GeoSPARQL-Jena: Implementation and Benchmarking of a GeoSPARQL Graphstore

Taha Osman and Gregory Albiston
Department of Computer Science, Nottingham Trent University
taha.osamn@ntu.ac.uk
gregory.albiston@ntu.ac.uk

Abstract: This work presents an RDF graphstore implementation for all six modules of the GeoSPARQL standard using the Apache Jena Semantic Web library. Previous implementations have provided only partial coverage of the GeoSPARQL standard. There is discussion of the design and development of on-demand indexes to improve query performance without incurring lengthy data preparation delays. A supporting benchmarking framework is also discussed for the evaluation of any SPARQL compliant queries with interfaces provided for integrating additional test systems. This benchmarking framework is utilised to examine the performance of the implementation against two existing GeoSPARQL systems using the Geographica benchmark. It is found that the implementation achieves comparable or faster query responses than the alternative systems while also providing much faster dataset loading and initialisation durations.

Keywords: GeoSPARQL, , RDF, Apache Jena, geospatial, geospatial data, geospatial query language

1. Introduction

This work discusses the implementation of a geospatial graphstore complying with the OGC GeoSPARQL standard (Battle and Kolas, 2011). Previous works have partially implemented the GeoSPARQL standard and have generally extended relational databases rather than utilising an RDF graphstore. This work describes the design features of the implementation, investigation of parameter tuning and the development of a benchmarking framework for comparison with other GeoSPARQL implementations.

The increasing usage of location-aware devices such as smartphones, IoT devices and in-vehicle navigation, has led to an increasing range of applications and datasets reliant upon geospatial data. The interpretation of geospatial data and relations is an established field of study upon which the GeoSPARQL incorporates Semantic Web concepts and principles. The Semantic Web is intended to increase the access and sharing of data through the internet and presents the opportunity to enrich data through knowledge modelling, automated inferencing, and interconnection of datasets to provide deeper insights and experiences.

A key technology of the Semantic Web is the Resource Description Framework (RDF) (Schreiber and Raimond, 2014) which is a data structure standard based upon a directed labelled graph of subject-predicate-object triples. It is intended that any data can be represented using RDF's design principles and extended by vocabularies, e.g. GeoSPARQL, to allow consistent interpretation across datasets. The SPARQL query language (Harris, Seaborne and Prud'hommeaux, 2013) provides a query mechanism to explore, retrieve and modify RDF data. The GeoSPARQL standard extends SPARQL's query mechanisms and provides a vocabulary to allow users to conveniently access geospatial data.

There are several challenges relating to geospatial graphstores. There is no implementation of the GeoSPARQL standard stating full compliance across all six components. In addition, there is no benchmarking framework to investigate GeoSPARQL compliance and consider user quality of life aspects, such as mixed coordinate reference system datasets and inference of inferring geometry metadata.

The reviewed GeoSPARQL implementations use modified relational databases for persistent storage rather than a specialised RDF graphstore, which can present a set-up and environment configuration burden that is justifiable in production scenarios but limiting for prototype development and research. Therefore, the chief research question to be addressed in this paper is feasibility of a comprehensive implementation of the GeoSPARQL standard utilising a specialised RDF graphstore to optimise the performance of geospatial indexing and retrieval. The principal contributions of this work addressing the research challenge can be summarised as follows:

1. Full GeoSPARQL implementation of all six components, including automatic inferring geometry properties and query re-writing, using an RDF graphstore.
2. Benchmarking framework for GeoSPARQL and SPARQL queries with results and comparison of three GeoSPARQL systems.
3. Identification of areas for review and revision in future versions of the GeoSPARQL standard.

The remainder of this work is organised as follows. Section 2 overviews related works and section 3 discusses our implementation of the GeoSPARQL extension details the development of the corresponding Benchmarking Framework. Section 4 evaluates the results obtained upon comparing our GeoSPARQL implementation with two existing GeoSPARQL systems, Parliament and Strabon, using the developed benchmarking framework. The final section provides a summary of the findings of this work and identifies areas for future work.

2. Related Work

There are numerous implementations of RDF frameworks with many featuring geospatial and sometimes more specifically GeoSPARQL support. These RDF frameworks provide persistent storage through RDF graphstores, or relational databases adapted for RDF with a geospatial extension, e.g. PostgreSQL with PostGIS. While these frameworks refer to supporting GeoSPARQL it has not been possible to find any explicit statements of the implemented GeoSPARQL extensions and requirements compliance. As far as we are aware, there has not been any systematic research or compliance testing of the various RDF frameworks in their support for GeoSPARQL.

The following reviewed frameworks to illustrate the fragmented nature of GeoSPARQL support and different implementation approaches. AllegroGraph is a graph database that supports geospatial operations but takes a more generalised approach than the Simple Features or GeoSPARQL standards, with a set of functions focused upon points within a radius or polygons. Apache Jena (Apache Software Foundation, 2020) is a Java based Semantic Web and Linked Data framework that provides various tools for developing applications including persistent storage, SPARQL query engine, and inferencing. It has a spatial module that supports a small set of functions which are unrelated to the Simple Features or GeoSPARQL standards. Apache Marmotta (Apache Software Foundation, 2018) is a Linked Data platform which uses PostgreSQL with PostGIS backend and provides functions from the *Geometry Extension* and *Geometry Topology Extension*, but geospatial support has not been included in any public release due to "portability issues".

Stardog is an RDF data unification platform that supports a subset of non-topological GeoSPARQL functions along with their own geospatial functions. Virtuoso is SQL database with RDF and SPARQL extensions that includes some geospatial support related to the Simple Features standard but not explicitly supporting GeoSPARQL. Parliament integrates the Apache Jena framework with a Berkley DB persistent storage to provide temporal and GeoSPARQL compliant spatial support. However, the Parliament is dependent on deprecated components of Apache Jena and has not been updated to the latest major version. Strabon (Kyzirakos, Karpathiotakis and Koubarakis, 2012) is a spatiotemporal store that uses the RDF4J framework backed by PostgreSQL with PostGIS to support the *Core*, *Geometry Extension* and *Geometry Topology Extension* components of the GeoSPARQL and its own stSPARQL syntax.

3. Implementation of the GeoSPARQL-Jena and Benchmarking Framework

The GeoSPARQL standard is developed from the Simple Feature standard, which is widely used in SQL databases, but provides a wider definition of spatial relations and incorporates RDF concepts, such as URIs, namespaces and RDFS inferencing. The Simple Features standard is based upon a simplified spatial model where all coordinates are on a planar surface regardless of the Spatial Reference Systems (SRS) (Spatial Reference System, 2013), which manages the consistent interpretation of spatial coordinate information.

This section describes the design and implementation considerations of the GeoSPARQL-Jena system and the supporting benchmarking framework.

3.1 Implementing the GeoSPARQL Standard as an extension of Apache Jena

The primary focus of implementing the GeoSPARQL standard is to extend ARQ, the query execution mechanism of Apache Jena, to support all the extension modules. Filter and Property functions are registered with the ARQ query engine for use when matching function or property names are encountered within a SPARQL query.

The implementation was designed to minimise user configuration requirements and minimise pre-computation through on-demand processing of queried data. This on-demand processing means that initialisation periods are short, only calculations relevant to utilised functionality are performed, and new datasets can be incorporated quickly and with consistent query durations.

Implementing the GeoSPARQL Standard as an extension of Apache Jena allows all the existing functionality of Apache Jena, which closely complies with Semantic Web standards, to be utilised, e.g. RDF file handling, RDFS inferencing, HTTP endpoint and persistent storage. The GeoSPARQL class and property hierarchy are achieved by loading the published GeoSPARQL RDF schema into an RDFS inferencing model and applying the model to the target dataset. This allows user and version modifications to the schema to be made without requiring alteration to the implementation source code. Various configuration options for the use of the GeoSPARQL-Jena extension have been made available to the developer, such as index sizing and in-memory or persistent storage options.

The implementation workflow, shown in Figure 1, is activated when the ARQ query engine reaches a registered Filter or Property function name. The following describes the workflow with relevant diagram points indicated by parenthesis. The ARQ engine processes query by checking triples in the dataset and passing values to the corresponding function (1). A key concept of SPARQL querying is the reduction of the full dataset graph down to a subset which are true for the graph described by the query. The ARQ engine seeks to optimise execution for the quickest resolution. Therefore, a set of partial results for the wider query may have been obtained prior to passing to the function.

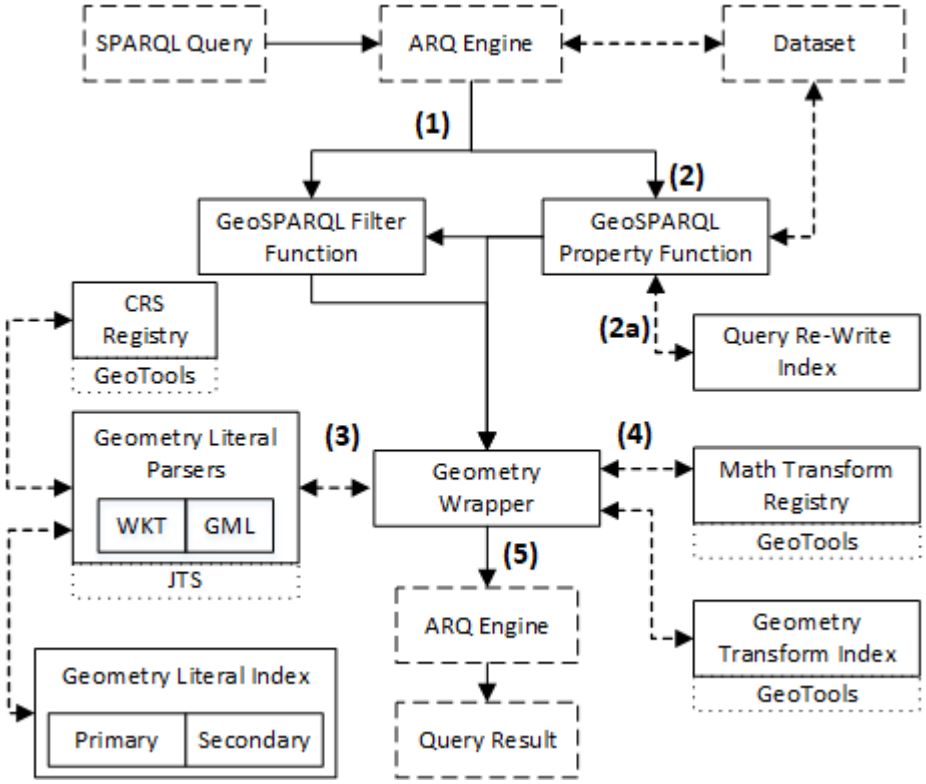

Figure 1: The GeoSPARQL-Jena query execution workflow from SPARQL query to result

GeoSPARQL Property Functions may retrieve additional values from the dataset (2). Many Property Functions in GeoSPARQL are syntactic sugar for a Filter Function which is used to perform the actual processing. The distinct Property Functions follow the same process as Filter Functions. The Query Re-Write Property Functions also checks for asserted Feature and Geometry relationships in the dataset to shortcut resolution, as required by the GeoSPARQL standard, followed by an additional check of the Query Re-Write Index (2a).

This index contains up to a maximum number of recently found relations between Geometry Literals. Each Geometry Literal pair could be involved in up to four Feature and Geometry relationships for each spatial relation, without considering that a single Geometry Literal could be used by multiple Features or Geometries.

Therefore, retaining recent results has been designed into the workflow to address the re-working behaviour of SPARQL query execution.

Geometry Literals are unparsed into Geometry Wrappers via relevant parsers (3). The Geometry Literal Index is checked for recently unparsed Geometry Literals to allow reuse between iterations through on-demand caching. Geometry Literals are immutable and so later extraction yields the same Geometry Wrapper. The Geometry Literal Index contains two indexes that relate to positions of function arguments which is likely to be consistent between query iterations and allows prioritising the index search. If the Geometry Literal is not present in the initial index, then the alternative index is checked before a new Geometry Wrapper is extracted from the Geometry Literal and added to the initial index. The CRS Registry similarly retains recently used CRS data for any future Geometry Literals with the same CRS.

The Geometry Wrapper handles all processing to resolve the request, e.g. spatial relations and functions, with one instance for each Geometry Literal (4). The spatial relations and functions leverage the JTS library (Eclipse Foundation, 2018), which is an implementation of the Simple Features standard, and has been extended to incorporate the additional concepts of the GeoSPARQL standard including multiple spatial reference systems, multiple units of measure, RCC8 and Egenhofer relation families (Egenhofer and Franzosa, 1991).

The functionality for CRS conversion is provided by the Apache SIS library (The Apache Software Foundation, 2019) which accesses coordinate transformations definitions provided by defining authorities. The outcome of the function is returned to the ARQ Engine (5) via the calling function to continue processing the query, potentially returning to step (1) for another function.

The retention of data in-memory can present issues when processing large datasets. To manage the sizes of the indexes and registries several strategies have been implemented. Firstly, the data retained within registries is common across multiple geometries and highly re-usable. Therefore, these have been not constrained in size or persistence and are intended to persist for the duration of the application. Secondly, to manage size the index contents expire after a fixed time-period (default: 5 seconds) which is refreshed whenever an item is retrieved. Checking for removals occurs in 1 second intervals. Therefore, memory usage will increase during periods of geospatial querying and then fall back during other activities.

3.2 Implementing the Benchmarking Framework

Similar to the common with the Geographica's approach, we implemented a Test System to integrate with the benchmarking framework (Figure **2**). However, in our approach new instances of the Test System are produced using a Test System Factory interface to create separate instances. The initialisation period of these instances is recorded as a separate duration from dataset loading and query execution. In contrast, in Geographica, the initiation duration is measured when a system is initially accessed via the Java API, which may be once, during start-up for production usage, or multiple times, such as during application execution or development.

The execution of queries is performed through a separate Query Task class, varying from Geographica, that is extended by each Test System to standardise the three stages of query execution, preparation, processing and closure, and recording of durations. This execution standardisation means that all target systems perform the same work and are monitored consistently. The Query Task is executed on a separate thread and is monitored for its duration to limit excessively long query executions.

The execution of the framework can be performed in four modes: Cold, Warm, Both and Dataload. In Cold mode, each iteration is a new instance of the Test System. The Warm mode, an initialising query is performed on a single Test System which is then re-used for the target iterations. The Both mode allows the Cold and Warm modes to be run consecutively. In the Dataload mode, a specified dataset is loading into the Test System's persistent storage for the target iterations with the storage being cleared after each iteration. These modes follow the principles described in the Geographica benchmarking paper (Garbis, Kyzirakos and Koubarakis, 2013).

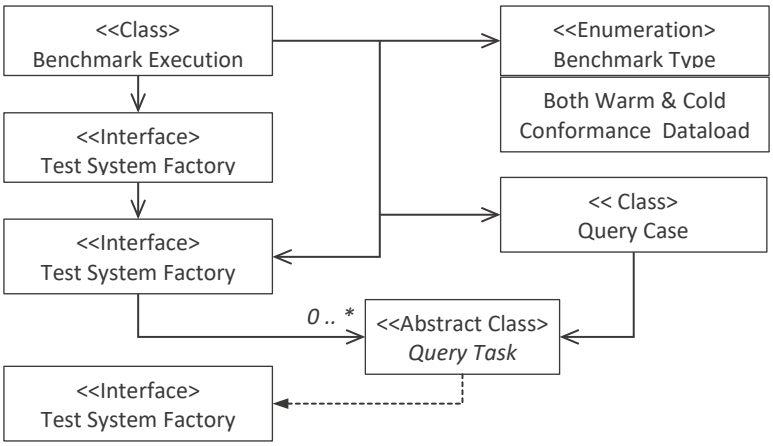

Figure 2: Class diagram of the main Benchmarking Framework classes and interfaces

4. Experimental Results and Discussion

This section outlines the evaluation, results, and analysis of this work's implementation of GeoSPARQL Jena against two existing GeoSPARQL systems, Parliament and Strabon. The first set of experiments analyses the effect of varying the size of indexes implemented in GeoSPARQL Jena. This is followed by an overview of the methodology and configuration of the benchmarking framework. The final sections discuss the results obtained for the three systems using the framework.

4.1 Experimentation configuration and methodology

All benchmarking experiments were performed on the same desktop computer which was a x64 Windows 10 operating system with Intel Core i7-4820K with 16GB and Samsung 850 EVO 500GB. Previously outlined is that all three test systems provide a Java API. Identical Java JVM settings were used for each target system with a maximum heap size of 3GB and parallel garbage collection enabled.

The framework measures the duration of each query execution with three values: query preparation, query processing and their sum for the total duration. For brevity the total durations are reported here, unless specified. A general observation is that the Apache Jena based test systems, Parliament and GeoSPARQLJena, perform more work during iteration over the results in the query processing stage while Strabon spent relatively longer periods in the query preparation stage. This highlights the different strategies deployed by the target systems and the importance of considering the whole execution duration of queries.

4.2 Dataset Loading and Initialisation

The six datasets published by Geographica have been used for the Microbenchmark and Macrobenchmark.

The target systems use their own persistent storage method with all supporting multiple graphs. Each dataset was loaded into their own graph within the storage. In the implementation the Apache Jena Java API loading mechanism was used and not the Bulk Loader for large datasets, which may provide faster results. Table 1. shows the data loading and initialisation durations for the target systems.

The loading durations include both the import of triples and any associated one-time spatial indexing setup. The initialisation duration is the duration required to initially access the dataset through the Java API. Therefore, in a production environment this may occur once but in an iterative development environment will be incurred at every execution. In all cases the operations were performed five times with arithmetic mean and standard deviations shown.

Table 1: Dataset loading and initialisation durations

	Loading		Initialisation	
Test System	mean (s)	sd	mean (s)	sd
GeoSPARQL Jena	90.694	7.401	0.047	0.003
Parliament	337.976	4.217	0.539	0.041
Strabon	374.439	6.194	80.317	1.882

The table demonstrates that the implementation, GeoSPARQL Jena, is noticeably quicker at both loading, 1.5 minutes compared to over 5 minutes, and initialising the datasets. This makes it very suited to a development environment when coupled with its minimal setup requirements. Parliament has a more prolonged setting up period when it is building its spatial indexes but is then quick to access prior to querying. Strabon takes longer to o load the datasets and requires an initialisation period of 80 seconds. This is incurred prior to any query execution and regardless of its data requirements. The loading overhead can be taxing if the application requires frequent restarts. The following sections discuss the query performance excluding the identified initialization periods.

4.3 Geographica Microbenchmark

The queries executed in this benchmark are those published by the Geographica project and follow the same numbering system. These utilise the non-topological query functions of the Geometry Extension and the filter functions of the Geometry Topology Extension for the Simple Feature relation family and therefore are a subset of potential queries for the GeoSPARQL standard. It was necessary to fix several namespaces in the published queries, but the body of the queries were unchanged. The queries Q6, Q28 and Q29 have been excluded as they contained spatial functions which are not defined in the GeoSPARQL standard and therefore only Strabon could execute.

The results of the microbenchmark are shown for Cold runs (Figure 3), Warm runs (Figure 4) and combined rankings (Table 2). Parliament did not complete any of the queries in the Spatial Joins section within the one-hour time limit. Results are shown in all cases for GeoSPARQL Jena and Strabon but in several cases results were achieved in less than a second. In all cases, except Q15, the number of results returned by all three systems were identical. Q15 uses the distance function to identify geometries within a radius of a fixed point. The difference in results may be attributable to precision of calculations and different approaches to handling the conversion of distance units in query. GeoSPARQL Jena is using the JTS library for calculations which does not reduce precision when other geospatial libraries may do so.

The query uses linear *metre* units for distance between two geometries. All geometries in the datasets are in CRS84, which is a geocentric SRS using non-linear *degree* units. In GeoSPARQL Jena, the handling of linear units with non-linear SRS geometries is achieved by temporarily transforming the geometries to a linear SRS. This is necessary as the ratio of *metres* to *degrees* is not fixed and varies according to the latitude of the coordinates. Alternative approaches may use a fixed ratio and tolerate the error as latitude varies.

Another query of note is Q18 which tests for the Simple Features *equals* relation. Strabon is noticeably quicker and completes this in less than 1 second when GeoSPARQL Jena requires just under 4.5 minutes and Parliament times out. This query checks the spatial equality between two graphs numbering 7,551 and 21,993 geometry literals to produce 166 million combinations. Therefore, the speed of processing this scale of combinations by Strabon is noticeable. Further investigation suggests that Strabon is not checking for spatial equality but only lexical equality by matching strings.

A final query of note is Q26 where GeoSPARQL Jena completes in less than 4 seconds while Strabon requires approximately 11 min and Parliament times out. This query tests the Simple Feature *touches* relation, where the boundaries of two geometries align but do not cross into each's interior. The two graphs used are the same graph that contains 325 complex multi-polygon geometries for 105 thousand combinations. GeoSPARQL Jena on demand indexing makes these geometries readily available and the optimisations for this relation, e.g. bounding boxes around the complex shapes, allow for the cases can be rejected quickly. Similar optimisations can be applied in Q25 and Q27 and could explain why GeoSPARQL Jena is noticeably quicker than Strabon.

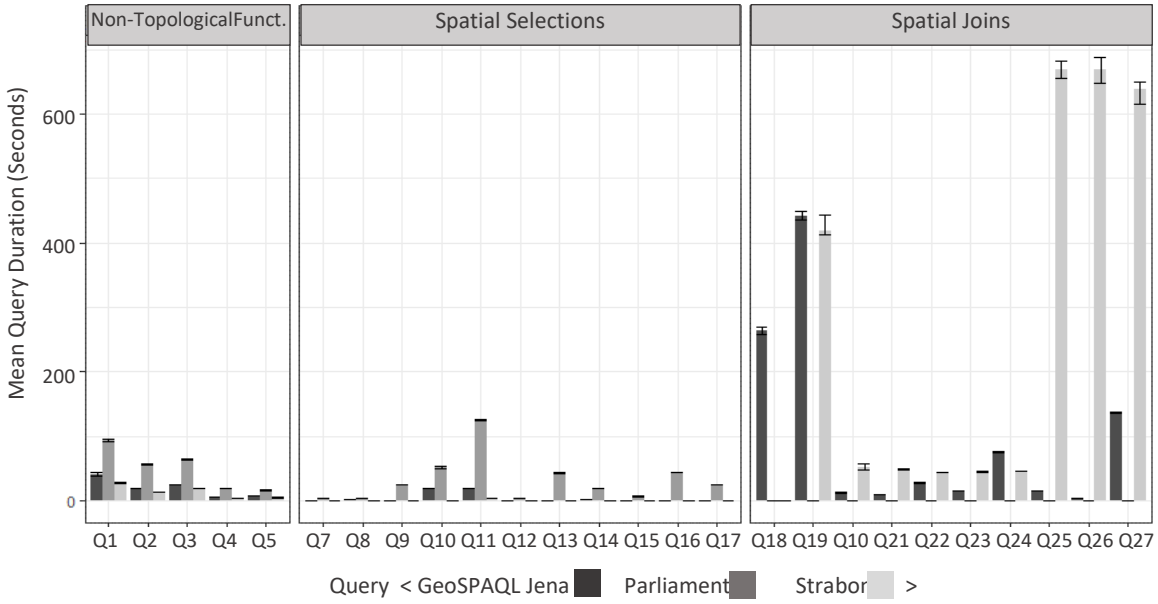

Figure 3: Mean Query Duration (seconds) by test system microbenchmark in Cold run. Error bars show min/max. Parliament timed out in Q18-Q27

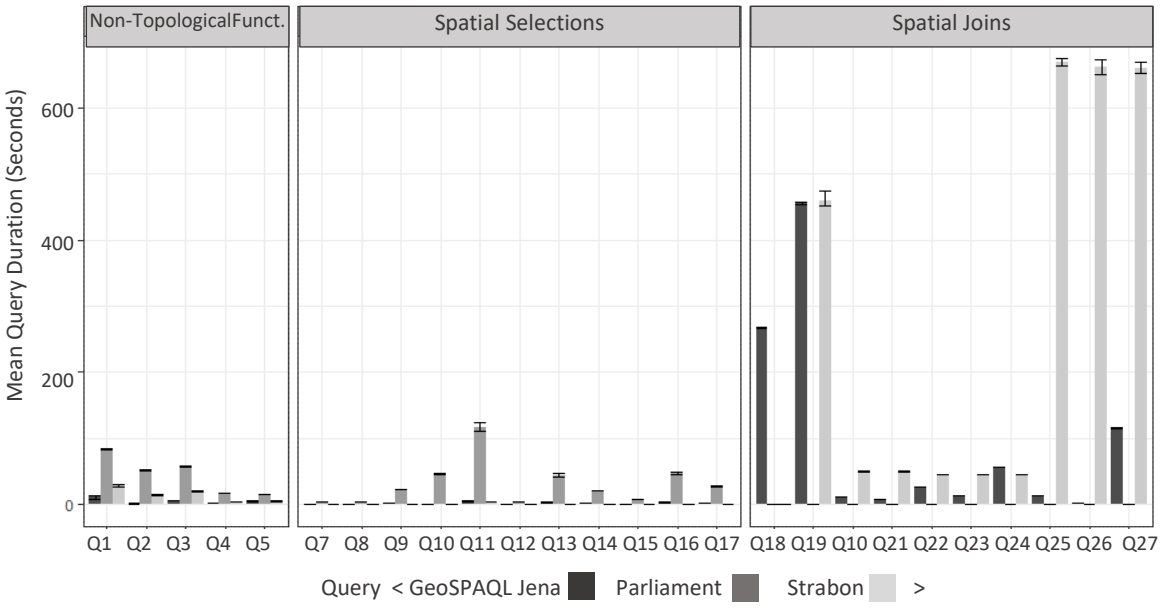

Figure 4: Mean Query Duration (seconds) by test system microbenchmark in Warm run. Error bars show min/max. Parliament timed out in Q18-Q27

Table 2 shows that Strabon achieves the most frequent fastest completion times in the Cold runs but GeoSPARQL Jena is more often fastest in the Warm runs. This reflects the on-demand indexing implemented in GeoSPARQL Jena. Parliament is consistently the slowest performer and did not complete one set of queries in the time limit. Strabon is quicker in the Non-Topological Functions and Spatial Selections queries. These queries utilise a single graph dataset while the Spatial Joins require geometries from two separate graph datasets. This suggests that Strabon is performing optimisations for intragraph operations during the lengthy initialization period, that may not be possible or tractable intergraph, i.e. the *Curse of Dimensionality* (Köppen, 2000).

Table 2: Microbenchmark test system rankings for Cold and Warm runs

	1st		2nd		3rd	
Test System	Cold	Warm	Cold	Warm	Cold	Warm
GeoSPARQL Jena	8	16	18	10	0	0
Parliament	0	0	0	0	26	26
Strabon	18	10	8	16	0	0

Although it would be expected that the Warm runs complete quicker than the Cold runs a comparison between the mean completion times finds this is not always the case with GeoSPARQL Jena 6 cases, Parliament 5 cases and Strabon 10 cases. In many cases the differences are fractional or less than a couple of seconds but the largest are exhibited by Strabon in Q19 and Q27 and GeoSPARQL Jena in Q19. It should also be noted that GeoSPARQL Jena is only marginally slower than Strabon in many queries, expect Q10 and Q11 and Q18 discussed earlier, while there is a noticeably lengthy difference in many queries in the Spatial Joins section for Strabon.

4.4 Geographica Macrobenchmark

The queries executed in this benchmark are those published by the Geographica project and follow the same numbering system. In keeping with the Geographica approach there is variation between query iterations. Each block of queries within a set uses related data but the data selected varies between iterations, e.g. MS0, MS1 and MS2 have consistent data in the first iteration but a different set of data is used in the second iteration. The same resulting queries have been used across each of the test systems to allow direct comparison. This variation between iterations can be seen in Figure 5 where there is more noticeable variation in maximum to mean completion durations (error bars show min/max). This can partly be attributed to many iterations returning zero results and resolving very quickly.

The query uses linear *metre* units for distance between two geometries. All geometries in the datasets are in CRS84, which is a geocentric SRS using non-linear *degree* units. In GeoSPARQL Jena, the handling of linear units with non-linear SRS geometries is achieved by temporarily transforming the geometries to a linear SRS. This is necessary as the ratio of *metres* to *degrees* is not fixed and varies according to the latitude of the coordinates. Alternative approaches may use a fixed ratio and tolerate the error as latitude varies.

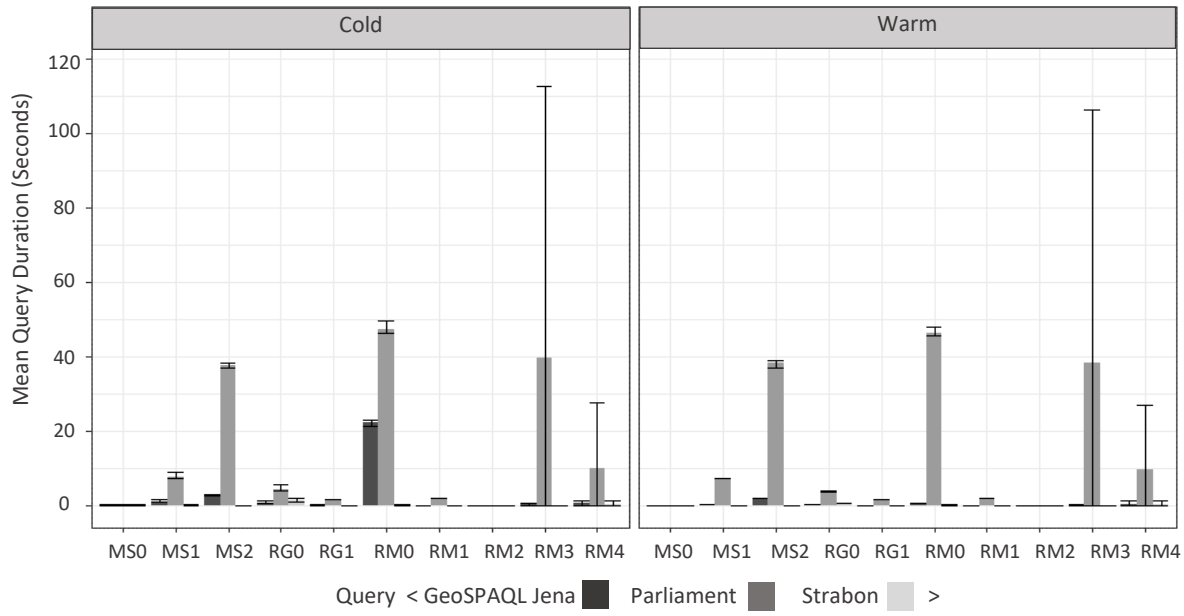

Figure 5: Mean Query Duration (seconds) by test system for both macrobenchmark runs

There is consistency in the number of results returned in all but two queries. Strabon returns zero results in four iterations of MS0 when Parliament and GeoSPARQL Jena both return a single result. The second case is MS2 where there is a difference in result count for the only non-zero iteration between Parliament and GeoSPARQL Jena. Strabon did not complete, repeatedly, this iteration and so no completion duration is reported. Strabon also did not complete all the iterations for RM3 and RG1. In all cases where Strabon did not produce results the

other test systems had non-zero result counts and Strabon produced an error rather than timing out, suggesting an issue with the processing of results within Strabon.

Parliament timed out after one hour in the RM5 query. This was also the most intensive query for GeoSPARQL Jena and required approximately 30 minutes in each iteration. The RM5 query uses two *intersection* filter functions and a geometry *difference* function with data from two graph datasets. In all iterations GeoSPARQL Jena required a similar duration to complete, even when no results were returned. Strabon was able to resolve the zero results in less than 1 second and only took 15 seconds for the non-zero result iteration. The i*ntersection* filter function is also used in microbenchmark Q19 and both test systems achieved similar results. This suggests that Strabon's query optimisation selected a resolution strategy which reduced the problem space much quicker than GeoSPARQL Jena.

The rankings across the three test systems in Table 3 show that GeoSPARQL Jena achieves the fastest results in the majority of queries. These rankings do not consider the discussed query manipulation for RM5 so Strabon is still ranked as faster than GeoSPARQL Jena in that case. Strabon achieves very quick results in certain cases but did not complete all iterations for three queries. Parliament did not resolve one query and is generally the slowest to complete.

Table 3: Macrobenchmark test system rankings for Cold and Warm runs

	1st		2nd		3rd	
Test System	Cold	Warm	Cold	Warm	Cold	Warm
GeoSPARQL Jena	6	7	5	4	0	0
Parliament	1	1	3	3	7	7
Strabon	4	3	3	4	4	4

5. Conclusion

This work sets out the design of a fully compliant implementation of the GeoSPARQL standard utilising an RDF graphstore. Previous implementations have achieved partial compliance of a sub-set of extensions. The examined implementations have generally provided geospatial and RDF functionality by extending relational databases requiring additional configuration and setup. Additional design points that have been achieved were Semantic Web standards compliance, minimal configuration, and a short initialisation period. This means that the implementation would be suited to both development and production environments.

An on-demand indexing approach was designed and implemented to retain geospatial and supporting data for improved performance. Experimentation was performed to consider the impact of controlling index size and retention periods on query duration. This innovative approach of short- and long-term caching of key data has been demonstrated to reduce query completion times by up to 20% without incurring initialisation delays. This on-demand indexing also has general utility for other SPARQL applications that require repeated processing of datatypes that are computationally expensive to de-serialise.

To understand the implementation's performance capabilities, a benchmarking framework has also been designed and implemented to perform a comparison with two existing GeoSPARQL systems: Strabon and Parliament.

The design of this framework is based upon importing, processing, and reporting on SPARQL queries, rather than a hardcoded queries as developed in the Geographica benchmarking framework for GeoSPARQL. Therefore, it has utility for benchmarking SPARQL queries broader than GeoSPARQL. This approach also provides transparency in the query content and data being benchmarked while variant queries can be easily written and processed. Queries can also be benchmarked on alternative datasets.

The reported benchmarking results show several advantages of the GeoSPARQL Jena implementation over the two benchmarked systems. The dataset loading and initialisation of the implementation are noticeably quicker. In the alternative systems, pre-query spatial index preparation is undertaken that is not incurred by the GeoSPARQL Jena implementation. Despite this, the benchmarking process has demonstrated that the GeoSPARQL Jena implementation has query times comparable or better than the alternative systems in all but one query case (RM5). The GeoSPARQL Jena implementation also completed all the GeoSPARQL benchmarking queries of the Geographica query set.

It has also been identified that minor manipulations to the benchmarking queries can trigger dramatic improvements in query resolution. This was found in the single query case (RM5) where GeoSPARQL Jena was dramatically slower than the Strabon test system. The application of two structural changes to the query, without altering content, reduced query time from 30 minutes to 8 seconds and overtook Strabon's performance.

This demonstrates that SPARQL query writing and optimisation can be very specific to the query engine being utilised. This highlights the challenge in preparing benchmarking queries which do not over accentuate the performance of a specific test system. It further emphasises the need for benchmarking frameworks to be adaptable in processing alternative versions of queries supplied by the user as designed and implemented in the benchmarking framework.

References

Apache Software Foundation. (2020) *Apache Jena* [online]. Available at: http://jena.apache.org/ [Accessed 2/2/2022].

Apache Software Foundation. (2018) *Apache Marmotta* [online]. Available at: http://marmotta.apache.org/kiwi/geosparql.html [Accessed 2/2/2022].

The Apache Software Foundation. (2019) *The Apache SIS library* [online]. Available at: https://sis.apache.org/ [Accessed 2/2/2022].

Battle, R. and Kolas, D. (2011) *Geosparql: enabling a geospatial semantic web*. Semantic Web Journal, 3(4), pp.355-370.

Butler, H., et al. (2016) *The geojson format*. Internet Engineering Task Force (IETF).

Eclipse Foundation. (2018) *Locationtech Java Topology Suite* [online]. Available at: https://projects.eclipse.org/projects/locationtech.jts [Accessed 2/2/2022].

Egenhofer, M.J. and Franzosa, R.D. (1991) *Point-set topological spatial relations. International Journal of Geographical Information System, 5 (2), 161-174.*

Garbis, G., Kyzirakos, K. and Koubarakis, M. (2013) *Geographica: A benchmark for geospatial RDF stores* (long version). In: International semantic web conference, Springer, pp. 343-359.

Harris, S., Seaborne, A. and Prud'hommeaux, E. (2013) *SPARQL 1.1 query language*. W3C Recommendation. (21 (10), 778.

Köppen, M. (2000) *The curse of dimensionality*. In 5th online world conference on soft computing in industrial applications (WSC5) (Vol. 1, pp. 4-8).

Kyzirakos, K., Karpathiotakis, M. And Koubarakis, M. (2012) *Strabon: a semantic geospatial DBMS*. *In: International Semantic Web Conference,* Springer, pp. 295-311.

OGC. (2020. *Simple Feature Access - Part 1: Common Architecture* [online]. Available at: https://www.ogc.org/standards/sfa [Accessed 2/2/2022].

Open Source Geospatial Foundation. (2020. *GeoTools The Open Source Java GIS Toolkit* [online]. Available at: https://geotools.org/ [Accessed 2/2/2022].

Schreiber, A.T. and Raimond, Y. (2014) *RDF 1.1 Primer*.

Spatial Reference System [online]. (2013) Available at: https://spatialreference.org/ [Accessed 2/2/2022].