Geographica: A Benchmark for Geospatial RDF Stores (Long Version) *

George Garbis, Kostis Kyzirakos, and Manolis Koubarakis

National and Kapodistrian University of Athens, Greece
{ggarbis,kk,koubarak}@di.uoa.gr

Abstract. Geospatial extensions of SPARQL like GeoSPARQL and stSPARQL have recently been defined and corresponding geospatial RDF stores have been implemented. However, there is no widely used benchmark for evaluating geospatial RDF stores which takes into account recent advances to the state of the art in this area. In this paper, we develop a benchmark, called Geographica, which uses both real-world and synthetic data to test the offered functionality and the performance of some prominent geospatial RDF stores.

Keywords: benchmarking, geospatial, RDF store, Linked Open Data, GeoSPARQL, stSPARQL

1 Introduction

The Web of data has recently started being populated with geospatial data and geospatial extensions of SPARQL, like GeoSPARQL and stSPARQL, have been defined. GeoSPARQL [12] is a recently proposed OGC standard for a SPARQL-based query language for geospatial data expressed in RDF. GeoSPARQL defines a vocabulary (classes, properties, and extension functions) that can be used in RDF graphs and SPARQL queries to represent and query geographic features with vector geometries. stSPARQL [9,1] is an extension of SPARQL 1.1 developed by our group for representing and querying geospatial data that change over time. Similarly to GeoSPARQL, the geospatial part of stSPARQL defines datatypes that can be used for representing in RDF the serializations of vector geometries encoded according to the widely adopted OGC standards Well Known Text (WKT) and Geography Markup Language (GML). stSPARQL and GeoSPARQL define extension functions from the OGC standard "OpenGIS Simple Feature Access" (OGC-SFA) that can be used by the users for manipulating vector geometries.

In parallel with the appearance of GeoSPARQL and stSPARQL, researchers have implemented geospatial RDF stores that support these SPARQL extensions (our own system Strabon[1], Parliament[2] and uSeekM[3]). Typically, this has

* This work was supported in part by the European Commission project TELEIOS (257662)

[1] http://strabon.di.uoa.gr/

[2] http://parliament.semwebcentral.org/

[3] http://dev.opensahara.com/projects/useekm/

been done by extending existing RDF stores that had no geospatial functionalities (e.g., Sesame) and by relying in state of the art spatially-enabled RDBMS (e.g., PostGIS) for the storage and querying of geometries. One reason that this approach has been successful is that the relational realization of the OGC-SFA standard has been widely adopted by many RDBMS for storing and manipulating vector geometries. The state of the art in this area is summarized in the survey paper [8].

The above advances to the state of the art in query languages and implemented systems has not so far been matched with much work on the evaluation and benchmarking of implemented geospatial RDF stores. Although there are various benchmarks for spatially-enabled RDBMS [17,13,4,14,15,11], there is only one paper in the literature that proposes a benchmark for geospatial data expressed in RDF [6]. However, since this work has preceded the proposal of GeoSPARQL and stSPARQL, it does not cover much of the features available in these languages. For example, only point and rectangle geometries are used in the data and only two topological functions and two non-topological functions are considered, while metric spatial functions and spatial aggregates are not discussed. Similarly, only the geospatial RDF store SPAUK, which is a precursor to Parliament, has been evaluated using the benchmark. Finally, [6] uses a synthetic workload only and does not consider any linked geospatial datasets such as the ones that are available in the LOD cloud today.

In this paper we go significantly beyond [6] and develop a benchmark, that can be used for the evaluation of the new generation of RDF stores supporting the query languages GeoSPARQL and stSPARQL. Our benchmark, nick-named Geographica[4], is composed by two workloads with their associated datasets and queries: a *real-world* workload based on publicly available linked data sets and a *synthetic* workload. The real-world workload uses publicly available linked geospatial data, covering a wide range of geometry types (e.g., points, lines, polygons). To define this workload, we follow the approach of the benchmark Jackpine [15] and we define a micro benchmark and a macro benchmark. The micro benchmark tests primitive spatial functions. We check the spatial component of a system with queries that use non-topological functions, spatial selections, spatial joins and spatial aggregate functions. In the macro benchmark we test the performance of the selected RDF stores in typical application scenarios like reverse geocoding, map search and browsing, and a real-world use case from the Earth Observation domain. In the second workload of Geographica we use a generator that produces synthetic datasets of various sizes and generates queries of varying thematic and spatial selectivity. In this way, we can perform the evaluation of geospatial RDF stores in a controlled environment. In this part we follow the rationale of earlier papers [14,9,3]. For reasons of reproducibility, both workloads are publicly available[5].

[4] Geographica (Greek: Γεωγραφικά) is a 17-volume encyclopedia of geographical knowledge written by the greek geographer, philosopher and historian Strabon (Greek: Στράβων) in 7 BC. (http://en.wikipedia.org/wiki/Geographica)

[5] http://geographica.di.uoa.gr/

We chose to test the systems Strabon, Parliament and uSeekM. To the best of our knowledge, these systems are the only ones that currently provide support for a rich subset of GeoSPARQL and stSPARQL. Other RDF stores like OpenLink Virtuoso, OWLIM and AllegroGraph, allow only the representation of point geometries and provide support for a few geospatial functions [8]. The limited functionality provided by these systems did not allow us to include them in the most experiments presented in this paper. A comparison between RDF stores with limited geospatial capabilities and geospatial RDF stores are given in Section 5.

The rest of the paper is organized as follows. Section 2 presents previous related work. The benchmark and its results are described in Sections 3 and 4, respectively and general conclusions and future work are discussed in Section 6.

2 Related Work

In this section we discuss the most important benchmarks that are relevant to Geographica. We first present well-known benchmarks for SPARQL query processing, then benchmarks from the area of spatial relational databases and, finally, the only available benchmark for querying linked geospatial data.

Benchmarks for SPARQL query processing. Four well-known benchmarks for SPARQL querying are the Lehigh University Benchmark (LUBM) [5], the Berlin SPARQL Benchmark (BSBM) [2], the SP^{2}Bench SPARQL Performance Benchmark [16] and the DBpedia SPARQL Benchmark (DBPSB) [10]. LUBM, BSBM and SP^{2}Bench create a synthetic dataset based on a use case scenario and define some queries covering a spectrum of SPARQL characteristics. For example, the synthetic dataset of SP^{2}Bench resembles the original publications dataset of DBLP while the dataset of LUBM describes the university domain. The creators of DBPSB take a different approach. They propose a benchmark creation methodology based on real-world data and query logs. The proposed methodology is used in [10] to create a benchmark based on DBpedia data and query-logs.

Benchmarks for spatial relational databases. One of the first benchmarks for spatial relational databases has been the SEQUOIA benchmark [17] which focuses on Earth Science use cases. In order for its results to be representative of Earth Sciences use cases, SEQUOIA uses real-world data (satellite raster data, point locations of geographic features, land use/land cover polygons and data about drainage networks covering the area of USA) and real-world queries. Its queries cover tasks like data loading, raster data management, filtering based on spatial and non-spatial criteria, spatial joins, and path computations over graphs. The SEQUOIA benchmark has been extended in [13] to evaluate the geospatial DBMS Paradise. Two other well known benchmarks for spatial relational databases which use synthetic vector data are Á La Carte [4] and VESPA[14]. Á La Carte uses a dataset consisting only of rectangles which are generated according to various statistical distributions and it has been used to compare the performance of different spatial join techniques. VESPA [14] creates a more com-

Datasets	Size	Triples	# of Points	# of Lines	# of Polygons
GAG	33MB	4K	-	-	325
CLC	401MB	630K	-	-	45K
LGD (only ways)	29MB	150K	-	12K	-
GeoNames	45MB	400K	22K	-	-
DBpedia	89MB	430K	8K	-	-
Hotspots	90MB	450K	-	-	37K

Table 1: Dataset characteristics

plex dataset with more geometry types (polygons, lines and points) and it has been used to compare PostgreSQL with Rock & Roll deductive object oriented database. More recently, [15] has defined a more generic benchmark for spatial relational databases, called Jackpine. It includes two kinds of benchmarking, micro and macro. Micro benchmarking tests topological predicates and spatial analysis functions in isolation. Macro benchmarking defines six typical spatial data applications scenarios and tests a number of queries based on them.

Benchmarks for geospatial RDF stores. The only published benchmark for querying geospatial data encoded in RDF has been proposed by Kolas [6]. He extends LUBM to include spatial entities and to test the functionality of spatially enabled RDF stores. LUBM queries are extended to cover four primary types of spatial queries, namely spatial location queries, spatial range queries, spatial join queries, nearest neighbor queries. Range queries aim to test cases of various selectivity, while spatial joins aims to test whether the query planner selects a good plan by taking into account the selectivity of the spatial and ontological part of each query.

3 The Benchmark Geographica

In this section we present our benchmark in detail. Section 3.1 presents its first part (the real-world workload) while Section 3.2 presents the second part (the synthetic workload).

3.1 Real-World Workload

This workload aims at evaluating the efficiency of basic spatial functions that a geospatial RDF store should offer. In addition, this workload includes three typical application scenarios.

Datasets. In this section we describe the datasets that we use for the real-world workload. We have datasets that play an important role in the Linked Open Data Cloud, such as the part of DBpedia and GeoNames referring to Greece, despite the fact that their spatial information is limited to points. In addition we have part of the LinkedGeoData[6] (LGD) dataset which has richer geospatial

[6] http://linkedgeodata.org/

information from OpenStreetMap[7] about the road network and rivers of Greece. We also chose to use the Greek Administrative Geography[8] (GAG) and the CORINE Land Use/Land Cover[9] (CLC) dataset for Greece which have complex polygons. The CLC dataset is made available by the European Environmental Agency for the whole Europe and contains data regarding the land cover of European countries. Both of these datasets with information about Greece have been published as linked data by us in the context of the European project TELEIOS[10]. Finally, we include a dataset containing polygons that represent wild fire hotspots. This dataset has been produced by the National Observatory of Athens (NOA) in the context of project TELEIOS by processing appropriate satellite images as described in [7]. Each dataset is loaded in a separate named graph so that each query access only the part of the dataset that is needed. Some important characteristics of the datasets used can be found in Table 1.

Micro Benchmark. The micro benchmark aims at testing the efficiency of primitive spatial functions in state of the art geospatial RDF stores. Thus, we use simple SPARQL queries which consist of one or two triple patterns and a spatial function. We start by checking simple spatial selections. Next, we test more complex operations such as spatial joins. We test spatial joins using the topological relations defined by stSPARQL [9] and the Geometry Topology component of GeoSPARQL.

Apart from topological relations, we test non-topological functions (e.g., `geof:buffer`), defined by the Geometry extension of GeoSPARQL, which construct a new geometry object. Additionally, we test the metric function `strdf:area` which is only defined in stSPARQL. The aggregate functions `strdf:extent`, and `strdf:union` of stSPARQL are also tested by this benchmark. GeoSPARQL does not define aggregate functions. We include aggregate functions in Geographica since they are present in all geospatial RDBMS, and we found them very useful in EO applications in the context of the project TELEIOS. A short description of queries used in the micro benchmark can be found in Table 2.

Macro Benchmark. In the macro benchmark we aim to test the performance of the selected RDF stores in the following typical application scenarios: reverse geocoding, map search and browsing, and two scenarios from the Earth Observation domain.

Reverse Geocoding. Reverse geocoding is the process of attributing a readable address or place name to a given point. Thus, in this scenario, we pose SPARQL queries which sort retrieved objects by their distance to the given point and select the first one.

Map Search and Browsing. This scenario demonstrates the queries that are typically used in Web-based mapping applications. A user first searches for points

[7] http://www.openstreetmap.org/
[8] http://www.linkedopendata.gr/dataset/greek-administrative-geography/
[9] http://www.linkedopendata.gr/dataset/corine-land-cover-of-greece
[10] http://www.earthobservatory.eu/

Query	Operation	Description
Non-topological construct functions		
Q1	Boundary	Construct the boundary of all polygons of CLC
Q2	Envelope	Construct the envelope of all polygons of CLC
Q3	Convex Hull	Construct the convex hull of all polygons of CLC
Q4	Buffer	Construct the buffer of all points of GeoNames
Q5	Buffer	Construct the buffer of all lines of LGD
Q6	Area	Compute the area of all polygons of CLC
Spatial selections		
Q7	Equals	Find all lines of LGD that are spatially equal with a given line
Q8	Equals	Find all polygons of GAG that are spatially equal a given polygon
Q9	Intersects	Find all lines of LGD that spatially intersect with a given polygon
Q10	Intersects	Find all polygons of GAG that spatially intersect with a given line
Q11	Overlaps	Find all polygons of GAG that spatially overlap with a given polygon
Q12	Crosses	Find all lines of LGD that spatially cross a given line
Q13	Within polygon	Find all points of GeoNames that are contained in a given polygon
Q14	Within buffer of a point	Find all points of GeoNames that are contained in the buffer of a given point
Q15	Near a point	Find all points of GeoNames that are within specific distance from a given point
Q16	Disjoint	Find all points of GeoNames that are spatially disjoint of a given polygon
Q17	Disjoint	Find all lines of LGD that are spatially disjoint of a given polygon
Spatial joins		
Q18	Equals	Find all points of GeoNames that are spatially equal with a point of DBpedia
Q19	Intersects	Find all points of GeoNames that spatially intersect a line of LGD
Q20	Intersects	Find all points of GeoNames that spatially intersect a polygon of GAG
Q21	Intersects	Find all lines of LGD that spatially intersect a polygon of GAG
Q22	Within	Find all points of GeoNames that are within a polygon of GAG
Q23	Within	Find all lines of LGD that are within a polygon of GAG
Q24	Within	Find all polygons of CLC that are within a polygon of GAG
Q25	Crosses	Find all lines of LGD that spatially cross a polygon of GAG
Q26	Touches	Find all polygons of GAG that spatially touch other polygons of GAG
Q27	Overlaps	Find all polygons of CLC that spatially overlap polygons of GAG
Aggregate functions		
Q28	Extension	Construct the extension of all polygons of GAG
Q29	Union	Construct the union of all polygons of GAG

Table 2: Queries of the micro benchmark

Query	Description
Reverse Geocoding	
RG1	Find the closest populated place (from GeoNames)
RG2	Find the closest street (from LGD)
Map Search and Browsing	
MSB1	Find the co-ordinates of a given POI based on thematic criteria (from GeoNames)
MSB2	Find roads in a given bounding box around these co-ordinates (from LGD)
MSB3	Find other POI in a given bounding box around these co-ordinates (from GeoNames)
Rapid Mapping for Fire Monitoring	
RM1	Find the land cover of areas inside a given bounding box (from CLC)
RM2	Find primary roads inside a given bounding box (from LGD)
RM3	Find detected hotspots inside a given bounding box (from Hotspots)
RM4	Find municipality boundaries inside a given bounding box (from GAG)
RM5	Find coniferous forests inside a given bounding box which are on fire (from CLC and Hotspots)
RM6	Find road segments inside a given bounding box which may be damaged by fire (from LGD and Hotspots)

Table 3: Queries of the macro benchmark

of interest based on thematic criteria. Then, she selects a specific point and information about the area around it is retrieved (e.g., POIs and roads).

Rapid Mapping for Wild Fire Monitoring. In this scenario we test queries which retrieve map layers for creating a map that can be used by decision makers tasked with the monitoring of wild fires. This application has been studied in detail in project TELEIOS [7] and the scenario covers its core querying needs. First, spatial selections are used to retrieve basic information of interest (e.g., roads, administrative areas etc.). Second more complex information can be derived using spatial joins and non-topological functions. For example, a user may be interested in the segment of roads that may be damaged by fire. We point out that this scenario is representative of many rapid mapping tasks encountered in Earth Observation applications. The queries of the macro benchmark can be found in Table 3.

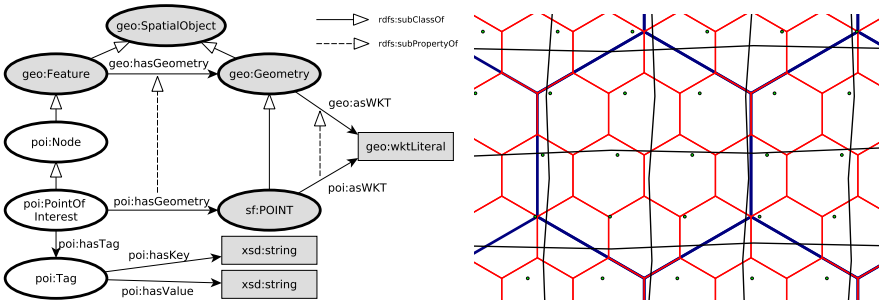
3.2 Synthetic Workload

The synthetic workload of Geographica relies on a generator that produces synthetic datasets of various sizes and instantiates query templates that can produce queries with varying thematic and spatial selectivity. In this way, we can perform the evaluation of geospatial RDF stores in a controlled environment in order to monitor their performance with great precision.

Datasets. The workload generator produces synthetic datasets of arbitrary size that resemble features on a map. As in VESPA [14], the produced datasets model the following geographic features: states in a country, land ownership, roads and points of interest. For each dataset, we developed a minimal ontology[11] that follows a general version of the schema of OpenStreetMap and uses GeoSPARQL ontologies and vocabularies. In Figure 1(a) we present the developed ontology for representing points of interest only. As in [3,9], 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 various thematic selectivities. For example, if we selected all points of interest tagged with key 1, we would select all available points of interest, if we selected all points of interest tagged with key 2, we would select half of them, etc.

Every feature has a spatial extent as well 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. By modifying the number of hexagons along

[11] http://geographica.di.uoa.gr/generator/{ontology, landOwnership, state, road, pointOfInterest}



(a) Ontology for Points of Interest

(b) Visualization of the geometric part of the synthetic dataset

Fig. 1: Synthetic dataset

an axis, we produce datasets of arbitrary size. As we will see in the following section, this enabled us to adjust the selectivity of the spatial predicates appearing in queries in a uniform way too.

As in [14], the generated land ownership dataset consists of n^2 features with hexagonal spatial extent, where each hexagon is placed uniformly on a $n \times n$ grid. The cardinality of the land ownerships is n^2. The generated state dataset consists of $(\frac{n}{3})^2$ features with hexagonal spatial extent, where each hexagon is placed uniformly on a $\frac{n}{3} \times \frac{n}{3}$ grid. The cardinality of the state geometries is $(\frac{n}{3})^2$. The generated road dataset consists of n features with sloping line geometries. Half of the line geometries are roughly horizontal and the other half are roughly vertical. Each line consists of $\frac{n}{2}+1$ line segments. The cardinality of the road geometries is n. The generated point of interest dataset consists of n^2 features with point geometries which are uniformly placed on n sloping, evenly spaced, parallel lines. The cardinality of the point of interest geometries is n^2. In Figure 1(b) we present a sample of the generated geometries.

Queries. The synthetic workload generator produces SPARQL queries corresponding to spatial selection and spatial joins by instantiating the two query templates presented in Table 4.

The query template used for producing SPARQL queries corresponding to spatial selections is identical to the query template used in [3,9]. In this query template, parameter `THEMA` is one of the values used when assigning tags to a feature and parameter `GEOM` is the WKT serialization of a rectangle. As in [9], we define the *thematic selectivity* of an instantiation of the query template as the fraction of the total features of a dataset that are tagged with a key equal to `THEMA`. For example, by altering the value of `THEMA` from 1 to 2, we reduce the thematic selectivity of the query by selecting half the nodes we previously did. We define the *spatial selectivity* of an instantiation of the query template as the fraction of the total features for which the topological relations defined

(a)

```
SELECT ?s
WHERE {
?s ns:hasGeometry/ns:asWKT ?g.
?s c:hasTag/ns:hasKey "THEMA".
FILTER(FUNCTION(?g, "GEOM"))}
```

(b)

```
SELECT ?s1 ?s2
WHERE {
?s1 ns1:hasGeometry/ns1:asWKT ?g1.
?s1 ns1:hasTag/ns1:hasKey "THEMA".
?s2 ns2:hasGeometry/ns2:asWKT ?g2.
?s2 ns2:hasTag/ns2:hasKey "THEMA'".
FILTER(FUNCTION(?g1, ?g2))}
```

Table 4: Query templates for generating SPARQL queries corresponding to (a) spatial selections, and (b) spatial joins.

by parameter `FUNCTION` holds between each of them and the rectangle defined by parameter `GEOM`. By modifying the value of the parameter namespace `ns` we specify the dataset and the corresponding type of geometric information that is examined by an instance of the query template.

The query template used for producing SPARQL queries corresponding to spatial joins involves two datasets identified by the values of the parameter namespaces `ns1` and `ns2`. In this query template as well, parameters `THEMA` and `THEMA'` control the thematic selectivity of the query. The value of parameter `FUNCTION` defines the topological relation that must hold between instances of the two datasets that are involved in an instance of the query template. Parameter FUNCTION can be instantiated with every function defined in the Geometry Topology extension of GeoSPARQL. In our experiments, as described in Section 4.3, we used `geof:sfIntersects`, `geof:sfTouches`, `geof:sfWithin`. For example, by instantiating the query template (b) with the values `poi` for `ns1`, `state` for `ns2`, `1` for `THEMA`, `2` for `THEMA'` and `geof:sfWithin` for `FUNCTION`, we get a SPARQL query that asks for all generated points of interest that are inside half of the generated states.

These query templates allow us to generate SPARQL queries with great diversity regarding their spatial and thematic selectivity, thus stressing the optimizers of the geospatial RDF stores that we test and evaluating their effectiveness in identifying efficient query plans.

4 Benchmark Results

In this section we present the results of running Geographica against various RDF stores. As we mentioned earlier, we chose to test the open source systems Strabon, uSeekM and Parliament that currently provide support for a rich subset of GeoSPARQL and stSPARQL. In order to investigate the trade-off between generic RDF stores that provide limited geospatial functionality and geospatial RDF stores we run the subset of Geographica that against Strabon, uSeekM, Parliament and OpenLink Virtuoso 7.

4.1 Experimental Setup

In this section we describe the setup of the experiments used to evaluate the selected triple stores. The machine that was used to run the benchmark is equipped

Workload	Strabon	uSeekM	Parliament
Real-world	220 sec.	214 sec.	250 sec.
Synthetic	221 sec.	406 sec.	462 sec.

Table 5: Storing times

Scenario	Strabon	uSeekM	Parliament
Reverse Geocoding	65s	0.77s	2.6s
Map Search and Browsing	0.9s	0.6s	22.2
Rapid Mapping for Fire Monitoring	207.4s	-	-

Table 6: Average Iteration times - Macro Scenarios

with two Intel Xeon E5620 processors with 12MB L3 cache running at 2.4 GHz, 24 GB of RAM and a RAID-5 disk array that consists of four disks. Each disk has 32 MB of cache and its rotational speed is 7200 rpm.

Each query in the micro and the synthetic benchmark was run three times on cold and warm caches. For warm caches, we ran each query once before measuring the response time, in order to warm up the caches. We measured the response time for each query posed by measuring the elapsed time from submitting the query until a complete iteration over the results had been completed. The response time of each query was measured and the median of all measurements is reported. A timeout of one hour is set as a time limit for all queries. For the macro benchmark, we run each scenario many times (with different initialization each time) for one hour without cleaning the caches and we report the average time for a complete execution of all queries defined in each scenario. Strabon and uSeekM utilize Postgres enhanced with PostGIS as a spatially-enabled relational backend. For these systems, we set up an instance of Postgres 9.2 with PostGIS 2.0 and we tuned it to make better use of the system resources.

For every dataset of Geographica, a unique property is used to connect geometries with their serialization (e.g. the Corine Land Use/Land cover ontology defines the property `clc:asWKT`), and this property is defined as a subproperty of the property `geo:asWKT` that is defined by GeoSPARQL. Parliament is able to identify and index a triple that represents the serialization of a geometric object only when the property `geo:asWKT` is used. As a result, the RDFS reasoning capabilities of Parliament have to be enabled so that it performs forward chaining during data loading and indexes the geometry using the spatial index as well. Strabon and uSeekM do not perform any reasoning on the input data.

4.2 Real-World Workload

Dataset Storage. In this section we present the time required by each system to store and index the datasets of the real-world workload. Strabon uses a storing scheme which creates a predicate table for every unique predicate of input dataset. Usually, this choice leads to the creation of a large number of predicate tables and consequently a lot of time is required for storing and indexing. The bulk loader of Strabon emulates this "per-predicate" scheme but it merges in a single table predicates that are used rarely on a dataset, so it reduces the required storing time. uSeekM needs less time since it is based on the native repository of Sesame which is known to be the most efficient implementation of Sesame for average sized datasets. Parliament is reasonably slower than uSeekM as it requires more time to perform forward chaining on the input dataset, as described in Section 4.1.

Micro Benchmark. The results of the micro benchmark are shown in Table 7 where the response time of each query is reported for both cold and warm caches.

First, the results of evaluating the queries with non-topological function are reported. For this class of queries uSeekM and Strabon have comparable response times while uSeekM is the fastest system. We observed that uSeekM does not utilize Postgres for evaluating these queries, but chooses to evaluate them using the native store of Sesame which is known to be more efficient, for small datasets, than Sesame implementations on top of a DBMS, like Strabon. Computing the area of polygons (Query 6) was tested only in uSeekM and Strabon since Parliament does not offer such functionality. We observe that none of the RDF stores highly exploits the warm caches when evaluating non-topological functions. This is because the non-topological functions used in this set of queries are computationally intensive (especially when complex geometries are used) and the time spent in the CPU dominates I/O time.

In the case of spatial selections, Strabon and uSeekM have comparable response times while Strabon is the fastest system most of the times. Both systems choose to start the query evaluation process by evaluating the spatial part of a query in PostGIS using the spatial index that is available. uSeekM continues by evaluating the rest of the query using the native store of Sesame. This adds a small overhead compared to Strabon which evaluates the whole query in PostgreSQL and utilizes a unified dictionary encoding scheme for both thematic and spatial information. On the contrary, the optimizer of Parliament does not take into consideration filters containing GeoSPARQL functions, so it evaluates the spatial predicate exhaustively over the results of the thematic part of the query. Queries 14 and 15 are semantically equivalent. Both ask for points that have a given distance from a given point. However, Query 14 creates the buffer of a given point with radius r and asks for points which are within this buffer, while Query 15 asks for points that have distance less than r from the given point. uSeekM and Parliament evaluate both queries by starting with the thematic part of the query and then they evaluate exhaustively the spatial operations without using the spatial index. The difference in the response time of queries 14 and 15 for these systems is due to the fact that calculating the distance between two points is much cheaper that evaluating the corresponding point-in-polygon operation. Strabon follows a similar plan for evaluating Query 15. However, for Query 14, Strabon calculates the buffer of the given polygon, and uses it to probe the spatial index for discovering points that lie inside the constructed polygon. This choice is correct since the response times remains constant.

In the case of spatial joins, uSeekM and Parliament are able to evaluate only queries 18 and 27 given the time limit of one hour. Parliament does not take into account GeoSPARQL extension functions during the optimization phase, resulting in query plans that evaluate separately the graph patterns corresponding to different graphs, compute the Cartesian product between them, and then apply the spatial predicate to the result of the Cartesian product. This strategy is very costly, thus Parliament is not able to answer most spatial joins given the time limit. uSeekM does not utilize PostGIS for evaluating spatial joins. Simi-

Type	Query	Cold caches (sec.)			Warm caches (sec.)		
		Strabon	uSeekM	Parliament	Strabon	uSeekM	Parliament
Non topological construct functions	Q1	42.33	38.11	152.71	41.36	36.25	132.67
	Q2	22.48	21.47	90.23	21.06	19.35	70.62
	Q3	29.48	27.06	98.56	27.73	24.13	79.40
	Q4	7.65	3.22	23.16	7.00	3.08	19.67
	Q5	14.68	4.17	21.63	13.78	5.00	19.58
	Q6	23.82	19.58	-	21.06	18.35	-
Spatial selections	Q7	0.36	1.22	2.42	0.01	0.02	1.36
	Q8	0.42	0.57	7.69	0.06	0.05	5.84
	Q9	0.83	1.27	35.03	0.16	0.05	34.09
	Q10	0.73	1.51	76.85	0.13	0.10	57.18
	Q11	2.66	2.96	195.87	2.03	1.29	175.98
	Q12	0.79	0.55	2.39	0.38	0.03	1.20
	Q13	0.82	0.89	63.14	0.13	0.04	59.60
	Q14	0.50	2.29	24.34	0.03	1.63	19.97
	Q15	0.50	0.99	3.44	0.12	0.30	0.56
	Q16	2.79	5.52	63.20	2.19	1.96	59.85
	Q17	3.06	1.60	35.89	2.62	0.86	34.39
Spatial joins	Q18	4.52	2233.73	2880.20	3.98	2504.24	2875.02
	Q19	1272.54	>1h	>1h	1284.62	>1h	>1h
	Q20	115.93	>1h	>1h	105.39	>1h	>1h
	Q21	113.26	>1h	>1h	107.76	>1h	>1h
	Q22	26.33	>1h	>1h	25.20	>1h	>1h
	Q23	26.29	>1h	>1h	25.01	>1h	>1h
	Q24	26.66	>1h	>1h	25.37	>1h	>1h
	Q25	342.87	>1h	>1h	341.04	>1h	>1h
	Q26	343.30	534.61	2040.00	341.28	534.15	2030.42
	Q27	343.72	>1h	>1h	342.06	>1h	>1h
Aggregate functions	Q28	3.56	-	-	2.92	-	-
	Q29	258.35	-	-	258.00	-	-

Table 7: Response times - Real Workload

larly to Parliament, it applies the spatial predicate to the result of the Cartesian product of the graph patterns. Strabon avoids evaluating Cartesian products by identifying graph patterns that are related only through the spatial predicate and pushes the evaluation of the spatial join in PostGIS, thus resulting in good response times. In all cases, warm caches do not affect the response time of the queries since a large number of intermediate results is produced. Finally, spatial aggregations are tested only in Strabon since it is the only system that supports such functions. We notice that Query 28 which computes the minimum bounding box that contains all geometries of the GAG dataset is much faster than Query 29 which computes the union of the same geometries since the former operation is much cheaper than the latter one which is computationally expensive.

Macro Benchmark. The results of the macro benchmark are shown in Table 6. In this table we report the average time needed for a complete iteration of all the queries of each scenario. The Reverse Geocoding scenario has two queries which use the function distance with a fixed limit. uSeekM performs the best in this scenario while Strabon needs an order of magnitude more time. The Map Search and Browsing scenario has one thematic query and two spatial selection queries. As described in Section 4.2 Strabon and uSeekm are efficient in evaluating spatial selections and they have good performace in this scenario as well. Finally, the Rapid Mapping for Fire Monitoring scenario is the most demanding scenario. It comprises three spatial selections queries, but also two complex queries which include spatial joins and construct new geometries (boundary and intersection). Only Strabon can serve this scenario since uSeekM and Parliament needed more than an hour to evaluate the query RM6. This happens because the query RM6

requires evaluating a demanding spatial join which is evaluated in a costy way by Parliament and uSeekM as described in previous paragraphs.

4.3 Synthetic Workload

Let us now discuss representative experiments that we run using a synthetic workload that was produced using the generator presented in Section 3. We generated a dataset by setting $n = 512$ and $k = 9$, where n is the number used for defining the cardinalities of the generated geometries, and k is the number used for defining the cardinalities of the generated tag values. This instantiation of the synthetic generator produces $262,144$ land ownership instances, $28,900$ states, 512 roads and $262,144$ points of interest. Each feature is tagged with key 1, every other feature with key 2, etc. up to key 512. The resulting dataset consists of 3,880,224 triples and its size is 745 MB.

Dataset Storage. Table 5 presents the time required by each system to store and index the synthetic dataset. The synthetic dataset has fewer predicates and more geometries that the real one. uSeekM requires more time than Strabon for storing the dataset, since it stores it in a Sesame native store and then it stores triples with geometric information at PostGIS as well. This overhead is significant compared to the total time required for storing the dataset, but leads to better response times in some cases. As we have already explained in Section 4.2, Parliament needs more time to store the synthetic dataset as well as the real-world dataset because it performs forward chaining on input data.

Queries. We instantiated the query template presented in Table 4(a) in order to produce SPARQL queries corresponding to spatial selections that ask for land ownerships that intersect a given rectangle, and points of interest that are within a given recangle. The given rectangle is generated in such as a way that the spatial predicate of the query holds for 1‰, 10%, 25%, 50%, 75% or all the features of the respective dataset. In addition, we instantiated the query template using the extreme values `1` and `512` of the parameter `THEMA` for selecting either all or approximatly 2‰ of the total features of a dataset. The response time of each system for evaluating the instantiations of this query template are presented in Figures 4(a)-4(h).

We instantiated the query template presented in Table 4(b) in order to produce SPARQL queries corresponding to spatial joins that ask for land ownerships that intersect a state, touching states and points of interest that are located inside a state. We also instantiated this query template using all combinations of the extreme values `1` and `512` for the parameters `THEMA` and `THEMA'`. The response time of each system for evaluating the instantiations of this query template are presented in Figures 4(i)-4(k).

By examining Figures 4(a)- 4(h), we observe that Strabon has very good performance overall. Strabon pushes the evaluation of a SPARQL query to the underlying spatially-enabled DBMS, which in this case is Postgres enhanced

with PostGIS. PostGIS has recently been enhanced with selectivity estimation capabilities. As a result, when a query selects only a few geometries, query evaluation always starts with the evaluation of the spatial predicate using the spatial index, thus resulting in few intermmediate results and good response times. While the spatial selectivity increases and more geometries satisfy the spatial predicate, the optimizer of Postgres chooses different query plans. For example, when the value of the parameter `THEMA` is 1 (Figures 4(a), 4(c), 4(e), 4(g)) and the value of the parameter `GEOM` is such that all geometries satisfy the spatial predicate, Postgres ignores the spatial index and performs a sequential scan on the table storing the geometries for evaluating the spatial predicate. Similarly, when the value of the parameter `THEMA` is 512 (Figures 4(b), 4(d), 4(f), 4(h)) and the value of the parameter `GEOM` is such that all geometries satisfy the spatial predicate, Postgres starts with the evaluation of the thematic selection that produces few intermediate results since only 2‰ of the features satisfy the thematic predicate, resulting in good query response times. In the case of spatial joins (Figures 4(i)- 4(k)), Strabon is the fastest system in most cases. The optimizer of Postgres takes into account the thematic selectivity of the queries and selects good query plans, thus Strabon is the only system that is able to answer the spatial joins given the one hour timeout when the parameters `THEMA` and `THEMA'` are equal to 1.

Regarding uSeekM, we observe that its performance is not affected by the thematic selectivity of the query. For spatial selections, uSeekM always start by evaluating the spatial predicate in PostGIS and then continues the query evaluation in the native Sesame store. As a result, regardless of the thematic selectivity, the response time of uSeekM increases while increasing the number of features with geometries that satisfy the given spatial predicate.

Regarding Parliament, we observe that its performance is not affected neither by the thematic nor by the spatial selectivity of a query. Parliament always starts by evaluating the non-spatial part of a query and then evaluates the thematic filter and the spatial predicate exhaustively on the intermediate results. Thus, the thematic and spatial selectivity of a query do not affect its response time.

In the case of spatial joins, uSeekM and Parliament produce the Cartesian product between the graph patterns that are joined through the spatial predicate, and evaluate the spatial predicate afterwards. This strategy is very costly, thus Parliament is not able to answer most spatial joins given the one hour timeout and uSeekM is more than two orders of magnitude slower than Strabon. However, in Figure 4(j) we observe that uSeekM outperforms Strabon. Strabon stores all geometries in a single table, so the evaluation of the spatial predicate *Touches* on this table returns not only the geometries of states that touch each other, but the touching geometries of land ownerships as well. The touching geometries of land ownerships are discarded later on, but this overhead proves to be more costly than producing a Cartesian product and evaluating the spatial predicate afterwards.

Workload	Strabon	uSeekM	Parliament	Virtuoso	System X
Real-world	65 sec.	45 sec.	36 sec.	9 sec.	8 sec.
Synthetic	496 sec.	771 sec.	874 sec.	63 sec.	31 sec.

Table 8: Storing times (Real world: GeoNames, DBpedia. Synthetic: PointsOfInterest)

5 Evaluating the Performance of RDF Stores with Limited Geospatial Capabilities

In this section we evaluate the performance of two RDF stores that provide limited geospatial capabilities and compare them with the geospatial RDF stores which are tested in the previous sections. For this purpose we use Virtuoso and a proprietary RDF store (which we will call System X). Virtuoso and System X support only point geometries. Thus, for each workload of Geographica, we keep the part of the datasets that contains only point geometries and we rerun only the queries which handle point geometries in the geospatial RDF stores Strabon, uSeekM and Parliament and the limited-function systems Virtuoso and System X.

Real-World Workload The real-world workload used in this case consists of only the corresponding datasets from DBpedia and GeoNames. Because Virtuoso and System X do not provide any non-topological function we tested only spatial selections and spatial joins. Also, the macro benchmark is not used in this section, since all scenarios use more complex geometry types than points.

Dataset Storage. The storing times for the real-world workload are presented in Table 8. For this subset of Geographica, we stored only the corresponding datasets of DBpedia and GeoNames. In this table we observe that the generic RDF stores Virtuoso and System X need considerably less time to store and index the real datasets than the geospatial RDF stores. Virtuoso and System X provide bulk loaders which are, also, able to exploit more than one processors. This leads to better storing times in comparison to storing times of uSeekM and Parliament, which use Sesame and Jena loading methods respectively, and in comparison to the bulk loader of Strabon. The bulk loader of Strabon processes input files in two steps. First, it transforms input RDF files into a series of CSV files which are later imported to PostgreSQL. These two steps add an overhead to Strabon storing times. Also, the bulk loader of Strabon consumes a lot of time (roughly $20\% - 30\%$ of the storing time) for gathering statistics.

Queries. Given the spatial selections that are supported by Virtuoso and System X, we have tested queries 14 and 15 (spatial selections) which, as described in Section 4.2, ask for points that have a given distance from a given point, but each query uses different function. We have also tested query 18 (spatial join) which asks for pairs of points of the datasets GeoNames and DBpedia that are equal. Virtuoso does not offer functions to create a buffer. Instead it provides the functions `bif:st_within`, `bif:st_intersects`, and `bif:st_contains`

```
SELECT ?s
WHERE { ?s ns:hasGeometry/ns:asWKT ?g.
        ?s c:hasTag/ns:hasKey "THEMA".
FILTER(FUNCTION(?g, bif:st_point(45, 45), "TOL"))}
```

Table 9: Query template for synthetic queries of Virtuoso

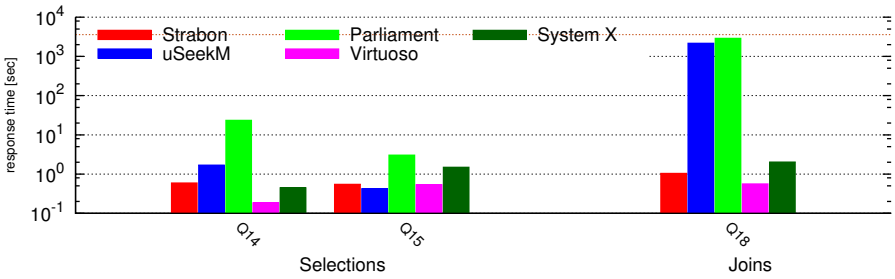
which receive a third argument that is a tollerance for the matching in units of linear distance. In order to emulate Query 15 in Virtuoso, instead of creating a buffer of the given point with radius r and asking for points inside the buffer, we can ask for points inside the given point with tollerance r.

The response times of these queries are reported in Figure 2. Query 14 has a complex filter clause (point inside a point buffer) but if the point buffer is computed the spatial index can be used to evaluate the query. Thus, Virtuoso, System X, and Strabon, which use their spatial index answer this query faster than uSeekM and Parliament which do not perform and indexed search. Especially Virtuoso, which is not burdened with the cost of evaluating a topological relation over complex geometries (like a buffer of a point), achieves the best response time. Query 15 does not favor the use of spatial index but its filter clause (distance between points) is evaluated easier than the filter clause of Query 14 (point inside point buffer). Thus, uSeekM and Parliament need less time to answer this query. Especially uSeekM needs the less time of all systems. Virtuoso and System X need slightly more time to answer Query 15, because they do not use their spatial indexes. In the case of spatial join (Query 18) Virtuoso has the less response time while Strabon answers second. Both systems are based on an R-tree to evaluate spatial queries.

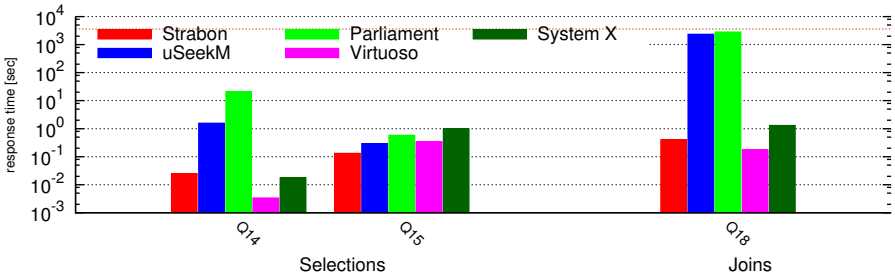
Synthetic Workload Regarding the synthetic workload we used the generator, which is described in Section 3.2, to generate a dataset and we stored only the generated points of interest. Because we use only points of interest we chose to set the generator parameters $n = 1024$ and $k = 10$ and generate a bigger dataset that contains about seven million triples.

Dataset Storage. The corresponding storing times are shown in Table 8. As well as for the real-world workload, Virtuoso and System X need less time to store the POI of the synthetic workload than the full geospatial RDF stores.

Queries. For this subset of Geographica we selected to run only spatial selections using topological relation `geof:sfWithin`. Since functions in Virtuoso cannot receive a rectangle as argument, the respective queries that were run by Virtuoso are produced by instantiating the template in Table 9. The parameter `TOL` is the tolerance value that will be used by Virtuoso for evaluating the topological relation defined by parameter `FUNCTION`. So a circle is considered by `bif:st_within` and the radius of the circle (defined by the parameter TOL) is instantiated to achieve each time the proper spatial selectivity.



(a) Cold caches

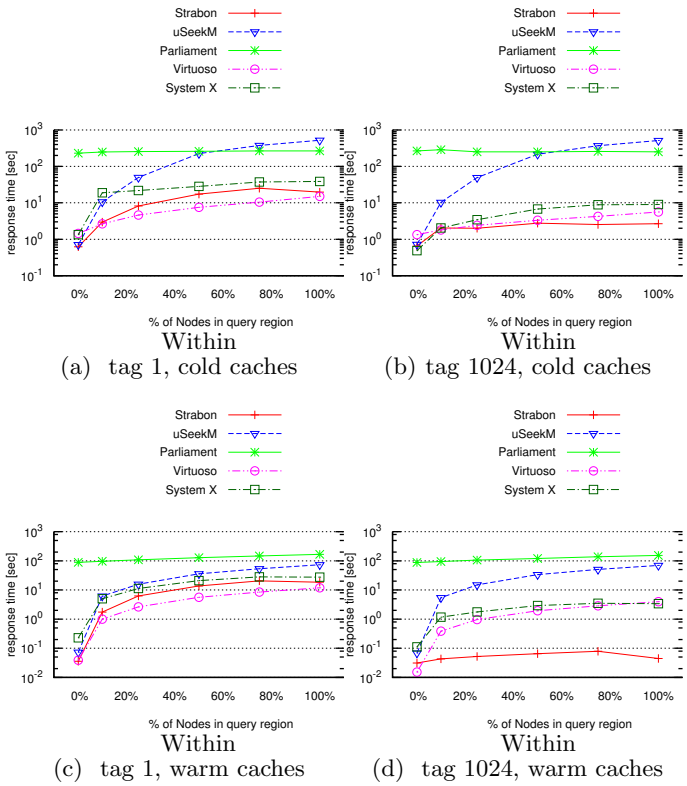


(b) Warm caches

Fig. 2: Response times, real-world workload

The response times for these queries are presented in Figure 3. In most cases Virtuoso is the fastest system while Strabon comes second. Both System X and Virtuoso evaluated all queries by starting with the spatial part of the query and then continuing with the thematic part and this is why their performance is affected more by the spatial selectivity of the query than by the thematic. For example, when the value of the parameter THEMA is 1 (Figures 3(a), 3(c)) Virtuoso needs the less time to evaluate the spatial selections. But when the value of the parameter THEMA is 1024 (Figures 3(b), 3(d)) Virtuoso does not exploit the fact that few points satisfy the thematic part of the query and its response time increases while the spatial selectivity is increased and more points satisfy the spatial predicate. Thus, Strabon, which changes its evaluation plans if the spatial selectivity is increased, answers such kind of queries faster.

In this section, we compared the performance of RDF systems that implement GeoSPARQL with general purpose RDF systems which provide limited spatial functionalities. Regarding storing data, Virtuoso and System X provide the best bulk loading capabilities. Regarding query evaluation Virtuoso has the best performance and Strabon has the second best performance close to the performance of Virtuoso. Finally, the query optimizers of Virtuoso and System X do not take into account the selectivity of a spatial predicate. However, this does not lead to bad performance because of their fast search mechanisms, even if the thematic selectivity of a query is greater than the spatial.



(a) tag 1, cold caches (b) tag 1024, cold caches

(c) tag 1, warm caches (d) tag 1024, warm caches

Fig. 3: Response times - Synthetic Workload

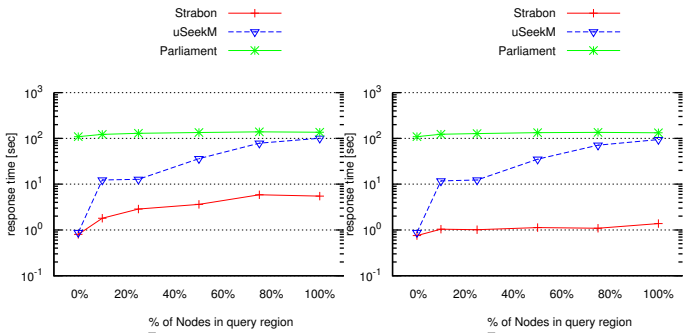
6 Conclusions

We presented a benchmark for evaluating the performance of geospatial RDF stores. We defined two workloads that test on the one hand the performance of the spatial component of such systems in isolation, and on the other hand test whether spatial query processing is deeply integrated in their query engines. Future work concentrates on extending the benchmark to capture the complete GeoSPARQL standard, publish larger real-world datasets and synthetic datasets that are not uniformly distributed, and evaluate them on centralized and distributed geospatial RDF stores that are beginning to emerge.

References

1. Bereta, K., Smeros, P., Koubarakis, M.: Representing and Querying the valid time of triples for Linked Geospatial Data. In: ESWC (2013)
2. Bizer, C., Schultz, A.: The Berlin SPARQL Benchmark. In: IJSWIS. vol. 5 (2009)
3. Brodt, A., Nicklas, D., Mitschang, B.: Deep Integration of Spatial Query Processing into Native RDF Triple Stores. In: SIGSPATIAL (2010)
4. Gunther, O., Picouet, P., Saglio, J.M., Scholl, M., Oria, V.: Benchmarking Spatial Joins À La Carte. In: IJGIS. vol. 13 (2007)
5. Guo, Y., Pan, Z., Heflin, J.: LUBM: A Benchmark for OWL Knowledge Base Systems. In: Web Semantics. vol. 3 (2005)

6. Kolas, D.: A Benchmark for Spatial Semantic Web Systems. In: International Workshop on Scalable Semantic Web Knowledge Base Systems (2008)
7. Koubarakis, M., Kontoes, C., Manegold, S., Karpathiotakis, M., Kyzirakos, K., Bereta, K., Garbis, G., Nikolaou, C., Michail, D., Papoutsis, I., Herekakis, T., Ivanova, M., Zhang, Y., Pirk, H., Kersten, M., Dogani, K., Giannakopoulou, S., Smeros, P.: Real-Time Wildfire Monitoring Using Scientific Database and Linked Data Technologies. In: EDBT (2013)
8. Koubarakis, M., Karpathiotakis, M., Kyzirakos, K., Nikolaou, C., Sioutis, M.: Data Models and Query Languages for Linked Geospatial Data. In: Reasoning Web. Semantic Technologies for Advanced Query Answering. Springer (2012)
9. Kyzirakos, K., Karpathiotakis, M., Koubarakis, M.: Strabon: A Semantic Geospatial DBMS. In: ISWC (2012)
10. Morsey, M., Lehmann, J., Auer, S., Ngomo, A.C.N.: DBpedia SPARQL Benchmark - Performance Assessment with Real Queries on Real Data. In: ISWC (2011)
11. Myllymaki, J., Kaufman, J.H.: DynaMark: A Benchmark for Dynamic Spatial Indexing. In: Mobile Data Management. vol. 2574 (2003)
12. Open Geospatial Consortium: OGC GeoSPARQL - A geographic query language for RDF data. OGC Implementation Standard (2012)
13. Patel, J., Yu, J., Kabra, N., Tufte, K., Nag, B., Burger, J., Hall, N., Ramasamy, K., Lueder, R., Ellmann, C., Kupsch, J., Guo, S., Larson, J., De Witt, D., Naughton, J.: Building a Scaleable Geo-Spatial DBMS: Technology, Implementation, and Evaluation. In: ACM SIGMOD (1997)
14. Paton, N.W., Williams, M.H., Dietrich, K., Liew, O., Dinn, A., Patrick, A.: VESPA: A Benchmark for Vector Spatial Databases. In: BNCOD (2000)
15. Ray, S., Simion, B., Demke Brown, A.: Jackpine: A Benchmark to Evaluate Spatial Database Performance. In: ICDE (2011)
16. Schmidt, M., Hornung, T., Lausen, G., Pinkel, C.: SP2Bench: A SPARQL Performance Benchmark. In: ICDE. pp. 222–233 (2009)
17. Stonebraker, M., Frew, J., Gardels, K., Meredith, J.: The SEQUOIA 2000 Storage Benchmark. In: ACM SIGMOD (1993)

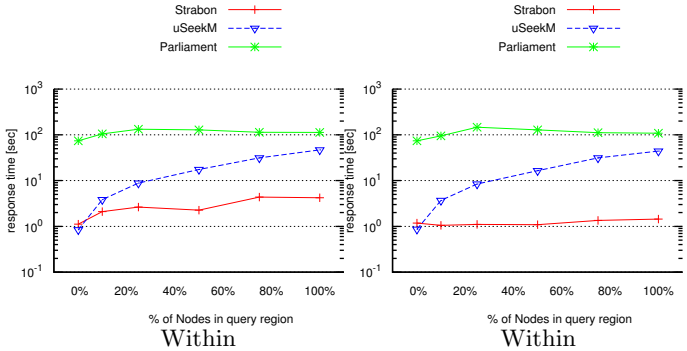


Intersects

(a) tag 1, cold caches

Intersects

(b) tag 512, cold caches

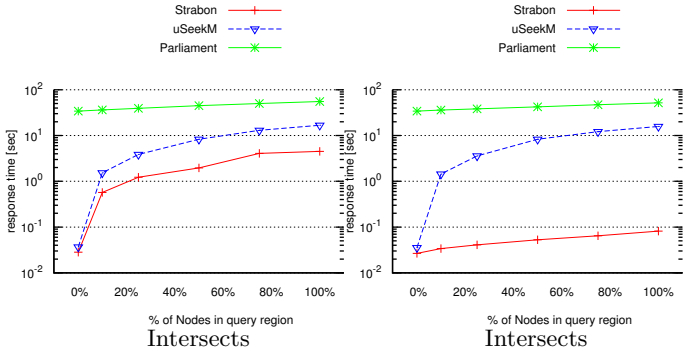


Within

(c) tag 1, cold caches

Within

(d) tag 512, cold caches

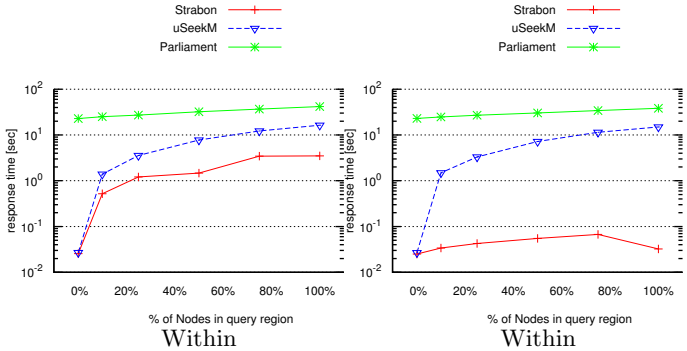


Intersects

(e) tag 1, warm caches

Intersects

(f) tag 512, warm caches

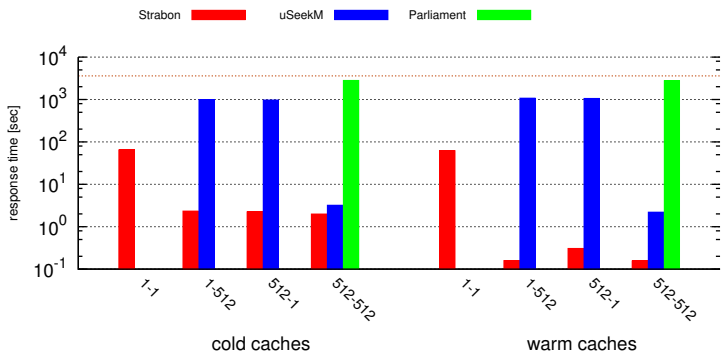


Within

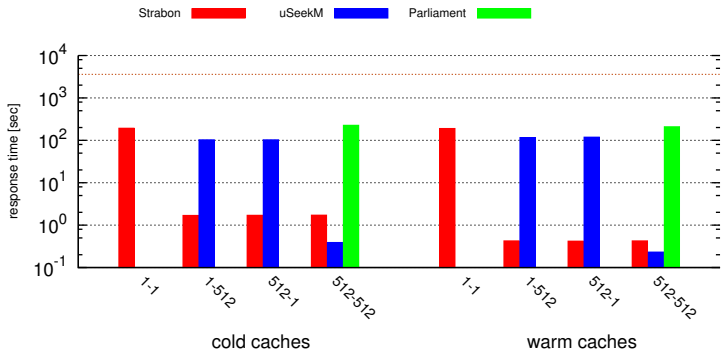
(g) tag 1, warm caches

Within

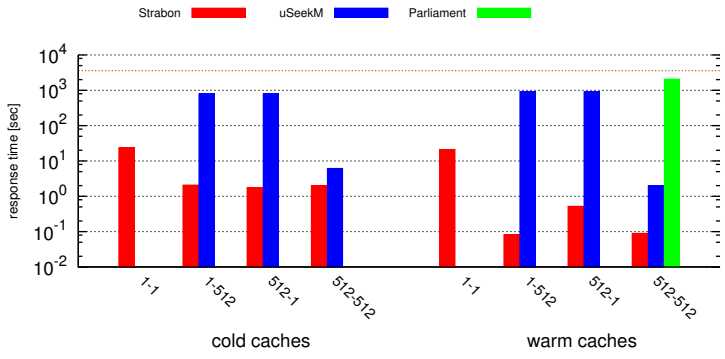
(h) tag 512, warm caches



(i) Intersects



(j) Touches



(k) Within

Fig. 4: Response times - Synthetic Workload