04	152111	Selection
Q05	146036	Selection
Q06	147468	Selection
Q07	164569	Selection
Q08	141737	Selection
Q09	137134	Selection
Q10	145127	Join
Q11	134914	Join
Q12	144041	Join
Q13	130845	Join
Q14	134303	Join
Q15	142198	Join
Q16	134373	Join
Q17	133919	Join
Q18	140007	Join
Q19	142181	Join
Q20	145099	Join
Q21	137109	Join
Q22	135898	Join

Q23	133025	Join
Q24	142054	Join
Q25	142111	Join
Q26	137723	Join
Q27	134736	Join
Q28	21336	Join
Q29	33408	Join
Q30	33380	Join
Q31	33085	Join
Q32	34876	Join
Q33	33604	Join
Q34	32637	Join
Q35	32478	Join
Q36	32597	Join
Q37	33155	Join
Q38	32862	Join
Q39	34451	Join
Q40	34375	Join
Q41	31425	Join
Q42	32942	Join
Q43	36023	Join
Q44	35866	Join
Q45	36219	Join
Q46	179207	Selection
Q47	148420	Selection
Q48	144277	Selection
Q49	138284	Selection
Q50	144529	Selection
Q51	146904	Selection
Q52	129080	Selection
Q53	143656	Selection
Q54	126967	Selection
Q55	110107	Join
Q56	105515	Join
Q57	101931	Join
Q58	107568	Join
Q59	103107	Join
Q60	104190	Join
Q61	103204	Join
Q62	103652	Join
Q63	102132	Join
Q64	102929	Join
Q65	109412	Join
Q66	101333	Join
Q67	101782	Join
Q68	103916	Join
Q69	108852	Join
Q70	106961	Join
Q71	122739	Join
Q72	111236	Join

Table 4.1: Query Execution Times for Synthetic Dataset Scale 16384

Scalability Experiments In order to provide a complete picture regarding the evaluation of Strabo2, in this paragraph we describe the scalability experiments that have also been reported in Deliverable D3.7. We have executed experiments with a varying number of worker nodes, and also

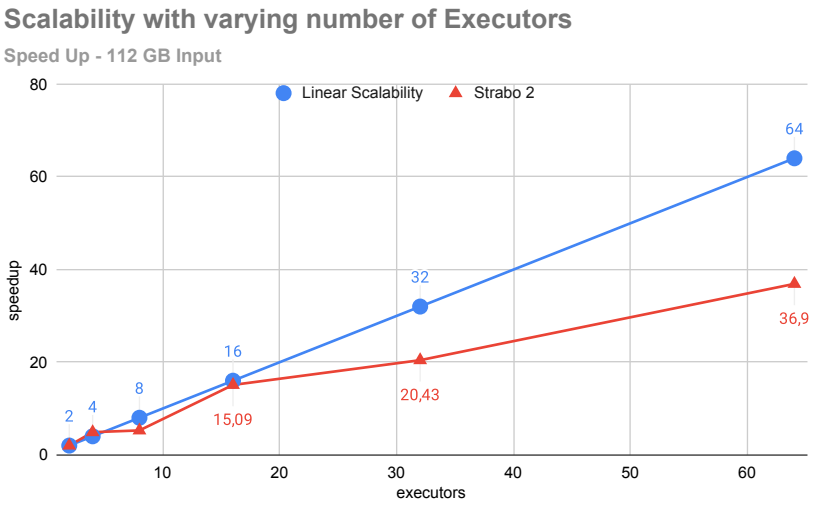

Figure 4.1: Execution with Varying Number of Executors

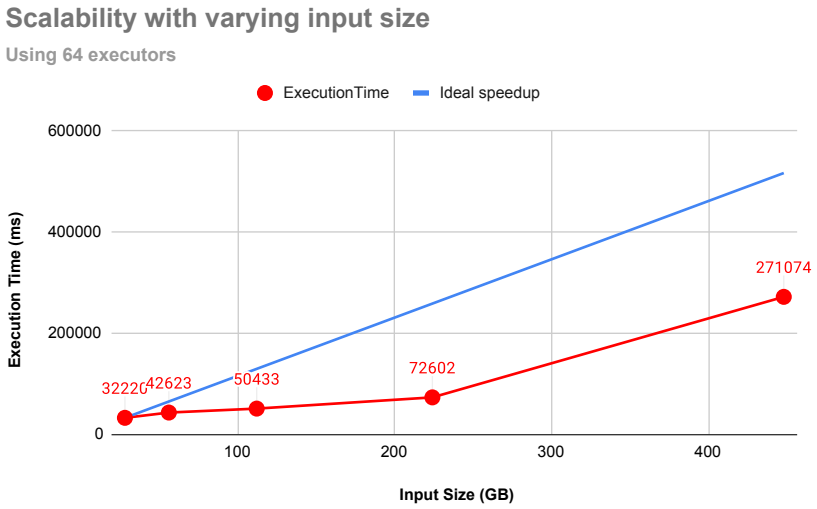

Figure 4.2: Execution with Varying Size of Input Dataset

with a varying dataset size. The results of the first set of experiments are shown in Figure 4.1, where we have executed 36 queries of the Geographica1 synthetic benchmark in an input dataset of about 112 GB in NTRIPLES files, with 2, 4, 8, 16, 32 and 64 executors, while the results of the second set of experiments are shown in Figure 4.2, where we are using 64 executors in order to execute the 36 queries in datasets of increasing size, starting from 10 GB up to 450 GB.

The two scalability experiments exhibit very good behavior for both spatial selection and spatial join queries. These experiments had been performed in a cluster operated by Hopsworks, able to provide almost 1000 virtual cores, whereas in the latest experiments on CREODIAS we are currently able to use up to 53 virtual cores and 165 GB of memory, with a single node acting as a datanode. Despite the limited capabilities of the CREODIAS cluster, we were able to handle a dataset with input Ntriples size of more than 1 TB. This, combined with the scalability results, leads us to anticipate that Stabo2 is able to handle much larger input datasets, given that more computing resources are available.

4.2 Evaluating Improvements in Query Execution

In order to evaluate the specific aspects and improvements in query execution we have used the Geographica 3 CL generator to generate a dataset with scale factor 1024 and queries for spatial selectivities of 1%, 0.1% and 0.01% and thematic selectivities corresponding to values 256, 512 and 1024. As before, we set the ALL_THEMA parameter of the generator to true, in order to generate all thematic tags. In total we have generated 72 queries. The experiments of this section have been executed with 4 worker nodes, each one with 2 virtual cores and 4096 MB of memory. All the necessary material in order to reproduce the experiments, includng the exact queries and instructions to generate the datasets, are available at the github repository of ExtremeEarth [1].

4.2.1 Caching of Thematic Tables

In this set of experiments, we modify the query plan produced by Strabo2 when Spark is going to execute a distributed sort-merge join between two thematic tables, in any kind of RDF join (subject to subject, subject to object or object to object joins). In a distributed sort-merge join, the two tables are hash partitioned on the join key using the same hash method, so records from the first table that match records of the second are placed in the same node. After the hash partitioning, each partition of each table is sorted, and then a merge join is performed locally in every node.

During query translation in Strabo2, we identify distributed merge joins, and without overhead in query evaluation, we cache the intermediate result of sorted and partitioned tables on the join keys. In subsequent queries we can reuse these cached tables in case of a distributed merge join in the same key. In order to evaluate the effect of thematic caching we have executed the 72 generated queries sequentially in two sets of execution runs, with the thematic cache enabled and disabled respectively. In the execution run where this feature is enabled, we start with an empty thematic cache, that is populated gradually with the execution of each query. The total execution time for the 72 queries of the benchmark was *407* and *340* seconds respectively, leading to an improvement of *16%* in total execution time.

4.2.2 Hybrid Translation with Persistent Spatial Index and Partitioning

In this setting, during startup we create a spatial index for all geometry tables that exist in the dataset, and then, during query execution we use the Sedona RDD API in order to access the spatial index and transform the results back to a dataframe, as described in Deliverable D3.7. This is only applicable when we want to directly access the original geometry table, and not an intermediate result that contains geometries. In the latter case the only available option is to create an index on the fly. According to the execution plan of Strabo2, in the case of a query that contains both thematic and spatial joins, the thematic filters and joins are executed first, as this tends to create smaller results. As a result, this optimization is taking place only in the case of spatial selections. For the 18 queries that contain a spatial selection, the total execution time when using the persistent spatial index and partitioning, drops from 111 seconds to 54 seconds, leading to a reduction in execution time of more than 50%.

The exact execution times are shown in Table 4.2. The query execution times in milliseconds are presented for the default translation (second column) and for the hybrid translation that uses the permanent spatial index and partitioning (third column). We also present the spatial and thematic selectivity of each query in fourth and fifth columns respectively.

[1] https://github.com/ExtremeEarth-Project/Strabo2-Experiments

Query	Default Translation (ms)	Hybrid Translation (ms)	Spatial Selectivity	Thematic Selectivity
S01	9148	5793	1	256
S02	6616	3808	1	512
S03	6420	2893	1	1024
S04	6840	2912	0.1	256
S05	6540	3136	0.1	512
S06	6428	2674	0.1	1024
S07	6070	2840	0.01	256
S08	5954	2750	0.01	512
S09	5862	2995	0.01	1024
S10	6199	2982	1	256
S11	5872	2591	1	512
S12	5143	2645	1	1024
S13	5736	2837	0.1	256
S14	6196	2554	0.1	512
S15	5541	2822	0.1	1024
S16	5711	2469	0.01	256
S17	5223	2506	0.01	512
S18	5705	2700	0.01	1024
Total	111204	53907		

Table 4.2: Effect of Hybrid Translation in Query Execution

4.2.3 Caching Qualitative Spatial Relations Using JedAI-Spatial

In this set of experiments we have used JedAI-spatial to precompute all spatial relations between geometries in the dataset, by setting the input dataset as both the source and target dataset in JedAI configuration. We used the "export as RDF" functionality of JedAI-spatial to export the qualitative spatial relations as triples using the corresponding qualitative spatial predicates of GeoSPARQL. Then, these triples were imported in Strabo2 using the default schema of vertical partitioning. The process of computing and exporting the spatial relations took 12 minutes. Regarding query execution, we observed that for the query set used in our experiments, using the produced tables instead of computing the spatial relations during query execution, did not provide gains. Instead, in many cases it led to slower query execution plans. On a closer inspection, the reason for this behavior is that Strabo2 performs thematic spatial and filters before the spatial join. As a result, in the default query execution mode, we limit the size of both input of the spatial join operator. On the contrary, when we are using the tables that have been created for the qualitative cache, the whole dataset is accessed for the subsequent distributed sort-merge join. As a result, the benefit from avoiding the spatial joins between small inputs is not preferable. In order to further examine this behavior, we have also tested Strabo2 with queries that access the whole dataset (thematic selectivity 1). In this set of queries, indeed the use of qualitative spatial index outperforms the default behaviour. We present the results with respect to the number of results of all the queries used in the experiment in Figure 4.3. For queries with less than 100 results, the use of qualitative cache results in worst query execution times. For queries with 100-1000 results, the two execution modes exhibit similar performance, whereas for queries that access large portion of the data with many thousands results, the use of qualitative cache outperforms the default execution mode.

4.3 Datasets and Queries from the Use Cases of ExtremeEarth

From the use cases of ExtremeEarth we have imported in Strabo2 the following datasets:

- For polar use case:
  - Ice Charts dataset
  - Ship positions dataset
  - Global Administrative Areas (GADM) for Norway and the Northern area

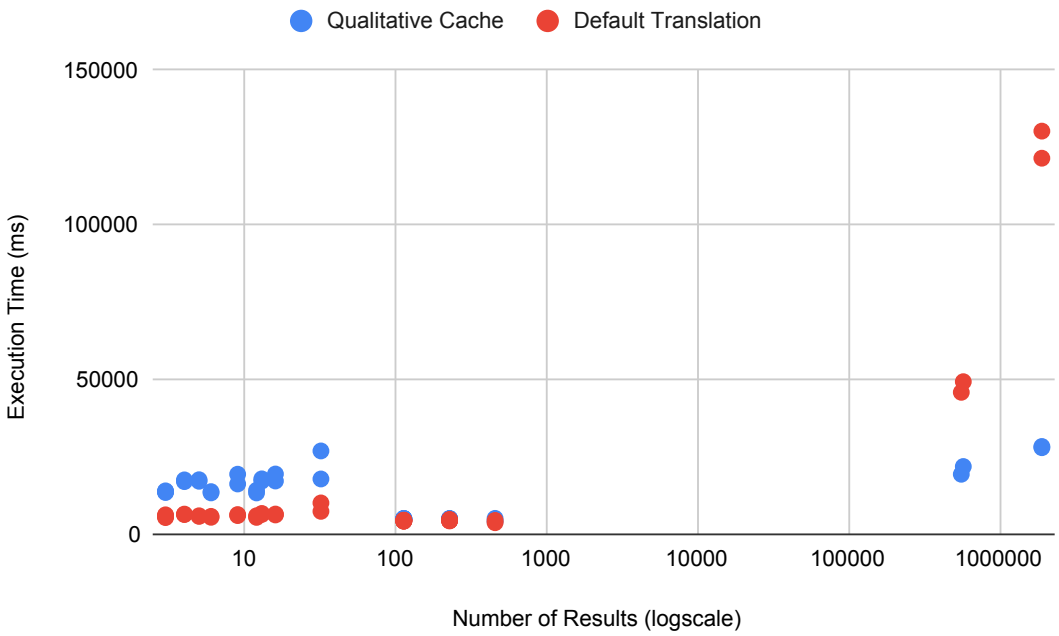

Figure 4.3: Effect of Qualitative Cache

- For food security use case:
  - Crop type maps
  - Hydro River Network EU (Danube, Rhine, Elbe)
  - Irrigation dataset
  - Precipitation dataset
  - Snow cover dataset
  - Interlinking result between Precipitation and GADM as produced by JedAI-Spatial

The total size of the NTriples text files is 30GB. Data loading using 8 executors, with 2 virtual cores and 4GB memory per executor took 80 minutes. Regarding query execution, we have tested the Strabo2 installation with several real world queries from the two use cases. Below we present two queries for the polar use case and two queries for the food security use case. For each query we present the natural language requirement, the GeoSPAQL formulation, the number of results and the query execution time using 12 executors with 2 virtual cores and 4GB memory per executor.

Query 1 (Polar)

Get all images that correspond to ice map observations that were obtained between 2018-03-03 and 2018-03-01 and the observation CT class name is Close Drift Ice

233 million results in 2 minutes

```
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX polar: <http://earthanalytics.eu/polar/ontology/>

SELECT ?imgTitle ?imgURL ?imgTN
WHERE {
?x polar:hasRECDAT ?date .
?x polar:hasCT ?ct .
```

```
?x rdf:type polar:IceObservation .
?x polar:hasCTClassName ?ctName .
?x geo:hasGeometry ?geo1 .
?geo1 geo:asWKT ?wkt1 .

?img rdf:type polar:SatelliteImage .
?img polar:hasURL ?imgURL .
?img polar:hasTitle ?imgTitle .
?img polar:hasThumbnail ?imgTN .
?img geo:hasGeometry ?imgGeo .
?imgGeo geo:asWKT ?imgWKT .

?x polar:belongsToImage ?img .

FILTER ( ?ct = "79"^^<http://www.w3.org/2001/XMLSchema#string>)

FILTER (?date < "2018-03-03T00:00:00"^^<http://www.w3.org/2001/XMLSchema#dateTime>
&& ?date > "2018-03-01T00:00:00"^^<http://www.w3.org/2001/XMLSchema#dateTime>)
}
```

Query 2 (Polar)

Get all observations in less than 5km distance from POLYGON ((0.0 0.0, 90.0 0.0, 90.0 77.94970848221368, 0.0 77.94970848221368, 0.0 0.0))

35.000 results in 47 seconds

```
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX polar: <http://earthanalytics.eu/polar/ontology/>
PREFIX uom: <http://www.opengis.net/def/uom/OGC/1.0/>

SELECT  ?wkt1 ?ctName ?date
WHERE {
?x rdf:type polar:IceObservation .
?x polar:hasCT ?ct .
?x polar:hasCTClassName ?ctName .
?x polar:hasRECDAT ?date .
?x geo:hasGeometry ?geo1.
?geo1 geo:asWKT ?wkt1 .

BIND(geof:distance(?wkt1,
"POLYGON ((0.0 0.0, 90.0 0.0, 90.0 77.94970848221368, 0.0 77.94970848221368, 0.0 0.0))"^^geo:wktLiteral, uom:metre)
as ?dist).

FILTER(?dist < 5000).
}
```

Query 3 (Food Security)

Get Regions affected by precipitation in Quarter 2 of 2021 that was lower than 15% of the normal rainfall and that are equipped with irrigation

12.000 results in 38 seconds

```
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX fso: <http://ai.di.uoa.gr/fs/ontology/>

SELECT ?pwkt ?ra ?cname ?cwkt
WHERE {
?fsobservation rdf:type fso:FoodSecurityObservation.
?fsobservation fso:hasStartDate "2021-10-01T00:00:00"
```

```
^^<http://www.w3.org/2001/XMLSchema#dateTime>.
?p fso:belongsTo ?fsobservation.
?p fso:hasPrecipitation ?prec.
?prec fso:hasRelativeAmount ?ra.
?prec geo:hasGeometry ?pgeo.
?pgeo geo:asWKT ?pwkt.
?ir fso:hasCapability ?cap.
?cap rdf:type fso:Capability.
?cap fso:hasClassName ?cname.
?cap geo:hasGeometry ?cgeo.
?cgeo geo:asWKT ?cwkt.

FILTER (geof:sfIntersects(?pwkt, ?cwkt))
FILTER (?ra <-15)
}
```

Query 4 (Food Security)

Get regions that showed a negative trend in precipitation in Q3 of 2020 but a positive in Q3 in 2019

64 results in 23 seconds

```
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX fso: <http://ai.di.uoa.gr/fs/ontology/>

SELECT ?pwkt ?ra1 ?ra2
WHERE {
?fsobservation1 rdf:type fso:FoodSecurityObservation.
?fsobservation1 fso:hasStartDate "2020-10-01T00:00:00"^^<http://www.w3.org/2001/XMLSchema#dateTime>.
?p1 fso:belongsTo ?fsobservation1.
?p1 fso:hasPrecipitation ?prec1.
?prec1 fso:hasRelativeAmount ?ra1.
?prec1 geo:hasGeometry ?pgeo1.
?pgeo1 geo:asWKT ?pwkt.
?fsobservation2 rdf:type fso:FoodSecurityObservation.
?fsobservation2 fso:hasStartDate "2019-10-01T00:00:00"^^<http://www.w3.org/2001/XMLSchema#dateTime>.
?p2 fso:belongsTo ?fsobservation2.
?p2 fso:hasPrecipitation ?prec2.
?prec2 fso:hasRelativeAmount ?ra2.
?prec2 geo:hasGeometry ?pgeo2.
?pgeo2 geo:asWKT ?pwkt2.

FILTER (?ra1 < 0)
FILTER (?ra2 > 0)
FILTER(geof:sfEquals(?pwkt, ?pwkt2))
}
```

5. Semagrow Experiments

In this chapter we present the final evaluation of Semagrow, developed in Task T3.4, using the KOBE Benchmarking Engine and the GeoFedBench benchmark, presented in the previous chapters.

5.1 The Semagrow query federation engine

Semagrow is a GeoSPARQL query federation engine. Semagrow provides unified access of linked geospatial data from multiple, possibly heterogeneous, geospatial data servers. The new version of Semagrow, was developed during Task T3.4 and presented in Deliverable D3.8 [21].

Our work in ExtremeEarth made Semagrow the first engine for federating big geospatial data sources and performing extreme geospatial analytics. All phases of the federated query processing (namely source selection, query planning and query execution) of Semagrow were extended, and Semagrow has been successfully integrated within Hopsworks. In addition, during Task T3.4 we introduced two novel techniques for federating geospatial linked data; a geospatial source selector and a federated geospatial join optimization, which have been both evaluated on data and queries from the use cases of the ExtremeEarth project; both experiments show that each of the techniques can substantially improve query processing time.

5.2 Evaluation using GSSBench suite of GeoFedBench

In the first part of the evaluation of Semagrow, we are using the GSSBench sute of GeoFedBench. This part of the evaluation extends Section 3.4 of D3.8 (where we evaluated the novel geospatial selector of Semagrow) in the following aspects: first, it contains more endpoints (34 instead of 28 previously), more federation setups (one federation setup that uses approximated shapes in the dataset metadata), and more queries (a workload of 700 queries instead if 8 previously); and second, it tests the final version of Semagrow instead of only the geospatial source selection component.

5.2.1 Experimental setup

Each dataset is deployed in a separate GeoSPARQL endpoint. We use the Strabon geospatial RDF store [10] for serving the data. Strabon encapsulates PostGIS for performing spatial operations, and uses a spatial index to optimize query processing time.

For each federation setup (i.e., the 27-dataset setup and the 25-dataset setup), we set up 4 Semagrow federators, each with a different source selection configuration. We illustrate the all the information about the federations used in the experiment in Table 5.1. For each federation, we display the source selection method, the number of federated endpoints, details and statistics about the Semagrow metadata used (namely, type of bounding polygon used, metadata size and number of coordinates that appear all WKT literals of the `svd:boundingWKT` property), and which datasets from Table 3.1 used in the federation.

The federators of thm-27 and thm-25 use only thematic metadata (i.e., no geospatial summaries for the source selector), thus the geospatial source selector is bypassed, while the remaining federators use not only thematic but geospatial metadata as well, thus the geospatial source selection is used. The difference between the remaining federators is on the accuracy of the bounding polygons that

Table 5.1: Federations used in the evaluation using the GSSBench suite.

	thm-27	geo-mbb-27	geo-appr-27	geo-poly-27
Source selector	thematic	geospatial	geospatial	geospatial
#datasets	27	27	27	27
Bounding WKT	-	Min.Bounding Box	Approx. shape	Exact shape
Meta-#triples	3024	3051	3051	3051
#coords	-	108	1998	68736
data file size	125 KB	132 KB	189 KB	1.8 MB
Datasets	gadm1-9 crops1-9 snowS1-9	gadm1-9 crops1-9 snowS1-9	gadm1-9 crops1-9 snowS1-9	gadm1-9 crops1-9 snowS1-9
	thm-25	**geo-mbb-25**	**geo-appr-25**	**geo-poly-25**
Source selector	thematic	geospatial	geospatial	geospatial
#datasets	25	25	25	25
Bounding WKT	-	Min.Bounding Box	Approx. shape	Exact shape
Meta-#triples	2941	2959	2959	2959
#coords	-	100	1360	45852
data file size	123 KB	127 KB	165 KB	1.3 MB
Datasets	gadm1-9 crops1-9 snowG1-7	gadm1-9 crops1-9 snowG1-7	gadm1-9 crops1-9 snowG1-7	gadm1-9 crops1-9 snowG1-7

the sources are tagged with; in geo-poly each source is tagged with the exact polygon that refers to the corresponding areas (i.e., the geographical grid for snowG datasets and the borders of Austrian states for the remaining ones); in geo-appr with an approximation of the above polygons a quadtree of height 2; in geo-mbb with the minimum bounding box of all shapes that appear in the source. All metadata were created using the Sevod Scraper tool[1]. The bounding polygons for geo-mbb-25, geo-mbb-27, geo-appr-25, and geo-appr-27 were calculated by the tool using the minimum bounding box and quadtree-based approximation, while for geo-poly-25 and geo-poly-27 we used the exact polygons of the states of Austria and the 4×2 grid according to the actual borders of the partitioned datasets. Notice that an increased accuracy leads to an increased metadata size (i.e., even though that the geospatial source selectors use metadata that have the same set of triples, the WKT literals contain a larger set of coordinates).

Notice that all federations (namely thm-27, geo-mbb-27, geo-appr-27, geo-poly-27, thm-25, geo-mbb-25, geo-appr-25, geo-poly-25 all refer to the new version of Semagrow, they only differ w.r.t. the configuration. We do not include pre-ExtremeEarth version of Semagrow in the evaluation results, because the execution plan for these queries is very inefficient since it contains all sources of the federation. Thus, the execution phase produces poor results and timeout errors.

We use a Kubernetes 1.14 cluster with 1 master node and 8 worker nodes with a total if 120 cores and 264GB RAM. Experiment deployment and execution is done through the KOBE benchmarking engine, and the KOBE configurations for reproducing the experiments are publicly available.[2].

[1] `https://github.com/semagrow/sevod-scraper`

[2] The experiment specifications can be found in `https://github.com/semagrow/benchmark-geofedbench` as a part of the `geofedbench` suite of KOBE.

5.2.2 Experimental results

In the following, we present the experimental results. We first focus on each phase of federated query processing separately, and then we discuss total query processing as a whole.

Evaluation metrics

All queries are decomposed successfully and for every query a correct execution plan is produced. However, in some queries the query execution phase evokes an error, and in these situations the federator returns no answer.

The experimental results are summarized in Tables 5.2, 5.3, 5.4, 5.5, 5.6, and 5.7. For each query template of Table 3.2 and for each federation of Table 5.1, we display the following evaluation metrics: to evaluate the efficiency of the query execution plan we display the average *query execution time* of all successful queries (cf. Table 5.5) and the *error rate* (cf. Table 5.6), i.e., the number of the unsuccessful queries over the number of all queries in the template; to check the efficiency of the other parts of query processing we display the average *source selection time* (cf. Table 5.2) and *planning time* (cf. Table 5.4); to check if the source selection time overheads are recovered by reduced planning and execution time, we display the time differences between each geospatial federation with its corresponding thematic one (cf. Table 5.7); to evaluate the efficiency of the pruning of each source selection, we display the average *number of sources* that are accessed during the evaluation (cf. Table 5.3); and finally, to check if any source selector achieves optimal pruning, we include the average *size of the optimal source set* (cf. opt-27 and opt-25 columns of Table 5.3).

Regarding the time measurements shown in Tables 5.2, 5.4, 5.5, and 5.7, apart from the average value, we include its standard derivation (displayed in parentheses). Moreover, regarding the average number of sources in Table 5.3, we include in parentheses the minimum and maximum number of sources.

Comparison of source selection times

In the following, we focus on the time overheads of the geospatial source selector. Thus, we will compare the federations of the experiment according to source selection time (cf. Table 5.2).

We observe that the source selectors of thm-27 and thm-25 (in short thm) are the fastest ones; then we have geo-mbb-27 and geo-mbb-25 (in short geo-mbb); then we have geo-appr-27 and geo-appr-25 (in short geo-appr); and finally we have geo-poly-27 and geo-poly-25 (in short geo-poly). This happens due to two main reasons. First, in thm the geospatial selector is bypassed, while in the remaining (i.e., geo-mbb, geo-appr and geo-poly) is not, which explains why thm is the fastest of all. Second, the sources in geo-poly are annotated with polygons, which are more complex shapes than the approximated shapes in geo-appr, which are, in turn, more complex shapes than the bounding boxes in geo-mbb. Thus, the boundary comparisons performed by the geospatial source selection are slower in geo-poly. This difference is more pronounced in Q3 and Q7, which include three geospatial filters and a within-distance operation (using the `geof:distance` function), which is computationally costlier than containment and intersection operations.

We observe that, in general, the source selection process is faster in the federations with 25 endpoints (e.g., compare the source selection time for the queries for geo-poly-25 with geo-poly-27). This happens not only because in the 27-dataset setup we have two additional endpoints, but mainly beacuse the snow data in the 25-dataset setup is partitioned using a canonical grid. Therefore, the boundary annotations of the snow datasets are rectangles not only in geo-mbb-25 (as expected), but in geo-poly-25 and geo-appr-25 as well. As a result, the geospatial computations performed by the source selector for identifying irrelevant snow sources is much faster in the 25-dataset setup.

Table 5.2: Source Selection time (sec): Average (and standard deviation) over 100 query instances per query template (Q1 – Q7).

	geo-poly-27		geo-appr-27		geo-mbb-27		thm-27	
Q1	0.10	(0.02)	0.14	(0.02)	0.13	(0.02)	0.08	(0.01)
Q2	0.87	(0.06)	0.26	(0.02)	0.22	(0.02)	0.13	(0.02)
Q3	8.28	(0.23)	0.74	(0.09)	0.22	(0.03)	0.14	(0.02)
Q4	1.76	(0.14)	0.37	(0.16)	0.21	(0.08)	0.17	(0.07)
Q5	0.42	(0.08)	0.21	(0.02)	0.20	(0.03)	0.26	(0.13)
Q6	1.57	(0.10)	0.37	(0.09)	0.30	(0.11)	0.22	(0.07)
Q7	8.53	(0.24)	1.11	(0.34)	0.42	(0.20)	0.39	(0.26)
	geo-poly-25		geo-appr-25		geo-mbb-25		thm-25	
Q1	0.16	(0.02)	0.14	(0.02)	0.13	(0.02)	0.08	(0.01)
Q2	0.46	(0.03)	0.14	(0.02)	0.20	(0.02)	0.13	(0.02)
Q3	0.43	(0.04)	0.21	(0.03)	0.19	(0.02)	0.14	(0.08)
Q4	0.36	(0.09)	0.19	(0.04)	0.19	(0.07)	0.16	(0.06)
Q5	0.33	(0.08)	0.29	(0.07)	0.19	(0.02)	0.22	(0.08)
Q6	0.76	(0.07)	0.29	(0.11)	0.28	(0.09)	0.18	(0.04)
Q7	0.83	(0.14)	0.39	(0.14)	0.35	(0.14)	0.41	(0.34)

To sum up, we notice that source selection time depends on the complexity of the bounding polygon annotations of the sources; in other words, higher accuracy leads to slower source selection.

Comparison of source selection pruning

In the following, we focus on the precision of the pruning of each source selector (cf. Table 5.3). In particular, we compare the number of sources of each source selector with those of the other source selectors and with the optimal ones.

We observe that the thematic source selector keeps many irrelevant sources in the query plan. The source selector of thm exploits the thematic information (i.e., properties and URI-prefixes) of the sources and assigns to the administrative (resp. crop, snow) part of the query only the administrative (resp. crop, snow) sources. Moreover, in Q4-7, the administrative part of the query is further restricted to a single administrative source, because the pattern that specifies the name of the municipality appears in a single administrative source. This explains why, for example, thm-27 keeps 19 (i.e., 9 crop sources, 9 snow sources, and 1 administrative source) and not all 27 sources for all queries in the template. However, as expected, we will show that the geospatial selectors of the remaining federations achieve better pruning by exploiting the geospatial knowledge of the sources.

Regarding the geospatial selectors, we make three observations: First, we notice that the accuracy of the source selector increases as the accuracy of the source metadata increases. In particular, geo-poly is more precise than geo-appr, and geo-appr is more precise than geo-mbb. Second, we notice that the optimal pruning can be achieved only by geo-poly in Q1-3, Q5 (and also in Q4 only for the 27-dataset setup). Finally, the average number of sources for geo-appr and geo-mbb tend to be lower in Q1-3 than in Q4-7. In the remaining paragraphs we will try to explain these observations.

Q1-3 have geospatial selection filters, parameterized with a fixed polygon; thus, the geospatial selector operates by pruning all sources that are irrelevant according to the given query polygon. Since the less accurate geospatial summaries in geo-appr and geo-mbb are larger than the actual dataset boundaries of the sources, we can have a situation where the query polygon is disjoint from a data source but not disjoint from its bounding polygon annotation. This explains the

Table 5.3: Source Selection pruning: number of sources selected by the different source selection methods, average (minimum and maximmum) over 100 query instances per query template (Q1 – Q7).

	opt-27		geo-poly-27		geo-appr-27		geo-mbb-27		thm-27
Q1	1.12	(1-2)	1.12	(1-2)	1.32	(1-3)	1.62	(1-3)	9
Q2	2.24	(2-4)	2.24	(2-4)	2.64	(2-6)	3.24	(2-6)	18
Q3	2.24	(2-4)	2.24	(2-4)	2.64	(2-6)	3.24	(2-6)	18
Q4	2.25	(2-4)	5.48	(3-7)	5.48	(3-7)	5.90	(3-7)	10
Q5	2.00	(2-2)	2.00	(2-2)	5.48	(3-7)	5.82	(3-7)	10
Q6	3.00	(3-3)	3.00	(3-3)	9.96	(5-13)	10.64	(5-13)	19
Q7	3.24	(3-5)	6.48	(4-8)	9.96	(5-13)	10.64	(5-13)	19
	opt-25		geo-poly-25		geo-appr-25		geo-mbb-25		thm-25
Q1	1.12	(1-2)	1.12	(1-2)	1.32	(1-3)	1.62	(1-3)	9
Q2	2.18	(2-4)	2.18	(2-4)	2.38	(2-4)	2.68	(2-5)	16
Q3	2.18	(2-4)	2.18	(2-4)	2.38	(2-4)	2.68	(2-5)	16
Q4	2.14	(2-3)	4.34	(2-5)	4.59	(2-5)	4.59	(2-5)	8
Q5	2.00	(2-2)	2.00	(2-2)	5.48	(3-7)	5.82	(3-7)	10
Q6	3.18	(2-4)	4.97	(3-6)	8.82	(4-11)	9.41	(4-11)	17
Q7	3.18	(3-5)	4.97	(3-6)	8.82	(4-11)	9.41	(4-11)	17

optimal pruning for geo-poly (where the annotations are the exact boundaries of the sources). In the remaining geospatial federations, the source selection returns more sources, because there are cases where the parameterized polygon is contained in the approximated shape (for geo-appr) or in the minimum bounding box (for geo-mbb) of a neighbor source. This explains why geo-appr is equally or more specific than geo-mbb.

Q4-7 contain only geospatial join filters; therefore, the geospatial selector operates as follows; first, similarly to thm, it restricts the administrative part of the query in the source of the state where the municipality belongs to; then, it tries to prune all irrelevant crop and snow sources according to the boundary annotation of this administrative source and the geospatial filters of the query. As previously, we observe that accurate source descriptions can lead to more precise source selection. For instance, regarding Q5, geo-mbb (resp. geo-appr) prunes all crop sources that their bounding box (resp. approximated shape) is disjoint from the bounding box (resp. approximated shape) of the state of interest, while geo-poly, being more accurate, does better by keeping only the crop sources that refer this state, because the source boundaries do not overlap. This explains the optimal pruning of geo-poly for Q5.

Q4-7 present two additional challenges in source selection, which are either non-present or non-important in Q1-3. First, Q4 and Q8 contain a within-distance federated join operation, but unlike Q3, the shapes of interest do not intersect with a given polygon in the query. In such operations, the geospatial selector cannot achieve optimal pruning even in geo-poly. To give an example, consider Q4 and assume that the given municipality appears towards the center of the state. Since the exact geometric shape of the municipality will be discovered only during query execution, the source selector cannot exclude the possibility of its position being towards the border, thus keeping all the neighboring snow sources that may contain relevant data within 5km from the border of the state. Second, an overestimation of the set of sources can appear when the geographical partitions between the data to be geospatially joined are unaligned. Consider Q6; since in the 27-dataset setup all data layer partitions are geographically aligned (each source refers to a specific Austrian state), geo-poly-27 achieves optimal pruning (i.e., the source selector keeps the sources that refer to the state where the municipality belongs to). In contrast, since in the 25-dataset setup the snow data partition is not aligned with the other layers, geo-poly-25 keeps some irrelevant neighboring snow sources (i.e., those who intersect the state that belongs to the municipality but not the municipality itself) and thus does not achieve optimal pruning.

To sum up, we notice that the precision of the pruning by the geospatial source selector depends

Table 5.4: Query planning time (sec): Average (and standard deviation) over 100 query instances per query template (Q1 – Q7).

	geo-poly-27		geo-appr-27		geo-mbb-27		thm-27	
Q1	0.02	(0.01)	0.03	(0.01)	0.03	(0.01)	0.04	(0.01)
Q2	0.28	(0.04)	0.30	(0.03)	0.31	(0.03)	0.39	(0.04)
Q3	0.26	(0.04)	0.24	(0.04)	0.26	(0.03)	0.38	(0.04)
Q4	0.10	(0.02)	0.15	(0.09)	0.13	(0.03)	0.19	(0.07)
Q5	0.12	(0.03)	0.11	(0.02)	0.12	(0.01)	0.15	(0.02)
Q6	13.56	(0.29)	14.48	(0.44)	14.33	(0.40)	16.00	(0.35)
Q7	13.69	(0.31)	14.55	(1.37)	14.81	(2.61)	16.55	(3.95)
	geo-poly-25		geo-appr-25		geo-mbb-25		thm-25	
Q1	0.03	(0.01)	0.03	(0.01)	0.03	(0.01)	0.05	(0.01)
Q2	0.27	(0.05)	0.21	(0.02)	0.31	(0.02)	0.39	(0.04)
Q3	0.25	(0.05)	0.26	(0.03)	0.26	(0.03)	0.38	(0.05)
Q4	0.13	(0.07)	0.14	(0.10)	0.14	(0.07)	0.19	(0.13)
Q5	0.13	(0.02)	0.12	(0.02)	0.12	(0.01)	0.15	(0.02)
Q6	14.30	(0.49)	14.70	(0.48)	14.27	(0.46)	15.56	(0.29)
Q7	13.66	(0.28)	14.10	(0.46)	14.64	(2.43)	17.24	(6.36)

on the accuracy of the bounding polygon annotations of the sources. We observe that using the exact polygons of the sources could lead us to optimal pruning. Finally, we notice that in queries with WKT parameters (Q1-3) the geospatial source selectors tend to achieve a better pruning, even when using approximated shapes instead of exact polygons.

Comparison of query planning and execution

In the following, we discuss the effect of geospatial source selection on query planning and query execution phases of federated query processing. In particular, we compare the query planning times (cf. Table 5.4), the query execution times (cf. Table 5.5), and the error rates (cf. Table 5.6) of each federation of the experiment.

Regarding query planning time, we observe that, in general, geo-poly is the fastest; then it comes geo-appr; then we have geo-mbb; and finally thm is the slowest. This behaviour happens because having a large number of sources requires the construction of a larger query plan, which clearly affects the time for producing it; this is highlighted in Q6 and Q7 which have 14 triple patterns.

Regarding query execution, notice that only for some query templates we obtain a complete evaluation of all queries in the template For instance, ∼90% of the queries of Q7 fail to be processed by thm-27 due to errors in the execution phase (i.e., the error rate in Table 5.6 is equal to 0.9). These errors occur when a federator issues a huge workload of source queries to the endpoints, and as a result, the sources are not able to serve all these requests. Therefore, in order to compare two query executions, we should consider both their query execution times and their error rates. For instance, consider again Q7; the query execution of thm-27 is faster than that of geo-poly-27, but the error rate of geo-poly-27 is much lower than that of thm-27. Thus, we argue that geo-poly-27 is more effective than thm-27 for Q7, because we believe that having more but slower successful query runs is a more important characteristic (recall that the average execution time refers only to successful query runs).

The number of source queries in the execution plan affects not only the completion of the execution but the execution time as well; having more sources in the plan means that more source queries are issued by the federator to the source endpoints. Consider, for instance, Q2 and Q3; in both cases,

Table 5.5: Query execution time (sec): Average (and standard deviation) over 100 query instances per query template (Q1 – Q7).

	geo-poly-27		geo-appr-27		geo-mbb-27		thm-27	
Q1	0.06	(0.05)	0.04	(0.03)	0.04	(0.04)	0.06	(0.06)
Q2	0.05	(0.05)	0.04	(0.02)	0.04	(0.02)	0.17	(1.30)
Q3	0.21	(1.20)	0.17	(1.12)	0.17	(1.15)	0.06	(0.25)
Q4	6.87	(4.18)	6.47	(3.49)	6.81	(3.55)	8.52	(4.96)
Q5	0.13	(0.08)	0.08	(0.03)	0.09	(0.06)	0.12	(0.09)
Q6	0.11	(0.05)	2.20	(2.21)	2.18	(2.39)	2.30	(1.47)
Q7	23.19	(67.86)	4.14	(1.63)	4.43	(1.06)	6.64	(2.30)
	geo-poly-25		geo-appr-25		geo-mbb-25		thm-25	
Q1	0.05	(0.05)	0.04	(0.01)	0.04	(0.01)	0.05	(0.02)
Q2	0.05	(0.05)	0.04	(0.03)	0.04	(0.02)	0.30	(1.32)
Q3	0.19	(1.15)	0.17	(1.09)	0.17	(1.13)	2.52	(10.28)
Q4	7.35	(4.73)	7.33	(4.70)	7.43	(4.77)	8.81	(5.27)
Q5	0.26	(0.41)	0.11	(0.05)	0.08	(0.03)	0.10	(0.04)
Q6	0.77	(0.83)	1.37	(1.10)	1.52	(1.50)	1.46	(1.04)
Q7	19.44	(57.78)	3.10	(2.33)	3.07	(2.37)	38.69	(70.86)

the query execution of geo-poly-25 is faster than that of thm-25 by 1 order of magnitude; geo-poly-25 consults 1 (or in some cases 2) snow datasets, while thm-25 consults all snow datasets. Even though Semagrow manages to execute many queries in parallel, the duration of the query execution has to be as slow as the slowest source. By having a smaller the set of sources, the query executor avoids issuing queries to irrelevant larger datasets if they contain irrelevant results. Moreover, the time difference in query execution is more pronounced in queries that contain within-distance operations (e.g. in Q3), because the source endpoints use spatial indexes, hence geospatial queries that contain standard spatial relations (e.g., Q2) are executed faster.

The above discussion suggests that, according to the effectiveness of their query execution (which is based both on error rate and query execution time), the federators are to be ordered as follows: geo-poly, geo-appr, geo-mbb, and finally thm; the only exception being Q6 in the 25-dataset setup. In this sole case, geo-poly-25 is better than thm-25, but thm-25 has lower error rate than geo-appr-25 and geo-mbb-25. This final observation indicates that, even though our geospatial source selector can provide a faster query processing, it seems that in order to achieve better performance in geospatial scenarios, the remaining components of federated query processing should be extended with geospatial-specific optimizations as well.

To sum up, we observe that higher accuracy in geospatial source annotations (which results to a

Table 5.6: Error rate: Number of errors divided with the number of queries of each query template (Q1 – Q7).

	geo-poly-27	geo-appr-27	geo-mbb-27	thm-27	geo-poly-25	geo-appr-25	geo-mbb-25	thm-25
Q1	-	-	-	-	-	-	-	-
Q2	-	-	-	-	-	-	-	-
Q3	-	-	-	0.1	-	-	-	-
Q4	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
Q5	-	-	-	-	-	-	-	-
Q6	-	0.7	0.7	0.8	-	0.7	0.7	0.1
Q7	0.1	0.9	0.9	0.9	0.1	0.9	0.9	0.9

Table 5.7: Time overhead of the geospatial source selection: Average (and standard deviation) of the difference in total query processing time (in sec) of each geospatial federation minus the time of its corresponding thematic one, over the successful query instances of query template Q1 to Q5. A negative measurement indicates that the geospatial source selection overheads are recovered by faster query planning and execution. Q6 and Q7 are missing from the table since most queries in geo-appr, geo-mbb, and thm evoke errors during query execution phase.

	geo-poly-27		geo-appr-27		geo-mbb-27	
Q1	–0.01	(0.0000)	+0.03	(0.0000)	+0.03	(0.0001)
Q2	+0.51	(0.0013)	–0.10	(0.0013)	–0.12	(0.0013)
Q3	+8.03	(0.0004)	+0.45	(0.0003)	–0.05	(0.0002)
Q4	+0.23	(0.0017)	–1.28	(0.0018)	–1.20	(0.0017)
Q5	+0.14	(0.0002)	–0.13	(0.0002)	–0.12	(0.0002)
	geo-poly-25		geo-appr-25		geo-mbb-25	
Q1	+0.05	(0.0000)	+0.03	(0.0000)	+0.01	(0.0000)
Q2	–0.05	(0.0013)	–0.43	(0.0013)	–0.27	(0.0013)
Q3	–1.16	(0.0027)	–1.39	(0.0027)	–1.39	(0.0027)
Q4	–0.71	(0.0028)	–1.01	(0.0028)	–0.85	(0.0029)
Q5	+0.03	(0.0001)	+0.04	(0.0001)	–0.09	(0.0001)

lower number of sources per query) could help reducing the query planning time and the number of source queries issued by the federator, this increasing the effectiveness of query execution.

Comparison of overall query processing time

The discussion so far indicates that using a geospatial selector provides a positive impact on query planning and execution time. However, as the accuracy of the bounding polygons of the federated sources increases, source selection becomes slower, especially when using the exact polygons (as in geo-poly). The question that arises is whether the time overhead of the use of exact boundaries in source selection can be recovered by the remaining phases of query processing.

In Table 5.7, we draw a comparison between the time overheads of the geospatial source selectors; among the query instances that succeed in all 8 federations, we show the difference of the total query processing time of geo-poly-27 (resp. geo-appr-27, geo-mbb-27) minus the total query processing time of thm-27; then, the same for the 25-dataset setup; and finally, we report the average (and standard deviation) for each query template. We leave out Q6 and Q7 because in both query templates less than 5 instances succeed in all 6 federations; in this case, we will compare the federations w.r.t. the error rate (Table 5.6).

Q1 is the easiest query of the experiment (it contains a single data layer and one geospatial selection filter, i.e., a filter that contains a spatial relation in which one of the two parameters is a WKT value). Thus, all federators perform equally in Q1 (i.e., all time differences are less than 0.05 seconds). In contrast, Q6 and Q7 are the most difficult queries of the experiment (they contain 3 data layers, 3 geospatial joins, and no no WKT literals appear in the query body). Since most query instances of Q6 and Q7 fail to be processed, we compare the federations w.r.t. Table 5.6; we note that geo-poly performs the best since it minimizes the error rate.

The remaining queries (i.e., Q2-5) are somewhere in between Q1 and Q6-7 in terms of difficulty; this fact makes them easier to be processed by all federators of the experiment. In Table 5.7 we notice that geo-mbb and geo-appr outperform thm and geo-poly (overheads are smaller or similar). Regarding the comparison between geo-mbb and geo-appr though, we observe that geo-mbb is better in the 27-dataset setup, while geo-appr is better in the 25-dataset setup. This happens because the source selection cost in the 27-dataset setup is much higher than that of the 25-dataset setup (cf. Table 5.2). Thus, in the former setup, only geo-mbb-27 can benefit from the reduction in query

planning and execution times, while in the second one, the drop in planning and execution time of geo-appr-25 is greater than its source selection overhead.

To sum up, we observe that for difficult queries (such as queries that contain more than one geospatial join and no WKT literals in the query body), precise bounding polygons should be preferred, because otherwise we may face a computationally intensive query execution. In contrast, the use of less accurate descriptions will suffice if we consider simpler queries. However, it appears that no size fits all; for the setup that the partitions are unaligned, we benefit from the higher accuracy of the approximated shapes since one layer is already a geographical grid; while for the other setup the minimum bounding boxes are effective since all data layers are fully aligned according to the same administrative regions.

5.3 Evaluation using GDOBench suite of GeoFedBench

In the second part of the evaluation of Semagrow, we are using the GDOBench sute of GeoFedBench. This part of the evaluation extends Section 4.4 of D3.8 (where we evaluated the novel geospatial join optimization of Semagrow) by including a comparison with a setup that does not use Semagrow at all, but instead a setup where all data (from both datasets) are loaded in a single PostGIS database.

5.3.1 Experimental setup

For the experimental evaluation of Semagrow, we use two federations; one federation which uses the final version of Semagrow (semagrow-opt) and one federation that uses the final version of Semagrow, but with the geospatial join optimization for within distance queries discussed in Section 4 of D3.8 disabled (semagrow-std). We do not include pre-ExtremeEarth version of Semagrow in the evaluation results, because the queries of the workload use GeoSPARQL constructs that could not be handled in the old version. Each dataset is deployed in a separate GeoSPARQL endpoint. We use the Strabon geospatial RDF store [10] for serving the data. Strabon encapsulates PostGIS for performing spatial operations, and uses a spatial index to optimize query processing time.

In addition to the federated linked data experiment, we have loaded both datasets in a single PostGIS database, in order to compare the federated scenario with a centralized solution that uses only standard PostGIS (postgis). We have translated the queries Q1-3 of GDOBench suite into their corresponding SQL queries.

We use a Kubernetes 1.14 cluster with 1 master node and 8 worker nodes with a total if 120 cores and 264GB RAM. Experiment deployment and execution is done through the KOBE benchmarking engine, and the KOBE configurations for reproducing the experiments are publicly available.[3].

5.3.2 Experimental results

In the following, we discuss the experimental results of our evaluation. We first discuss the improvement of the execution time of the query set obtained from the data validation task (Q1-3) and then we analyze the performance of our optimization technique for various distances (Q4).

[3] The experiment specifications can be found in `https://github.com/semagrow/benchmark-geofedbench` as a part of the `geofedbench` suite of KOBE.

Table 5.8: Experimental results for Q1-3.

		query processing time					
		postgis		semagrow-std		semagrow-opt	
	#queries	total	average	total	average	total	average
Q1	2488	54 hours	79 sec	83 hours	120 sec	106 mins	2.6 sec
Q2	2488	54 hours	78 sec	82 hours	119 sec	99 mins	2.4 sec
Q3	2488	54 hours	79 sec	81 hours	117 sec	74 mins	1.8 sec

Data Validation Query Set (Q1-3)

In the first part of the experimental study, we compare the optimized (semagrow-opt) version of Semagrow over the unoptimized one (semagrow-std) and the setup with standalone PostGIS (postgis) using the query load obtained by the data validation task, i.e., Q1-3 from GDOBench suite. Table 5.8 contains the experimental results. For every query template, we illustrate: the number of queries for each template (#queries), and the query processing time. For each federation, we display the total time to evaluate the query load and the average time for each query of the query load.

Comparing PostGIS with the unoptimized version of Semagrow, we observe that PostGIS is faster; i.e., postgis spends 65% of the time of semagrow-std. This is an expected result because having all data in a centralized PostGIS is faster than performing geospatial joins in a federated setting (i.e., over HTTP). The two measurments though are roughly in the same order of magnitude. In contrast, notice that the queries are much faster if we use the optimized version of Semagrow. The unoptimized version would require several days for the evaluation of the full workload, while with our optimization technique the total evaluation reduces to several hours. Thus, notice that the final, optimized version of Semagrow is faster than standalone PostGIS in the queries obtained by land-usage data validation task.

As discussed in Section 4.4 of D3.8, the operation "retrieve all WKTs from INVEKOS within 10m distance from a specific WKT (obtained from LUCAS)" is the costliest operation in the data validation task. As a result, it is safe to conclude that our optimization technique which targets such queries is the reason for the extreme speed-up of the queries of the experiment.

Within-distance operation from a given LUCAS point (Q4)

In the second part of the experimental study, we compare the optimized (semagrow-opt) over the unoptimized (semagrow-std) version of Semagrow using a query of fetching all INVEKOS parcels that are found within increasing distance from a specific LUCAS point. Table 5.9 contains the experimental results. For every instance of the query template Q4, we illustrate: the distance parameter of the within-distance operation, the query processing time and the number of results for semagrow-std and semagrow-opt. Moreover, we display the number and percentage of shapes that are pruned from INVEKOS by the additional filter that the optimization process places in the source query that corresponds right part of the federated join.

Regarding semagrow-std, which issues the unoptimized query in INVEKOS, we notice that every query requires at least 57 seconds. For parameterized distance between 10 meters and 10 kilometers, the query time is $57 - 58$ seconds, even if the result is either very small (2 results) or moderate (4,700 results). Only for distances greater than 50 kilometers, where we have a very large result set (greater than 140,000 results), the query time seems to increase as we increase the distance parameter. This behavior can be explained if we consider that the unoptimized query cannot be answered using the spatial index of the source, and the source has to check for every shape in INVEKOS if it is within a specific distance. Thus, the experimental results can be explained as

Table 5.9: Experimental results for Q4.

			query processing time			
	distance	#results	semagrow-std	semagrow-opt	shapes pruned by optimization	
Q4	10 m	2	58 sec	0.1 sec	2,008,134	(>99%)
Q4	100 m	7	57 sec	0.1 sec	2,008,129	(>99%)
Q4	1 km	70	58 sec	0.1 sec	2,008,004	(>99%)
Q4	10 km	4,739	57 sec	1.2 sec	1,996,169	(99%)
Q4	50 km	141,973	72 sec	26 sec	1,702,032	(84%)
Q4	100 km	528,026	110 sec	86 sec	1,212,393	(60%)

follows; the time needed to check potential candidates within a given distance is around 1 minute, and the remaining time is used for passing the results back to the clients. Notice that in our case, the transfer cost can be relatively high (e.g., 50 seconds for 500,000 results), because the result set includes WKT values which are in general long strings.

Regarding semagrow-opt, which issues the optimized query in INVEKOS, we notice that the query processing time is analogous to the size of the distance parameter, i.e., for a smaller distances we have a faster query processing time. Recall that the optimized source query has an additional filter expression, which is used to prune all shapes that are too far away from the given LUCAS WKT. Indeed, we observe that there exists a connection between the amount of the pruning of the additional filter and the query processing time. Unlike previously, where the source had to consider all shapes from INVEKOS, the source here computes the distance for a reduced set of candidate shapes, which is relevant to the distance parameter. Moreover, since the `geof:sfIntersects` function of the additional filter can be evaluated using the spatial index, this filter does not introduce any time overhead in the evaluation of the query.

Comparing the two approaches, we observe that the optimized version reduces the query processing time by 3 orders of magnitude for distances less than 1 km and by 2 orders of magnitude for distances around 10 km. For larger distances, the time difference is less pronounced, but in any case, we can safely conclude that the optimization technique can be effective for federated within-distance queries for any distance length.

5.4 Summary

We evaluated Semagrow using the GeoFedBench benchmark. We observe that the new version of Semagrow that we have developed during ExtremeEarth was able to provide an effective performance on the benchmark, even though the queries of GeoFedBench challenge all phases of federated query processing (cf. Subsection 3.3.4)

Regarding source selection, Semagrow is able to handle effectively chains of known properties of the form `?s geo:hasGeometry ?g.?g geo:asWKT ?wkt` which appear in linked geospatial data, through the use of the extended thematic source selector. Moreover, Semagrow can exploit geospatial dataset metadata to reduce the set of sources that will be tested as potentially holding relevant data w.r.t. geospatial filters. Regarding query planning, Semagrow is able to process effectively complex SPARQL constructs (such as subqueries, negation though `FILTER NOT EXISTS`, etc.). Moreover, it can evaluate geospatial operation effectively, by pushing filters and joins to its relevant sources through specific optimizations. Regarding query execution, we observe that the implementation of the relevant operations of the GeoSPARQL extension (e.g., through geospatial join optimization of within-distance queries).

The experimental results can be summarized as follows:

- In GSSBench suite of GeoFedBench, the new source selector of Semagrow keeps in average 12% of the federated sources in the query plan when using precise geospatial source metadata (the corresponding measurement is 100% for pre-ExtremeEarth Semagrow and 9% for the optimal source set).
- The queries of GDOBench suite of GeoFedBench, which are obtained by the use case of validating land usage data (joint work with UNITN) are evaluated faster by 2 orders of magnitude if we use Semagrow federation over 2 GeoSPARQL endpoints instead of placing all data in a PostGIS database.

6. Scale-to-Petabyte experiment

In the final part of the evaluation of the linked data tools of querying and federating big linked geospatial data of ExtremeEarth, we combine the cluster-level scalability results offered by Strabo2 with Semagrow's ability to transparently federate multiple such clusters. The aim is to prove that the combination of these key ExtremeEarth technologies can bring geospatial linked data query processing to the order of magnitude of PBs.

Unfortunately, we do not have either the data or the infrastructure to experiment at the PB scale. However, we can show that the Linked Data techniques developed in ExtremeEarth *should* scale to PBs. Regarding federated query processing, we can achieve this by using federations with a high number of endpoints, such that each endpoint behaves as if contained TBs of data.

6.1 Advances in querying and federating big linked geospatial data

Our work during ExtremeEarth advances the state-of-the-art of federating big linked geospatial data in several aspects. These aspects include advancements in source selection, query optimization, and benchmarking of federated query processors.

The new, sophisticated source selector mechanism of Semagrow [21][Section 5.1] allows distributing large volumes of data (that scale to PB) without unnecessarily accessing all the data. In particular, experimental results show that when the source endpoints refer to non-overlapping areas, the geospatial source selector of Semagrow provides effective pruning on the source endpoints that appear in the query execution plan, thus increasing the effectiveness of the overall query processing.

Semagrow has been extended with several optimizations that increase the efficiency of the query execution plan. In particular, the new filter and join pushdown optimizations [21][Section 5.2] reduce the number of source queries issued by the federator to the source endpoints and reduce the the number of intermediate query results transferred over the network. Since Semagrow queries the endpoints efficiently, it increases the scalability by allowing each member of the federation to contain a large dataset. For example, if each member of the federation can serve multiple TBs, then we can have federations of PBs.

Apart from the advancements in federated query processing by Semagrow, during ExtremeEarth we have also advances in benchmarking of linked data systems. Recall that KOBE can be used to simulate real-life scenarios through injecting network delays to the source endpoints, which mimic real-world dataset server latency limitations. However, endpoint delay does not always appear on situations where we have network problems or a lot of clients that are trying to connect with the endpoint, but also in cases where the endpoint serves a very large volume of data that is time consuming to process. As a result, KOBE can be used so that smaller datasets to behave as larger ones by simulating endpoint delays. Intuitively speaking, if each endpoint "behaves" as if it contains TBs of data, then the federation "behaves' as if it federates PBs of data.

In the current experimental results, Strabo2 is able to process queries of the synthetic Geographica3 benchmark with an average execution time of 107 seconds over 1.16 TBs of data (in NTriples format). However, in the two scalability experiments that we have performed, it is suggested that Stabo2 is able to handle datasets with even larger volumes of data, given that more computing resources are available.

6.2 Experimental setup

In this section we discuss the experimental setup of our evaluation. We make use of synthetic datasets and queries that are based on those of Geographica2.

Endpoints For our evaluation we use the synthetic dataset of Geographica2 [1]. We use this dataset as a initial source in order to construct a federation with 100 endpoints. The endpoints are created as follows: First, we calculate the minimum bounding box of all shapes in the dataset. We then split the minimum bounding box into a 10×10 grid, and we create one GeoSPARQL endpoint for each of the 100 rectangles. We populate each endpoint with triples that correspond to features from the initial dataset such that their geometry is *within* the corresponding rectangle. Finally, we modify the URIs of all resources so that all resources that appear in the same endpoint a common prefix, which is unique among the prefixes of all endpoints of the experiment. The instructions [2] and the code [3] for creating the RDF dumps of the source endpoints are publicly available. We use Strabon for serving the data. The statistics of the source endpoints can be found in Tables 6.1, 6.2. In Figure 6.1 we illustrate the area that corresponds to each source endpoint.

Federator configuration We use Semagrow [4] for federating the 100 source endpoints. The configuration of Semagrow is created using a bash script [5]. Regarding source selection, Regarding source selection, Semagrow is configured to use metadata-based thematic source selector and geospatial selector (i.e., the source selector that uses ASK queries is disabled in the `repository.ttl` configuration file).

Queries For the experimental evaluation we used 36 queries, taken from the Geographica2 benchmark [6]. Information about the queries are illustrated in Table 6.3. All queries contain a single geospatial filter, and can be divided in two categories according to the arguments of the filter. PBQ00-PBQ23 are *geospatial selectors*, i.e., the geospatial filter contains one variable and one WKT literal:

```
... ?g geo:asWKT ?w . FILTER(geof:sfWithin(?w, "KNOWN_WKT"^^geo:wktLiteral)) ...
```

while PBQ24-PBQ35 are *geospatial joins*, i.e., both arguments of the filter are variables:

```
... ?g1 geo:asWKT ?w1 .  ?g2 geo:asWKT ?w2 . FILTER(geof:sfWithin(?w1, ?w2)) ...
```

The queries of the second category are generally considered more difficult than those of the first category, especially in a federated setting.

Experiment deployment and execution We use a Kubernetes 1.20.7 cluster with 3 master nodes and 14 worker with a total if 136 cores and 606GB RAM. Experiment deployment and execution is done through the KOBE benchmarking engine, and the KOBE configurations for reproducing the experiments are publicly available.[7]. We have executed the experiment 6 times. In each experiment execution, the endpoints have been configured with varying delay parameters, that is 0 seconds (no delay), 1 second, 10 seconds, 1 minute, 5 minutes, and 10 minutes. Finally, for every endpoint delay parameter value, each query has been executed 3 times.

[1] Cf. `http://geographica2.di.uoa.gr/datasets/generator_512.tar.xz`
[2] Cf. `https://github.com/semagrow/experiment-mandie/blob/master/files/dataset-instructions.txt`
[3] Cf. `https://github.com/semagrow/semagrow-geotools`
[4] Cf. `https://github.com/semagrow/semagrow/commits/59118ea88d709dea14a19a0c13f5ca7d7075d41a`
[5] Cf. `https://github.com/semagrow/experiment-mandie/blob/master/scripts/generate-metadata-ttl.sh`
[6] Cf. `http://geographica2.di.uoa.gr/queries/synthetic.lst`
[7] Cf. `https://github.com/semagrow/experiment-mandie`

Table 6.1: Federated endpoints used in the experiment. For each endpoint we illustrate the number of triples (#triples), the number of geometries (#geom), the number of tags (#tags), the number of land ownerships (#lo), the number of states (#st), and the number of points of interest (#poi).

	#triples	#geom	#tags	#lo	#st	#poi
PBD00	257	35	38	15	5	15
PBD01	201	27	30	15	4	8
PBD02	194	26	29	15	3	8
PBD03	271	37	40	15	7	15
PBD04	215	29	32	15	6	8
PBD05	222	30	33	15	7	8
PBD06	222	30	33	15	7	8
PBD07	271	37	40	15	7	15
PBD08	222	30	33	15	7	8
PBD09	222	30	33	15	7	8
PBD10	257	35	38	15	5	15
PBD11	201	27	30	15	4	8
PBD12	194	26	29	15	3	8
PBD13	271	37	40	15	7	15
PBD14	215	29	32	15	6	8
PBD15	222	30	33	15	7	8
PBD16	222	30	33	15	7	8
PBD17	271	37	40	15	7	15
PBD18	222	30	33	15	7	8
PBD19	222	30	33	15	7	8
PBD20	271	37	40	15	7	15
PBD21	201	27	30	15	4	8
PBD22	201	27	30	15	4	8
PBD23	271	37	40	15	7	15
PBD24	222	30	33	15	7	8
PBD25	222	30	33	15	7	8
PBD26	229	31	34	15	8	8
PBD27	278	38	41	15	8	15
PBD28	229	31	34	15	8	8
PBD29	222	30	33	15	7	8
PBD30	271	37	40	15	7	15
PBD31	201	27	30	15	4	8
PBD32	201	27	30	15	4	8
PBD33	271	37	40	15	7	15
PBD34	222	30	33	15	7	8
PBD35	222	30	33	15	7	8
PBD36	222	30	33	15	7	8
PBD37	278	38	41	15	8	15
PBD38	229	31	34	15	8	8
PBD39	222	30	33	15	7	8
PBD40	271	37	40	15	7	15
PBD41	194	26	29	15	3	8
PBD42	201	27	30	15	4	8
PBD43	257	35	38	15	5	15
PBD44	222	30	33	15	7	8
PBD45	215	29	32	15	6	8
PBD46	222	30	33	15	7	8
PBD47	271	37	40	15	7	15
PBD48	222	30	33	15	7	8
PBD49	222	30	33	15	7	8
PBD50	271	37	40	15	7	15
PBD51	194	26	29	15	3	8
PBD52	201	27	30	15	4	8
PBD53	257	35	38	15	5	15
PBD54	222	30	33	15	7	8
PBD55	222	30	33	15	7	8
PBD56	222	30	33	15	7	8
PBD57	271	37	40	15	7	15
PBD58	226	30	34	15	7	8
PBD59	222	30	33	15	7	8
PBD60	271	37	40	15	7	15
PBD61	194	26	29	15	3	8
PBD62	201	27	30	15	4	8
PBD63	257	35	38	15	5	15
PBD64	222	30	33	15	7	8
PBD65	222	30	33	15	7	8
PBD66	222	30	33	15	7	8
PBD67	271	37	40	15	7	15
PBD68	222	30	33	15	7	8
PBD69	222	30	33	15	7	8
PBD70	271	37	40	15	7	15
PBD71	194	26	29	15	3	8
PBD72	201	27	30	15	4	8
PBD73	257	35	38	15	5	15
PBD74	222	30	33	15	7	8
PBD75	215	29	32	15	6	8
PBD76	222	30	33	15	7	8
PBD77	271	37	40	15	7	15
PBD78	222	30	33	15	7	8
PBD79	222	30	33	15	7	8
PBD80	271	37	40	15	7	15
PBD81	201	27	30	15	4	8
PBD82	201	27	30	15	4	8
PBD83	271	37	40	15	7	15
PBD84	222	30	33	15	7	8
PBD85	222	30	33	15	7	8
PBD86	222	30	33	15	7	8
PBD87	278	38	41	15	8	15
PBD88	229	31	34	15	8	8
PBD89	222	30	33	15	7	8
PBD90	271	37	40	15	7	15
PBD91	201	27	30	15	4	8
PBD92	201	27	30	15	4	8
PBD93	271	37	40	15	7	15
PBD94	222	30	33	15	7	8
PBD95	222	30	33	15	7	8
PBD96	229	31	34	15	8	8
PBD97	278	38	41	15	8	15
PBD98	229	31	34	15	8	8
PBD99	222	30	33	15	7	8

Table 6.2: Statistics about the federated endpoints used in the experiment. (Total and Average values of the measurements of Table 6.1).

	#triples	#geom	#tags	#lo	#st	#poi
total	23170	3138	3439	1500	628	1010
average	231.7	31.4	34.4	15.0	6.3	10.1

Table 6.3: Information about the queries used in the experiment. For each query we illustrate the number of triple patterns (#tp), whether the query is a geospatial selection or a geospatial join (type), the geospatial relation of the filter (relation), and for geospatial selections, the area of the parameterized shape that appears in the query w.r.t. the total area (area).

	#tp	type	relation	area
PBQ00	4	Selection	Intersects	100%
PBQ01	4	Selection	Intersects	100%
PBQ02	4	Selection	Intersects	75%
PBQ03	4	Selection	Intersects	75%
PBQ04	4	Selection	Intersects	50%
PBQ05	4	Selection	Intersects	50%
PBQ06	4	Selection	Intersects	25%
PBQ07	4	Selection	Intersects	25%
PBQ08	4	Selection	Intersects	10%
PBQ09	4	Selection	Intersects	10%
PBQ10	4	Selection	Intersects	1%
PBQ11	4	Selection	Intersects	1%
PBQ12	4	Selection	Within	100%
PBQ13	4	Selection	Within	100%
PBQ14	4	Selection	Within	75%
PBQ15	4	Selection	Within	75%
PBQ16	4	Selection	Within	50%
PBQ17	4	Selection	Within	50%

	#tp	type	relation	area
PBQ18	4	Selection	Within	25%
PBQ19	4	Selection	Within	25%
PBQ20	4	Selection	Within	10%
PBQ21	4	Selection	Within	10%
PBQ22	4	Selection	Within	1%
PBQ23	4	Selection	Within	1%
PBQ24	8	Join	Intersects	–
PBQ25	8	Join	Intersects	–
PBQ26	8	Join	Intersects	–
PBQ27	8	Join	Intersects	–
PBQ28	8	Join	Touches	–
PBQ29	8	Join	Touches	–
PBQ30	8	Join	Touches	–
PBQ31	8	Join	Touches	–
PBQ32	8	Join	Within	–
PBQ33	8	Join	Within	–
PBQ34	8	Join	Within	–
PBQ35	8	Join	Within	–

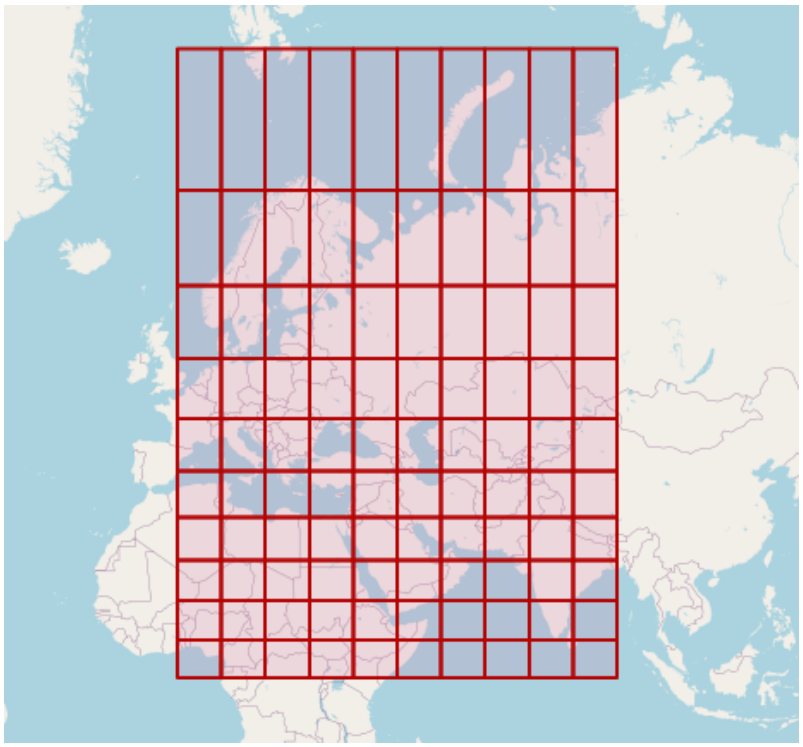

Figure 6.1: The 10×10 grid used to partition the data of the dataset. Each cell corresponds to a GeoSPARQL endpoint.

6.3 Experimental results

In Table 6.4 we illustrate the experimental results. We split the time measurements for each phace of the federated query processing, namely source selection time, query planning time, and query execution time, and for each phase we provide the time measurement for each endpoint delay parameter. Moreover, in Table 6.5 we provide a summary of the results of the previous table. In the following, we will discuss the performance for each phase of federated query processing in detail.

Regarding source selection time, we observe that in PBQ00-PBQ23 we spend 0.4-0.9 seconds. while in PBQ24-PBQ36 we spend 1.1-1.4 seconds. It is not hard to see that source selection time depends on the number of triple patterns of the query. Thus, this difference can be explained by the fact that queries PBQ00-PBQ23 comprise 4 triple patterns each, while queries PBQ24-PBQ36 difference 8 triple patterns each. Moreover, we do not observe any correlation between source selection time and endpoint delay, and depends only on the query itself and not in the size of the federated data. This can be explained due to the fact that in our experimental setup the source selection of Semagrow is based solely on dataset metadata and does not make use of ASK queries. Since the schema of the data does not change, neither the size of the metadata do. Finally, we observe that for the queryset of the experiment, Semagrow spends on source selection at most 1.4 seconds.

Regarding source selection pruning (column #s), we observe that we have a very good pruning of the number of sources that appear in the query execution plan. In queries PBQ00-PBQ23 (i.e., the geospatial selection queries of the experiment) the number of sources that the source selection keeps is analogous to the area of the parameterized shape that appears in the query. For instance, consider PBQ02, which returns features within a WKT of area equal to 75% of the total experiment area. In this query, the source selector keeps 81 (of total 100) sources of the experiment, or equivalently it prunes 19 irrelevant sources. In the remaining queries of the experiment, the source selector keeps all 100 sources in the plan because these queries contain geospatial joins and require data from all sources of the experiment.

Regarding query planning, the new version of the Semagrow planner operates by checking whether a set of triple patterns (with their corresponding filters) can be grouped in the same subquery if the patterns are assigned the same set of sources by the source selector and these sources are disjoint (two sources are disjoint if no URI appears in both sources and all WKT literals are disjoint) [8]. Notice that the query planning time is very short and less than 4 seconds for every query; this happens because the test whether this optimization is applicable occurs early in the process of query planning, thus speeding up the computation of the query plan. Moreover, notice that query planning time clearly is related to the number of sources that the source selector produced; this fact showcases that for the queries of the experiment, even in a high number of disjoint sources, the planning time is acceptable from a practical point of view (for 1 source it is less than 0.05 sec, for 100 sources it is around 3.5 sec). As with source selection time, the query planning time does not depend on the endpoint delay.

Naturally, query execution time is the only time measurement that clearly depends on the endpoint delay, because in this phase the federator communicates with the source endpoints in order to evaluate the query execution plan. In particular, we notice that for delays ≥ 10 seconds the query execution time is (almost) equal to the endpoint delay. This fact can be explained as follows: First, according to its execution plan, each query of the experiment is evaluated by issuing the query as is in all relevant source endpoints and then by calculating the union of the query results of the individual queries – since all sources are disjoint, there are no shapes that belong to more than one sources (PBQ00-PBQ23), and there is no shape that intersects/touches/is within any other shape that belongs to another source (PBQ24-PBQ35). Therefore, the reason why query execution time is equal to the endpoint delay, is because the source queries are evaluated in parallel, and the

[8] Cf. `https://github.com/semagrow/semagrow/pull/88`

Table 6.4: Experimental results. We illustrate source selection time, query planning time and query execution time, for each experiment execution. Each experiment is characterized by its delay (namely 0s (no delay), 1s, 10s, 1m, 5m, and 10m). All times are average times of 3 runs, and are displayed in seconds. Moreover, we illustrate the number of sources that appear in the execution plan (#s), and the number of results of each query (#r), which are the same for all experiment executions.

	source selection time						query planning time						query execution time						#s	#r
	0s	1s	10s	1m	5m	10m	0s	1s	10s	1m	5m	10m	0s	1s	10s	1m	5m	10m	all	all
PBQ00	0.9	0.9	0.9	0.9	0.8	0.9	3.5	3.6	3.2	3.6	3.3	3.2	1.0	1.7	11.1	61.0	301.0	601.2	100	1500
PBQ01	0.8	0.8	0.6	0.7	0.7	0.6	3.4	3.4	2.9	3.4	3.0	3.0	0.1	1.2	10.2	60.2	300.2	600.2	100	100
PBQ02	0.7	0.7	0.6	0.7	0.6	0.6	2.1	2.1	1.9	2.2	1.8	1.8	0.1	1.2	10.1	60.2	300.2	600.2	81	1215
PBQ03	0.6	0.6	0.6	0.7	0.6	0.6	2.1	2.1	1.8	2.1	1.8	1.8	0.1	1.1	10.1	60.2	300.1	600.2	81	81
PBQ04	0.6	0.6	0.5	0.6	0.5	0.5	1.3	1.3	1.1	1.3	1.1	1.2	0.1	1.1	10.1	60.1	300.1	600.1	64	735
PBQ05	0.6	0.6	0.5	0.6	0.5	0.6	1.3	1.3	1.1	1.3	1.1	1.1	0.1	1.1	10.1	60.1	300.1	600.1	64	49
PBQ06	0.6	0.6	0.5	0.6	0.5	0.5	0.4	0.4	0.4	0.4	0.3	0.3	0.1	1.1	10.1	60.1	300.1	600.1	36	375
PBQ07	0.6	0.6	0.5	0.6	0.5	0.5	0.4	0.4	0.3	0.4	0.3	0.3	0.1	1.1	10.1	60.1	300.1	600.1	36	25
PBQ08	0.6	0.5	0.5	0.6	0.5	0.5	0.1	0.1	0.1	0.1	0.1	0.1	0.0	1.0	10.1	60.1	300.1	600.1	16	135
PBQ09	0.5	0.6	0.5	0.6	0.5	0.5	0.1	0.1	0.1	0.1	0.1	0.1	0.0	1.0	10.1	60.1	300.1	600.1	16	9
PBQ10	0.5	0.5	0.5	0.5	0.4	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	10.0	60.0	300.1	600.1	1	0
PBQ11	0.5	0.5	0.4	0.5	0.4	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	10.0	60.0	300.0	600.1	1	0
PBQ12	0.6	0.6	0.5	0.6	0.5	0.5	3.2	3.2	2.7	3.2	2.6	2.8	0.1	1.2	10.2	60.2	300.1	600.2	100	1010
PBQ13	0.6	0.6	0.5	0.6	0.6	0.5	3.2	3.2	2.8	3.2	2.8	2.8	0.1	1.1	10.1	60.1	300.1	600.1	100	100
PBQ14	0.6	0.6	0.5	0.6	0.6	0.6	2.0	2.0	1.8	2.1	1.7	1.8	0.1	1.1	10.1	60.1	300.1	600.1	80	762
PBQ15	0.6	0.6	0.5	0.6	0.5	0.5	2.0	2.0	1.7	2.0	1.7	1.7	0.1	1.1	10.1	60.1	300.1	600.1	80	70
PBQ16	0.6	0.6	0.5	0.6	0.5	0.5	1.1	1.1	1.0	1.2	1.0	1.0	0.1	1.1	10.1	60.1	300.1	600.1	60	505
PBQ17	0.6	0.6	0.5	0.6	0.5	0.5	1.1	1.2	1.0	1.2	1.0	1.0	0.1	1.1	10.1	60.1	300.1	600.1	60	50
PBQ18	0.6	0.6	0.5	0.6	0.5	0.5	0.3	0.3	0.2	0.3	0.2	0.2	0.1	1.1	10.1	60.1	300.1	600.1	30	257
PBQ19	0.6	0.6	0.5	0.6	0.5	0.5	0.3	0.3	0.2	0.3	0.2	0.2	0.0	1.1	10.1	60.1	300.1	600.1	30	20
PBQ20	0.5	0.5	0.5	0.6	0.4	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	10.1	60.0	300.1	600.1	10	101
PBQ21	0.5	0.5	0.4	0.5	0.5	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	10.1	60.0	300.1	600.1	10	10
PBQ22	0.5	0.6	0.5	0.5	0.5	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	10.0	60.0	300.1	600.1	10	0
PBQ23	0.5	0.6	0.5	0.5	0.5	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	10.0	60.1	300.1	600.1	10	0
PBQ24	1.3	1.4	1.1	1.4	1.2	1.2	3.2	3.2	2.7	3.2	2.7	2.8	0.1	1.2	10.2	60.2	300.2	600.2	100	2360
PBQ25	1.3	1.4	1.1	1.4	1.2	1.2	3.2	3.2	2.7	3.2	2.8	2.8	0.1	1.1	10.1	60.2	300.1	600.1	100	80
PBQ26	1.3	1.4	1.1	1.4	1.2	1.1	3.2	3.2	2.7	3.2	2.7	2.7	0.4	1.1	10.1	60.2	300.1	600.1	100	197
PBQ27	1.3	1.3	1.2	1.4	1.2	1.2	3.2	3.2	2.7	3.2	2.8	2.7	0.1	1.1	10.1	60.2	300.2	600.1	100	14
PBQ28	1.3	1.3	1.1	1.3	1.2	1.2	3.2	3.2	2.7	3.2	2.8	2.8	0.2	1.2	10.1	60.2	300.1	600.1	100	1140
PBQ29	1.3	1.3	1.1	1.4	1.2	1.2	3.2	3.2	2.7	3.2	2.8	2.8	0.1	1.2	10.1	60.2	300.1	600.2	100	139
PBQ30	1.3	1.4	1.2	1.4	1.2	1.2	3.2	3.2	2.8	3.2	2.7	2.8	0.1	1.2	10.1	60.2	300.1	600.1	100	139
PBQ31	1.3	1.3	1.2	1.4	1.2	1.1	3.2	3.2	2.7	3.2	2.8	2.7	0.1	1.1	10.1	60.2	300.1	600.1	100	0
PBQ32	1.4	1.4	1.2	1.3	1.2	1.2	3.2	3.2	2.7	3.2	2.8	2.8	0.1	1.2	10.1	60.2	300.1	600.2	100	1010
PBQ33	1.3	1.4	1.2	1.4	1.2	1.2	3.2	3.3	2.7	3.2	2.8	2.8	0.1	1.1	10.1	60.2	300.1	600.1	100	88
PBQ34	1.3	1.3	1.2	1.4	1.1	1.2	3.2	3.2	2.8	3.2	2.7	2.8	0.1	1.1	10.1	60.1	300.1	600.1	100	100
PBQ35	1.3	1.3	1.2	1.4	1.1	1.2	3.2	3.2	2.7	3.2	2.8	2.8	0.1	1.1	10.1	60.1	300.1	600.1	100	45

Table 6.5: Experimental results. Summary of Table 6.4. We illustrate average source selection time according to the number of the triple patterns of the query, average planning time according to the number of sources that appear in the plan, average query execution time according to the endpoint delay.

#triple patterns	source selection time (s)	#sources in the plan	query planning time (s)	endpoint delay (s)	query execution time (s)
4	0.62	100	3.01	0	0.11
8	1.26	∼80	1.92	1	1.12
		∼60	1.14	10	10.13
		∼30	0.30	60	60.15
		∼10	0.05	300	300.13
		1	0.00	600	600.15

federator simply passes all query results from the source endpoints to the client. Finally, we notice that the queries of the experiment return a small number of results each, which is an expected behavior because the federated sources contain a small number of triples and shapes.

To summarize, we observe that for 100 disjoint source endpoints (i.e. no URI belongs to more than one sources and all pairs of shapes that belong to different sources are disjoint), the sum of source selection time and the query planing time for each query is less than 5 sec. As a result, since the query execution time is (almost) equal to the endpoint delay, for delays greater than 1 minute the *overall query processing time* is equal to the endpoint delay. We note though that since the queries of our experiment returned a small number of results, this evaluation is suited mainly for needle-in-a-haystack querying scenarios; that is federated queries that the queries issued by the federator return a small number of results.

6.4 Summary

In this chapter, we performed a federated experiment that uses queries from the Geographica2 benchmark, which can be used for showing scalability to PBs of the linked data tools of ExtremeEarth. In particular, we combine the cluster-level scalability results offered by Strabo2 with Semagrow's ability to transparently federate multiple such clusters. The aim is to prove that the combination of these key ExtremeEarth technologies can bring geospatial linked data query processing to the order of magnitude of PBs.

In this experiment, Semagrow evaluates a set of needle-in-a-haystack queries over a federation of 100 disjoint GeoSPARQL endpoints that behave as if they contained larger volumes of data through KOBE's simulated endpoint delay. We observe that the overall query processing time is equal to the endpoint delay. This result, suggests that if the federated endpoint can process TBs of data in an acceptable time, then a Semagrow federation can process 100 such endpoints (i.e., PBs of data) in acceptable time.

7. Conclusions

In this deliverable, we develop an evaluation framework for big linked geospatial data systems. This framework comprises a benchmarking engine and a set of benchmarks for linked geospatial data systems. In addition, this deliverable contains the final experimental evaluation of Strabo2 and Semagrow.

We presented the KOBE Benchmarking Engine, an open-source benchmarking engine that reads declarative benchmark and experiment definitions and uses modern containerization and Cloud computing technologies for automating the process of deployment, initialization, and experiment execution processes. The engine offers simulation of realistic endpoint delays and provides collection of logs and visualization of the experiment results using a WebUI. We have used KOBE for conducting our experiments.

We provided three new benchmarks for linked geospatial data. First, the Geographica2 benchmark, for evaluating single node linked geospatial stores; second, the Geographica3 CL benchmark, for evaluating distributed big linked geospatial stores; and finally, the GeoFedBench benchmark for evaluating federated linked geospatial data engines. These benchmark are integrated in KOBE and make use of datasets and queries from the use cases of ExtremeEarth.

We presented the experimental evaluation of Strabo2. First, we used the Geographica3 synthetic dataset to generate a dataset with size 1.16 TB in NTriples text format, store it in the Hopsworks deployment in CREODIAS, and execute queries produced by the benchmark with an average execution time of 107 seconds. Then, we evaluated specific aspects of the system. We found that the caching of thematic tables leads to an improvement of 16% n execution time for our query set, whereas the use of persistent spatial index and partitioning leads to a reduction of more than 50% for the queries that contain spatial selections. The use of JedAI-Spatial in order to cache the qualitative spatial relations between the geometries of the dataset is not beneficial for very selective queries, but it has important impact on queries that access a large portion of the dataset. Finally we presented the execution times for real world queries and datasets from the use cases, which in most cases vary from few seconds to a few minutes.

We presented the experimental evaluation of the new version of Semagrow. First, we used GeoFedBench: we observed that Semagrow achieves a good perfrmance, even though GeoFedBench presents many challenges in all phases of federated query processing, both thematically and geospatially. Regarding source selection, we observed that the new source selection of Semagrow reduces the number sources that appear in the execution plans of the queries of GSSBench suite by 88% (comparing to the pre-ExtremeEarth version). Regarding query execution, we observed that Semagrow is faster than standalone PostGIS by 2 orders of magnitude in the queries of GDOBench suite.

Finally, we demonstrated scalability to Petabytes of data by combining the cluster-level scalability results offered by Strabo2 with Semagrow's ability to transparently federate multiple such clusters. In particular, we conducted an experiment which uses a Semagrow federation of 100 GeoSPARQL endpoints that behave as if they contained larger volumes of data through KOBE's simulated endpoint delay, and we used data and a series of needle-in-a-haystack queries from Geographica2 benchmark. We observed that the overall query processing time of Semagrow is equal to the endpoint delay. Intuitively speaking, this means that if each endpoint "behaves" as if it contains TBs of data and processes queries in an acceptable time, then the federation "behaves' as if it federates PBs of data and processes queries in an acceptable time.

Bibliography

[1] Maribel Acosta, Maria-Esther Vidal, and York Sure-Vetter. Diefficiency metrics: Measuring the continuous efficiency of query processing approaches. In *Proceedings of the 16th International Semantic Web Conference (ISWC 2017)*. Springer, 2017.

[2] Andreas Brodt, Daniela Nicklas, and Bernhard Mitschang. Deep integration of spatial query processing into native RDF triple stores. In *18th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, ACM-GIS 2010, November 3-5, 2010, San Jose, CA, USA, Proceedings*, pages 33–42, 2010.

[3] Angelos Charalambidis, Antonis Troumpoukis, and Stasinos Konstantopoulos. SemaGrow: Optimizing federated SPARQL queries. In *Proceedings of the 11th International Conference on Semantic Systems (SEMANTiCS 2015), Vienna, Austria, 16–17 September 2015*, 2015.

[4] George Garbis, Kostis Kyzirakos, and Manolis Koubarakis. Geographica: A benchmark for geospatial RDF stores. In *Proceedings of the 12th International Semantic Web Conference (ISWC 2013). Sydney, Australia, 21-25 October 2013*, 2013.

[5] Olaf Görlitz and Steffen Staab. SPLENDID: SPARQL endpoint federation exploiting VOID descriptions. In *Proceedings of the 2nd International Workshop on Consuming Linked Data (COLD 2011), Bonn, Germany, October 23, 2011*, volume 782 of *CEUR Workshop Proceedings*, 2011.

[6] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. LUBM: a benchmark for OWL knowledge base systems. *Web Semantics*, 3(2), July 2005.

[7] Theofilos Ioannidis, George Garbis, Kostis Kyzirakos, Konstantina Bereta, and Manolis Koubarakis. Evaluating geospatial RDF stores using the benchmark geographica 2. *J. Data Semant.*, 10(3-4):189–228, 2021.

[8] Charalampos Kostopoulos, Giannis Mouchakis, Nefeli Prokopaki-Kostopoulou, Antonis Troumpoukis, Angelos Charalambidis, and Stasinos Konstantopoulos. KOBE: Cloud-native open benchmarking engine for federated query processors. Demonstration at the 19th International Semantic Web Conference (ISWC 2020), 2-6 November 2020, 2020.

[9] Charalampos Kostopoulos, Giannis Mouchakis, Antonis Troumpoukis, Nefeli Prokopaki-Kostopoulou, Angelos Charalambidis, and Stasinos Konstantopoulos. KOBE: cloud-native open benchmarking engine for federated query processors. In *Proceedings of ESWC 2021*, June 2021.

[10] 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.

[11] Kostis Kyzirakos, Dimitrianos Savva, Ioannis Vlachopoulos, Alexandros Vasileiou, Nikolaos Karalis, Manolis Koubarakis, and Stefan Manegold. GeoTriples: Transforming geospatial data into RDF graphs using R2RML and RML mappings. *J. Web Semant.*, 52-53:16–32, 2018.

[12] Axel-Cyrille Ngonga Ngomo and Michael Röder. HOBBIT: Holistic benchmarking for big linked data. In *Processings of the ESWC 2016 EU Networking Session*, 2016.

[13] Claudia Paris, Lorenzo Bruzzone, TorbjÃÿrn Eltoft, Thomas KrÃęmer, Andrea Marinoni, Salman Khaleghian, Corneliu Octavian Dumitru, and Mihai Datcu. Large training database. Technical Report Public Deliverable D2.1, ExtremeEarth Project, December 2019.

[14] Norman W. Paton, M. Howard Williams, Kosmas Dietrich, Olive Liew, Andrew Dinn, and Alan Patrick. VESPA: A Benchmark for Vector Spatial Databases. In *Advances in Databases, 17th British National Conference on Databases, BNCOD 17, Exeter, UK, July 3-5, 2000, Proceedings*, pages 81–101, 2000.

[15] Suprio Ray, Bogdan Simion, and Angela Demke Brown. Jackpine: A benchmark to evaluate spatial database performance. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, pages 1139–1150, 2011.

[16] Muhammad Saleem, Ali Hasnain, and Axel-Cyrille Ngonga Ngomo. Largerdfbench: A billion triples benchmark for SPARQL endpoint federation. *J. Web Semant.*, 48:85–125, 2018.

[17] Michael Schmidt, Olaf Görlitz, Peter Haase, Günter Ladwig, Andreas Schwarte, and Thanh Tran. FedBench: A benchmark suite for federated semantic data query processing. In *Proceedings of the 10th International Semantic Web Conference (ISWC 2011), Bonn, Germany, 23-27 October 2011*, volume 7031 of *Lecture Notes in Computer Science*. Springer, 2011.

[18] Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. FedX: A federation layer for distributed query processing on Linked Open Data. In *Proceedings of the 8th Extended Semantic Web Conference (ESWC 2011), Heraklion, Crete, Greece, May 29 – June 2, 2011*, volume 6644 of *Lecture Notes in Computer Science*, pages 481–486. Springer, 2011.

[19] Antonis Troumpoukis, Angelos Charalambidis, Giannis Mouchakis, Stasinos Konstantopoulos, Ronald Siebes, Victor de Boer, Stian Soiland-Reyes, and Daniela Digles. Developing a benchmark suite for Semantic Web data from existing workflows. In *Proceedings of the Benchmarking Linked Data Workshop (BLINK), held at the 15th International Semantic Web Conference (ISWC 2016), Kobe, Japan, 18 October 2016*, 2016.

[20] Antonis Troumpoukis, Stasinos Konstantopoulos, Giannis Mouchakis, Nefeli Prokopaki-Kostopoulou, Claudia Paris, Lorenzo Bruzzone, Despina-Athanasia Pantazi, and Manolis Koubarakis. GeoFedBench: A benchmark for federated GeoSPARQL query processors. In *Proceedings of the Posters and Demos Session of the 19th International Semantic Web Conference (ISWC 2020), 2-6 November 2020.*, 2020.

[21] Antonis Troumpoukis, Nefeli Prokopaki-Kostopoulou, Giannis Mouchakis, and Stasinos Konstantopoulos. Software for federating big linked geospatial data sources – version 2. Technical Report Public Deliverable D3.8, ExtremeEarth Project, June 2021.

[22] Ruben Verborgh, Miel Vander Sande, Olaf Hartig, Joachim Van Herwegen, Laurens De Vocht, Ben De Meester, Gerald Haesendonck, and Pieter Colpaert. Triple pattern fragments: A low-cost knowledge graph interface for the web. *Journal of Web Semantics*, 37–38, 2016.