Client-side Processing of Geospatial Linked Data with Triple Pattern Fragments

by

Shifan Gu, B.Sc.

Dissertation

Presented to the

University of Dublin, Trinity College

in fulfillment

of the requirements

for the Degree of

Master of Science in Computer Science

University of Dublin, Trinity College

August 2017

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Shifan Gu

August 28, 2017

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Shifan Gu

August 28, 2017

Acknowledgments

I would like to express my best gratitude to the Prof. Declan O'Sullivan and Dr. Christophe Debruyne for all the invaluable guidance and enthusiastic encouragement in this project. I also would like to say thank you to my parents for their understanding and support and Finally, I would express my thanks to Network and Distribute System class for this memorable year.

Shifan Gu

University of Dublin, Trinity College

August 2017

Client-side Processing of Geospatial Linked Data with Triple Pattern Fragments

Shifan Gu, B.Sc.
University of Dublin, Trinity College, 2017

Supervisor: Prof. Declan O'Sullivan
Assistant-Supervisor: Dr. Christophe Debruyne

Over the last few years, Trinity College Dublin and Ordnance Survey Ireland have been collaborating to publish Irelands geospatial information as Linked Data on the web. However, how to efficiently consume such rich data is considered a big problem in the industry. The most recommended way to solve this problem is to take advantage of the Triple Pattern Fragment approach which simply retrieves the triples that match a given triple pattern and then it is up to the client to decide how to efficiently execute the rest of the query specified. In this dissertation, we extend the Triple Pattern Fragment client to support GeoSPARQL, which is an OGC standard for representation and querying geospatial linked data. We also evaluated our implementation using the Geographica benchmark to assess the overall performance of the implementation. Finally, we applied the insights gathered from this evaluation to make recommendations as to how to best formulate GeoSPARQL queries to achieve efficient execution of queries when using the Triple Pattern Fragment implementation.

Contents

Acknowledgments **iv**

Abstract **v**

List of Tables **x**

List of Figures **xi**

Chapter 1 Introduction **1**

1.1 Motivation 1

1.2 Research Objectives 3

1.3 Outline 3

Chapter 2 Background **5**

2.1 Semantic Web 5

2.2 RDF 8

2.3 RDF Schema 9

2.4 SPARQL 10

2.5 GeoSPARQL 11

Chapter 3 State-of-Art **15**

3.1 RDF-based Web Interfaces 15

3.1.1 Linked Data Fragments(LDF) 15

3.1.2 Formal Framework For Comparing LDF 17
3.1.3 Data Dump 17
3.1.4 SPARQL endpoints 17
3.1.5 Triple Pattern Fragment 18
3.1.6 Binding-Restricted Triple pattern Fragments 18
3.2 Query Execution Approach 19

Chapter 4 Design **21**
4.1 TPF code base 21
4.1.1 Definition of TPF web interface 22
4.1.2 General work-flow 24
4.1.3 Architecture of TPF client 25
4.2 Design of GeoSPARQL extension 27
4.2.1 Properties functions vs filter functions 28
4.2.2 Solutions 29
4.2.3 Requirements 30
4.2.4 Coordinate Reference Systems 30
4.2.5 DE-9IM 31
4.3 Requirements Class in GeoSPARQL 33
4.3.1 Requirements for Spatial Query Filter Functions 33
4.3.2 Requirements for Non-topological Query Functions 35

Chapter 5 Implementation **38**
5.1 Technology Stack 38
5.2 Establishment of the TPF server 38
5.3 Implementation of Filter functions 39
5.4 Implementation of Non-topological Functions 42

Chapter 6 Experiment 45

6.1 Evaluation of Implementation 45

6.1.1 Geographica Benchmark 45

6.1.2 Datasets 46

6.1.3 Benchmark Queries 48

6.1.4 Supplementary Content 48

6.1.5 Performance Metrics 53

6.1.6 Experiment Setup 54

6.1.7 Benchmark Results 54

6.1.8 Summary 55

6.2 Performance analysis of TPF client 58

6.2.1 Analysis Setup 58

6.2.2 Memory analysis from external view 60

6.2.3 Memory analysis from internal view 61

6.2.4 CPU analysis from external view 64

6.2.5 CPU analysis from internal view 64

6.2.6 Summary 65

6.2.7 Query Formulation Recommendations 66

Chapter 7 Conclusions 69

7.1 Conclusion 69

7.2 Future work 70

Appendix A Abbreviations 71

Appendix B Parsed SPARQL JavaScript Object Example 73

Appendix C TPF Server Configuration File 78

Appendix D Supported Feature in TPF Client v2.0.5 82

Bibliography **84**

List of Tables

4.1 Simple Features Topological Relations [50] 34

4.2 Non-topological functions[50] 37

6.1 Real-world workload Dataset characteristics in Geographica 47

6.2 Synthetic workload Dataset characteristics in Geographica 48

6.3 Synthetic workload Queries in Geographica 49

6.4 Micro Benchmark Queries in Geographica 50

6.5 Macro Benchmark Queries in Geographica 51

6.6 Feature Expressed in Geographica Benchmark Queries 52

6.7 memory heap comparison of example query 61

6.8 performance comparison of equivalent query 67

D.1 Feature Expressed in GeoGraphica Benchmark Queries 83

List of Figures

2.1 Semantic Web Layered Architecture [13] 7
2.2 A RDF description with three triples and its labeled graph 8
2.3 An example of SPARQL query [22] 10

3.1 LDF spectrum[60] 16

4.1 An example TPF server 23
4.2 Running example of TPF client work-flow[7] 24
4.3 Package diagram of TPF client 26
4.4 Class diagram of TPF client 27
4.5 Intersection Matrix [10] 31
4.6 A motivating DE-9IM example [2] 32

5.1 example configuration file 39
5.2 An example of GeoSPARQL query [23] 40
5.3 iterator chain of example query 40
5.4 code snippet of filter iterator 41
5.5 partial code snippet of geof:sfTouches 42
5.6 Partial result of finding touching Irish counties 42
5.7 Find the cities around Athens 43
5.8 handle different literal types 44
5.9 Partial result of Finding cities around Athens 44

6.1 (a) spatial selections, and (b) spatial joins. 48
6.2 real-world benchmark results (1) 56
6.3 real-world benchmark results cnt. (2) 56
6.4 synthetic benchmark results (1) 57
6.5 synthetic benchmark results cnt. (2) 57
6.6 envelop example for performance analyze 59
6.7 memory consumption trend of example query 60
6.8 String object list in 62
6.9 function trace of geometry literal 62
6.10 code snippet of historical reader 63
6.11 code snippet of historical reader 64
6.12 CPU time statistic of different query 65
6.13 accurate CPU time statistic 66
6.14 Get around POI (a) 67
6.15 et around POI (b) 68

Chapter 1

Introduction

1.1 Motivation

The ADAPT Centre at Trinity College Dublin and Ordnance Survey Ireland (OSi), Irelands national mapping agency, are collaborating to publish Irelands geospatial information as Linked Data [17] on the Web[40][31]. Linked Data requires the use of the Resource Description Framework (RDF) [24], a W3C Recommendation to describe and represent that information on the Web. RDF is a common and flexible data model that serves to capture and represent information on the Web. With the help of RDF, spatial information can be easily exchanged among applications.

To query RDF data, SPARQL[52], also a W3C Recommendation, is the de facto standard query language. In 2008, SPARQL 1.0 was published as the fundamental version of SPARQL specification. In 2013, W3C SPARQL working group extended several aspects of SPARQL 1.0 and published SPARQL 1.1[35].SPARQL, however, is a query language for RDF triples and provides no support for geospatial information.

To support geospatial information, GeoSPARQL standard working group developed OGC GeoSPARQL that is called the Geographic Query Language for RDF Data 1.0 standard[50].

GeoSPARQL is not only an extension of SPARQL for querying geospatial data, GeoSPARQL also defines a vocabulary for representing geospatial data in RDF. The former was achieved by extending the set of filter functions and has some query rewriting rules for RDF in terms of the GeoSPARQL vocabulary.

To date, a couple of bespoke triplestores have been developed for the purpose of supporting the GeoSPARQL standard. Strabon [45] is a popular open source semantic spatiotemporal RDF store that supports both stSPARQL[44] and GeoSPARQL standard. Parliament[6] is another well-known high performance triplestore, which makes use of Apache Jena[41]. Unfortunately, no engine on the market was found to fully implement the recent standard for GeoSPARQL. A market research conducted by GeoKnow project[14] revealed this difference by making a qualitative comparison between different triplestores on geospatial support.

In addition to the compliance-to-standard challenge, another serious problem is the "low availability"[20]of queryable knowledge graphs on the Web, which seems to be caused by the observations (1) majority of knowledge graphs are not published in queryable form and (2) a large amount of public SPARQL endpoints suffer from frequent downtime.

To reduce the complexity of hosting a queryable public SPARQL endpoint and to improve availability, the Triple Pattern Fragments (TPF) [60] approach was proposed as a trade-off solution. TPF has been chosen as the basis for this dissertation research. Generally, a TPF server only returns the result set for a simple triple pattern <subject, predicate, object >, where each element is either a variable, a URI, or a literal. A TPF client is responsible for breaking down a complex SPARQL query into simple queries and to compute the overall result, only relying on the TPF server for the result sets of those triple patterns.

However, current implementations of TPF do not support GeoSPARQL, leading to the research objectives of this dissertation.

1.2 Research Objectives

Objectives of the research reported in this dissertation are:

- Extend a TPF client to support GeoSPARQL standard.
- Evaluate the developed extension with the Geographica benchmark[30] to test if the related functions are correctly implemented and working as expected.
- Collect and analysis the query run-time data to explore the best practice query pattern and based on the indications found, make recommendations for how to design queries to enhance performance of query execution.

1.3 Outline

This dissertation is organized as follows:

- **Chapter 2** - Describe the key standards and technologies in the area of Semantic Web and elaborates the core concept behind them.
- **Chapter 3** - Presents a literature review of current research with regards to Triple Pattern Fragments, including the research code base and some cutting-edge optimization research.
- **Chapter 4** - Illustrates the structure of Triple Pattern Fragments client as well as the general query work-flow. It also presents the overall design for extending Triple Pattern Fragments to support spatial predicates filter functions and spatial analysis functions.

- **Chapter 5** - Describes the implementation detail of GeoSPARQL spatial predicates filter functions and non-topological functions (spatial analysis functions).
- **Chapter 6** - Introduces the methodology adopted to evaluate overall performance and explores the best practice query patterns with an aim of providing insights for query optimization.
- **Chapter 7** - Give a conclusion of the entire dissertation.

Chapter 2

Background

This chapter first presents the background knowledge of the key technologies and standards related to this dissertation, in particular the GeoSPARQL standards which were proposed in order to provide uniform geographical support.

2.1 Semantic Web

World Wide Web has become the most far-reaching and widely-used media in influencing the ways that human gather information. Which is also known as the impetus of next generation technologies [28], Berners-Lee [16] purposed the methodology of Semantic Web that aims to "bring structure to the meaningful content" of Web pages, dealing with the unreliability of computers semantics processing and vesting the software a more flexible, integrated, automatic and self-adapting Web.

In short, the motivation of the Semantic Web is to create a more accessible data web for computers [48]. From users' perspective, it should provides more interactive and diversified experience in using the Web. Current Web is functioning in a manner that merely index the data and transmit it from the target server to the client. All the intelligent work like information classifying and filtering are conduct by the end user. In contrast,

Semantic Web was designed to link isolated data, merge the knowledge graph from different industries and organize the data in a more machine-readable way to accelerates the data flow in a global scope [39]. As a result, computers would be capable of taking up more intelligent work. For example, searching will no longer be limited to simple key words matching. Instead, it will bring in more semantic such as finding synonyms to help generate a more accurate result. In addition to this, automatically integrating the information across different sites could also become feasible.

However, building up a such more "semantic" web has raised a debate across the industry. Halevy et al. [34] purposed to constitute an "enormous Google" to find out the proper connections among vocabularies, terms and circumstances, which mainly relies on the efficacy of data. However, recent performance standstill of search engines implies the deficiency of this approach: no search engines can find a better way to represent the search result instead of the lists with scattered pages. Consequently, academia acknowledges that the design of Semantic Web should follows the below principles [13]: 1). ensure the structured and partly-structured data be accessible in World Wide Web in standardized formats; 2). create data set and readable meta data with its connections; and 3). utilize formal model to describe the underlying meanings of the data to make it accessible by computers.

In fact, the current World Wide Web remains large quantities of both structured and semi-structured data. But, principle part of the World Wide Web is generated from the content management system with only structured data set or database. However, even the rich structures in these data set are almost lost after the process of transferring structured data into human readable Hyper-text Markup Language(HTML). Therefore, if all the structured data set on the Web could be released and interlinked with each other, the idea of a more "semantic" Web would make sense.

To this respect, Berners-Lee developed [48] a layered architecture for Semantic Web, which is shown in Figure 2.1, which describes the basic technology stacks and entire framework. The following part explains main technology stacks of each layer. 1). Uni-

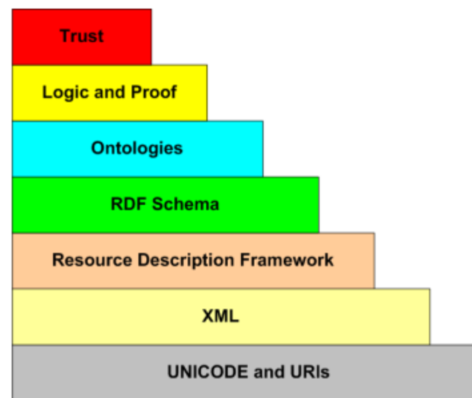


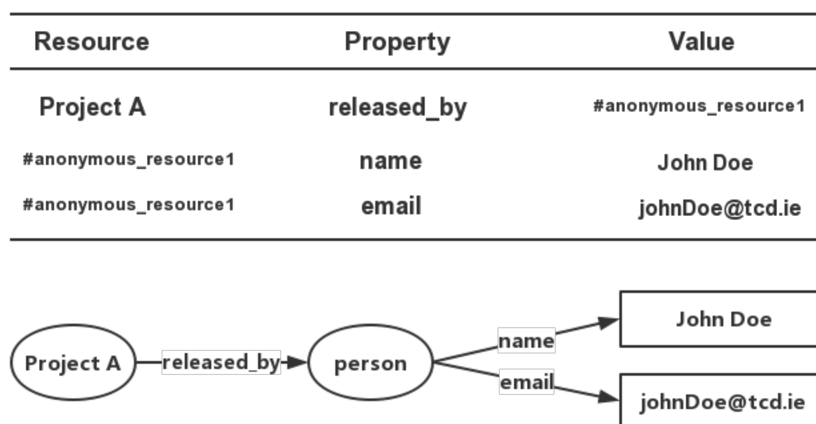
Figure 2.1: Semantic Web Layered Architecture [13]

code and URIs: Unicode is the standard of computer character representation and URI (Uniform Resource Identifier) is a string of characters used to identify a resource on the Web, which are known as the baselines to represent characters and resources in different formats. 2). XML: XML stands for Extensible Markup Language, which is designed to store and transport data. It allows users to design their own markup languages for limitless different types of documents. 3). RDF: The concept of Resource Description Framework (RDF) was purposed to express this labeled graph. RDF is a simple meta-data representation framework, which is also known as a standard data model, providing semantics that can be processed by a machine. 4). RDF Schema: RDF schema a simple type modeling language based on RDF for describing classes of resources and properties between them. 5). Ontologies: "An ontology is an explicit and formal specification of a conceptualization" [33], which is considered as a richer language to provide more abundant representations of object and its properties. 6). Logic and Proof: Normally, logic provides formal language to represent knowledge and well-understood formal semantics. Proof includes the real deductive procedure and use languages in low levels to show the

evidential process.

2.2 RDF

RDF stands for Resource Description Framework, which is a common data model for representing information and resources published on the Web. Compared to traditional approaches, RDF aims to organize the information in a machine-accessible manner [24] to facilitate data integration among different knowledge graphs. Briefly, A RDF description



Resource	Property	Value
Project A	released_by	#anonymous_resource1
#anonymous_resource1	name	John Doe
#anonymous_resource1	email	johnDoe@tcd.ie

Figure 2.2: A RDF description with three triples and its labeled graph

consists of three triples of objects (entities), attributes (properties) and corresponding values [42].This allows RDF to represent simple statements about resources as a graph of nodes and arcs representing the resources, and their properties and values. The basic structure of RDF utilizes labelled graph as the data model of object and its connections. In the graph, the object is known as node while the connections among objects are presented as edges. For example, figure 2.2 shows the three triples necessary to state that a specific project was released by someone with a name "John Doe" and an email address "johnDoe@tcd.ie". A directed labeled graph that is depicted from the previous RDF model is also shown in figure 2.2, which is defined as a collection of RDF triples.The

actual RDF data requires a more strict representation. It is also made up of set of subject-predicate-object triples, But, subject and predicate are URIs, and the object is URIs or literals[51] which can help uniquely identify the entity for ontology reasoning .

2.3 RDF Schema

RDF is a common language to provide ways for users to describe resources using their own statements. However, in order for those triples to be meaningful, RDF users are required to define the vocabularies on top of these statements [55]. These vocabularies represent a specific type restriction to describes the resource as well as the attributes. According to the definition of RDF, it neither presume to be related to specific application fields nor define semantics of any fields. Therefore, the concept of RDF vocabulary was proposed to describe classes, properties and their relationships by applying extensions to RDF in RDF Schema(RDFS), a set of reserved words.

RDF Schema defines three parts of constructions: core classes, relationships and restrictions [11]. A class in RDFS can be understood as a set of resources and the resources belonging to this class are called instance, which is defined as rdfs:Class. Likewise, a property in RDFS can be understood as the descriptions of instances, which is conducted by rdf:Property. Additionally, the rdfs:Resource, the rdf: Literal and the rdf: Statement are also the core classes in RDFS. When talking about defining relationships, the rdf:type shows the category of an instance. The rdfs:subClassOf can be used to implement a hierarchical structure. In order to define the property hierarchies, RDFS provides the rdfs:subPropertyOf. Two constructs are used as the core restrictions in RDFS: the rdfs:domain and the rdfs:range.

2.4 SPARQL

After transferring information to be the format of RDF, we need to store and utilize the existing RDF data to meet the requirements of reasoning and application development. Consequently, A query language called SPARQL provides the ability to gather specific content from the knowledge graph in the format of RDF, using selecting and filtering operations[13]. SPARQL is designed specifically for RDF, which has many similarities with the queries in SQL and is also a W3C Candidate Recommendation[49].

SPARQL selects information by utilizing graph pattern matching, as well as providing filtering mechanism based on comparing numbers and strings. A basic graph (BGP) pattern is a set of triple patterns and SPARQL operators. Therefore, SPARQL graph pattern matching is defined in terms of combining the results from triples that matching given basic graph patterns [37]. The operators of the SPARQL algebra are: AND (i.e., conjunction), UNION (i.e., disjunction), OPTIONAL (i.e., optional patterns, like left outer join) and FILTER (i.e., restriction).The overall structure of the language resembles SQL with its three main blocks: A WHERE clause that is composed of a graph pattern, A FROM clause that defines the datasets to be queried and A SELECT clause that specifies the final form containing results to return to users. Figure 2.3 gives a simple example

```
SELECT ?name ?birthcountry ?number ?country
WHERE {
    ?someone rdf:type :Person .
    ?someone :name ?name .
    ?someone :birthcountry ?birthcountry .
    OPTIONAL {?someone :ssn ?number}
     OPTIONAL {
        ?someone :passportno ?number .
        OPTIONAL { ?number :visacountry ?country }
     }
  }
```

Figure 2.3: An example of SPARQL query [22]

to describe SPARQL query with the optional patterns, whose returning results are the name of a person, the country of birth and the social security number or the passport number if available. In this query, the WHERE clause has two different formats that are non-optional and optional part. The previous one is from line1 to line5, which returns the properties(name and birthplace) of the instance Person. Relatively, the OPTIONAL clause in line06 searches for person's social security number[22]. The second OPTIONAL contains another nested OPTIONAL, which means if the passport number in line7 is available, then the nested OPTIONAL will find out those countries which have issued visas to the passport.

Generally, the core concept of SPARQL is selecting resources using graph pattern matching and implementing filtering section. SPARQL query utilizes the similar grammar as Turtle, in which both data and schema can be inquired. The UNION and OPTIONAL operators allow SPARQL deal with open-linked data easily [25].

2.5 GeoSPARQL

So far, we have introduced the basic concept related to semantic web and RDF as well as the idea of the query language SPARQL for RDF data. With respect to geospatial data, several schemes were proposed for encoding simple geometry in RDF, such as W3C Basic Geo vocabulary [19] that represent the point in WSG84 or GeoRSS[56] that support more complex geometries like rectangle and polygon. However these vocabularies inevitably have their own limitations. For this reason, OGC proposed GeoSPARQL standard in an attempt to provide a uniform, platform-independent access to geospatial RDF datasets and accordingly enable a rich query semantic for these datasets. In other words, GeoSPARQL is more than a standard for representing the geospatial RDF data, but also an extension of SPARQL for querying such data. In addition to this, GeoSPARQL also includes an Ontology that has been designed for the purpose of supporting exchange of

RDF data.

According to the most recently released standard[50] uses a modular design and the entire standard is made up of six components. Vendors can accordingly choose the components to implement based on their requirements. In this dissertation our implementation focuses on implementing the geometry component and geometry topology component. Each component is introduced as follows:

- **Core Component:**the core component defines the top-level RDFS/OWL classes for representing spatial objects. With the help of RDFS/OWL classes, systems that support RDFS entailment and systems that support OWL-based reasoning can hence read and understand the spatial objects. Implementations of this component must support SPARQL query language for RDF and the SPARQL Protocol for RDF and the SPARQL Query Results XML Format. This component enables the data exchange of spatial data with other RDF datasets.

- **Topology Vocabulary Component:** the topology component defines the properties for asserting and querying topological relations between spatial objects. There are three spatial relation families in the standards which are all described with DE-9IM.[58] DE-9IM describes the spatial relations by specifying the spatial dimensions of intersection of interiors, boundaries and exteriors between geometries. Vendors do not need to implement all the spatial relations family. It is open for each implementation to choose which to implement. Particularly, implementation of the OGC Simple Features family must allow properties: geo:sfEquals, geo:sfDisjoint, geo:sfIntersects, geo:sfCrosses, geo:sfWithin, geo:sfContains to be used in a graph pattern.

- **Geometry Component :**the geometry component defines RDFS data types for serializing geometry data, geometry related RDF properties, and non-topological spatial query functions for geometry objects. Single root geometry class is defined

with geo:Geometry. geo:hasGeometry would link a feature with a geometry that represents the spatial extent. With regards to these geometries, two approaches have been used to represent geometry literals - WKT and GML. WKT is short for well-known text that is a markup language for representing vector geometry objects across different spatial reference system while GML standards for Geography Markup Language which is the XML grammar defined by the Open OGC to express geographical features. The WKT serialization aligns the geometry types with ISO 19125 Simple Features, and the GML serialization aligns the geometry types with ISO 19107 Spatial Schema. geo:asWKT and geo:asGML are the properties that link the geometry with the geometry literal. To enable spatial analysis, geof:distance, geof:buffer, geof:convexHull, geof:intersection, geof:union, geof:difference, geof:symDifference, geof:envelope and geof:boundary must be implemented.

- **Geometry Topology Component :** the geometry topology component defines the topological query functions which served as the filter against geometry literals that return boolean value. . Implementations must support SPARQL extension function geof:sfEquals, geof:sfDisjoint, geof:sfIntersects, geof:sfTouches, geof:sfCrosses, geof:sfWithin, geof:sfContains, geof:sfOverlaps for the Simple Features model or other topological families like RCC8[53] or Egenhofer[26].

- **RDFS Entailment Component :** the RDF entailment component defines a mechanism for matching implicit RDF triples that are derived based on RDF and RDFS semantics. Implementations must support graph patterns involving terms from an RDFS/OWL class hierarchy of geometry types consistent with the one in the specified version of OGC Simple Features (ISO 19125-1) or the GML schema (OGC 07-036).

- **Query Rewrite Component :** the query rewrite component defines the RIF[18] rule for transforming a simple triple pattern that tests a topological relation between

two features into an equivalent query involving concrete geometries and topological query functions.

Chapter 3

State-of-Art

A large amount of the triples have been published that follow the rules of linked data which means that as a consequence thousands of knowledge graphs have been published on the Web. According to statistic from [27] LODstats project, over 88 billion Linked Data triples are distributed over 9960 knowledge graphs. However, how to consume such massive amount data has become a real challenge for the industry. This chapter is going to introduce several typical knowledge graph interfaces from a Linked Data Fragment (LDF) perspective, present state-of-art research achievements in this area to date and then introduce the different approaches for query execution.

3.1 RDF-based Web Interfaces

3.1.1 Linked Data Fragments(LDF)

Verborgh et al.[60] proposed a uniform view for analyzing existing knowledge graph web interfaces. They found that all kinds of knowledge web interface can respond to a request with a specific fragment of a knowledge graph. For example, a data dump response includes all the triples of a partial view of the knowledge graph. In contrast, SPARQL endpoints only responds with all the result triples of a query. Hence, they propose a general definition of these responses as Linked Data Fragment. Informally as they state

in their paper, An LDF of a RDF based knowledge is a resource consisting of a specific subset of a RDF triples of this graph, potentially combined with metadata and hypermedia controls to retrieve related LDFs. They also give a formal definition of the LDF but we would not discuss it in this dissertation for simplicity. In general, a LDF usually consists of the following three parts:

- Data: triples from a certain knowledge graph.
- Metadata: informative data is used to describe the data triples, e.g. triples count in LDF
- Controls: links and forms are used to retrieve other related LDFs

LDF with different selection granularity can represent different mechanisms of distributing query workload between the client and server. As shown in the Figure 3.1, the two extremes on the LDF spectrum are the data dump on one end and SPARQL results on the other end. Data dump does not require server to answer specific request as the client would download all the triples and consume them locally. In reverse, the SPARQL result

Figure 3.1: LDF spectrum[60]

means the server has to finish executing the SPARQL posted by the user based on their needs and then responds only with the results specific to that query specification. Both extremes of LDF have their deficiencies, as it is very hard to strike a balance between the client cost and server cost. However, some research has explored between the 2 extremes of the LDF spectrum. A couple of works has shown that it is feasible to finding a fragment type that can be used efficiently, whilst also having a reliable web interface. These works are introduced in the following subsections.

3.1.2 Formal Framework For Comparing LDF

Hartig et al.[38] proposed a Linked Data Fragment Machine (LDFM) which is a classical Turing machine provides a formalized way to model different LDF scenarios. LDFM model the server and client capabilities By proving formal results based on LDFMs, this model finally proved can be used to give a formal analysis between several metrics such as the number of requests sent to the server, and the bandwidth of communication between client and server.

3.1.3 Data Dump

Data dump, also referred as file-based datasets is an LDF that consists of all the triples of a dataset. Most of the times data dumps would be compress in an archive and then published on the web at a single URL. Consequently, a data dump does not require very complex control compared to the other LDF. Especially, in the metadata set of the data dump, it usually contains the data like release data or license. Although data dump is very easy to publish, it is very hard for user to make sure the dataset is always up-to-dated and monitoring the alteration of datasets requires good technical background. So, it is impractical for most of use cases.

3.1.4 SPARQL endpoints

SPARQL endpoint enables the client to write very specific SPARQL queries against the published dataset. Server would execute the query upon receiving the request and send back the result via HTTP request. Benefits of the SPARQL endpoint is that it requires very low client cost and barely no requirements for control and metadata as LDF. However, SPARQL endpoint suffers from two-fold availability problem. as 1) majority of the datasets are not published in the SPARQL endpoint 2) existing SPARQL endpoints suffers from frequent downtime.

3.1.5 Triple Pattern Fragment

Triple pattern fragments were designed to leverage the most basic building block of the SPARQL query - triple patterns to redistribute the query workload between the client and server. The client can only ask server for triple patterns. Server would then find out all the triples that match the given triple pattern and send back a triple pattern fragment. A triple pattern fragment is one kind of Linked Data Fragment and is made up of the triples data, metadata, and controls. Normally, a TPF consists of a small number of triples approximately, 100 triples and having a hypermedia control pointing to the next TPF. The more complex components of query processing like FILTER, ORDER BY and so on are conducted by the client in an efficient manner locally. The TPF requires very small server processing power as the evaluation work is done by the client which in turn improves the server availability. However, this benefit comes with the price that it takes more time to finish the query than a bespoke SPARQL endpoint.

In this situation, the client plays a very important role in improving the query efficiency. Verborgh et al.[60] proposed a greedy strategy that made a better use of metadata to address the client-side query planning problem. During the evaluation of the SPARQL Basic Group Pattern (BGP), the client would first split the BGP into several connect-independent sub-BGPs and evaluate these BGPs recursively. For each of the sub-BGPs, the client would start downloading the triples for the smallest triple pattern based on the count estimate of triples in the metadata. Then the evaluation results are bound to the next triple pattern and repeat the same process until the entire BGP is evaluated. A more detailed query execution work flow is introduced in the section 4.1.2.

3.1.6 Binding-Restricted Triple pattern Fragments

brTPF slightly extends the TPF interface that allows the client attach some intermediate results along with the triple pattern request. Response of such request is expected to

contain the triples that can potentially contribute to a join with the intermediate result in addition to merely matching the given triple pattern. As a result, It becomes possible to distribute the execution of joins between client and server by using the well-known bind join strategy by this means, HTTP request and response time would be reduced significantly.

Joachim et al. [59] conducted a research by extending the TPF server to support the literal substring matching and investigating the impacts of different implementation including elastic search and case-insensitive fm-index. Their research reveals that, this approach allows the filter-based sparql generate a faster response with significantly increase the server load, and such substring matching feature can be used to support other filters like complete regular expressions and range queries.

3.2 Query Execution Approach

Server-side Query Processing: Traditional query execution was completely conducted by the server. Server is responsible for hosting the datasets, parsing and executing the query. Client only responsible for writing the queries based on their needs and send it to the sever. Several efforts were devoted into optimizing the sever execution process. Markus et al.[57] proposed an approach by using the heuristic to predict the optimal join path. In addition to this, Buil-Aranda et al.[20] use the estimated query selectivity to rewrite the query and thus produce a less complex query. However, server-side query execution has two drawbacks that client may not aware what kind of query the server is able to perform, as not all the features in the SPARQL standard may supported by the server.

Client-side Query Processing: As the most well know research in this area, Hartig et al.[36] proposed several for the client side query processing. Especially the linked-

traversal-based querying. It would dereference the URLs inside the SPARQL queries and then traverse the links to fetching more relevant information. Drawback of this approach is very obvious that it takes very long time to finish the query and not all queries can be resolved in general.

Hybrid-Query Processing: The general idea of hybrid-query processing is that server and client both resolve part of the query and thus reduce the server cost and speed up the query execution in comparison with the client query execution approach. TPF and binding-restricted TPF have adopted such approach. In addition to this , Verborgh et al.[60] proposed an optimized strategy in the TPF client query execution by exploiting a local optimal sequence to evaluate the BGP.

Federated Query Processing: Federated query processing acquire the selection of proper server that is necessary to resolve the query and at the same time, ensure the network traffic and response time is minimized.

Chapter 4

Design

This chapter aims to give an overall view of the design work undertaken for the research dissertation. The first section would briefly introduce some technical details about the TPF code base, including the server API definition and the client side query execution workflow. Then, the next section would present the software architecture of TPF client in detail by using UML diagrams. After that, the next section would focus on illustrating what features should be included this extension and which part of the current TPF client should be altered to support such features. Finally, the last section would list the requirements for implementing the extension.

4.1 TPF code base

To begin with, official website of Linked Data Fragment provides various versions of Triple Pattern Fragment implementation by different programming language. We choose the most recent JavaScript release as the startup code base for two considerations, 1) Running JavaScript in the browser environment is compatible for all kinds of mobile device which has significant industrial potential if efficient geospatial calculation is enabled in TPF client. 2) Communication between TPF server and client would produce massive network load whilst the availability was the major consideration for the TPF server design. Node.js

was specially designed for the purpose of optimizing throughput and scalability in Web applications with many I/O operations which, under this scenario, is the most appropriate solution for given requirement.

4.1.1 Definition of TPF web interface

As explained in the section 3.1.1, LDF provides a uniform way to analyse the existing RDF web interfaces which also enables people to develop new RDF web interfaces by combining different characteristics of existing RDF web interface. TPF web interface is one such new web interface aimed at enabling live querying of RDF knowledge graph at the minimal cost of the server resources. To mitigate the availability issue, TPF web interface only offers the triple-pattern-based access to RDF knowledge graphs. This is based on the consideration that triple pattern is the most basic building block of SPARQL query and thus avoid the client downloading the entire knowledge graph. In addition to this, TPF web interface uses the hypermedia control data to facilitate the response. With the help of hypermedia control, a client can identify which TPF interface they are talking to and how to access the TPFs. In general, a TPF web interface should consist of three properties:

- **data:** all triples of a knowledge graph that match a given triple pattern;
- **metadata:** an estimate of the number of triples that match the given triple pattern;
- **controls:** a hypermedia form that allows clients to retrieve any TPF of the same knowledge graph.

Figure 4.1 shows an example TPF interface. Normally a TPF server should divide each fragment into reasonably sized pages to avoid clients accidentally download very large chunks. In this case, TPF server serves 100 triples per page. In addition, this page also displayed the metadata at the very beginning of the triple list to indicating how many triples are there in the entire TPF. In this dissertation, we developed an experimental

Geo Project

Available datasets

Browse the following datasets as Triple Pattern Fragments:

- Boundaries -- counties
- Boundaries -- electoral divisioins
- Greek-Administrative-Geography-Dataset
- CORINE-Land-Use/Land-Cover-Dataset
- LinkedGeoData-Dataset
- GeoNames-Dataset
- Hotspots-Dataset
- synthetic -- dataset
- DBpedia-Dataset

The current dataset ***index*** contains metadata about these datasets.

Dataset index

Query dataset index by triple pattern

subject:

predicate:

object:

Find matching triples

Matches in dataset index for *{ ?s ?p ?o }*

Showing triples 1 to 30 of 30 with 100 triples per page.

```
dataset   type   Dataset.
dataset   label   "Boundaries -- counties".
dataset   title   "Boundaries -- counties".
dataset   type   Dataset.
dataset   label   "Boundaries -- electoral divisioins".
dataset   title   "Boundaries -- electoral divisioins".
```

#LD Linked Data Fragments

Figure 4.1: An example TPF server

TPF interface to host Geographica datasets for later evaluation purposes. Detailed steps involving in how to setup a such TPF server are introduced in the next chapter and for the next section, we would briefly introduce the general workflow of SPARQL execution in TPF client.

4.1.2 General work-flow

To start the query execution on the client side, TPF client must be given the URI of an arbitrary page of some fragment of the TPF collection. To this end, we use the URI http://fragments.dbpedia.org/2016-04/en and the example query as show in figure 4.2 to elaborate the query process. This query is constructed to find out all the desserts made by plants from DBpedia 2016-04 knowledge graph. The live demo is available on the

```
SELECT ?dessert ?fruit
WHERE {
  ?dessert dbpedia-owl:type <http://dbpedia.org/resource/Dessert>;
           dbpedia-owl:ingredient ?fruit.
  ?fruit dbpedia-owl:kingdom <http://dbpedia.org/resource/Plant>.
}
```

Figure 4.2: Running example of TPF client work-flow[7]

website of Linked Data Fragment. This query is a simple query that consists of only one BGP group but is classical to describe the core idea of TPF-based query execution. To begin with, the query clause should be passed to the client either as a string argument or in a separate file. After the TPF client receives the query string, it would first send the request to the server to obtain the hypermedia control of the TPF page. Hypermedia control in TPF page includes all the necessary information to inform the client if the server supports the triple pattern lookup. Then client would then start to evaluate the BGP group based on server response. The TPF client uses a divide-and-conquer strategy to evaluate the BGP group. It first splits the entire BGPs to several connect sub-BGPs and ensures each sub-BGP is independent with each other, so that the evaluation of each

sub-BGPs can be executed in parallel. In the given example. It has only one connected sub-BGP. Next, for each of the independent sub-BGP, TPF client would find a solution mapping based on the estimated triples count metadata of each triple pattern and then the triple pattern with the minimal count would be resolved first. This approach can help reduce the HTTP request number since each match triple would cause a solution mapping. In the given example, the first triple pattern has around 425 matching triples, the second triple pattern has around 12459 triples, the third triple pattern has around 53,342 triples. According to the greedy strategy, TPF client would first send first triple pattern "?dessert dbpedia-owl:type <http://dbpedia.org/resource/Dessert>" to the server, that would result in 425 matching triples and then result values would be bound to the second triple pattern. In the next step, TPF client would map the solution of the chosen triple pattern to the rest of the triple patterns until it yields a complete solution.

In contrast to the TPF client, TPF server is "fixed" in terms of its functionality which simply response the triples that matching the given triple pattern and it is up to the client to finish the complex feature of the query. For this reason, it is fair to analysis the functional architecture of the TPF client to find out how the other query features are supported in this client and finally target the place where the TPF client should be modified to support GeoPSRAQL.

4.1.3 Architecture of TPF client

Figure 4.3 is the package diagram of the TPF client. It has two root packages that called "bin" and "lib". The "bin" package is used to store JavaScript files that provide a high level encapsulation for end users. In this case, ldf-client is the entry point of the TPF client. In contrast, "lib" package stores JavaScript files that served as underlayer libraries. There are six sub-packages in this "lib" package, namely "sparql", "util", "Triple-Pattern-Fragments", "browser", "extractor" and "writers". "sparql" package is

responsible for creating iterators for different types of the SPARQL queries. The "Triple-Pattern-Fragments" package has the files that dealing with triple pattern fragment related tasks like communicating with TPF server or evaluating the BGP. "extractor" package contains files that are used to extract metadata and hypermedia control information from HTTP requests. "Writers" stores various kinds of writer files that translate the SPARQL query result into certain format. "util" package is used to store some useful utility tools. "browser" packages contains the files that cope with browser environment. In a more detailed perspective, Figure 4.4 presents the class diagram of the TPF client. In this

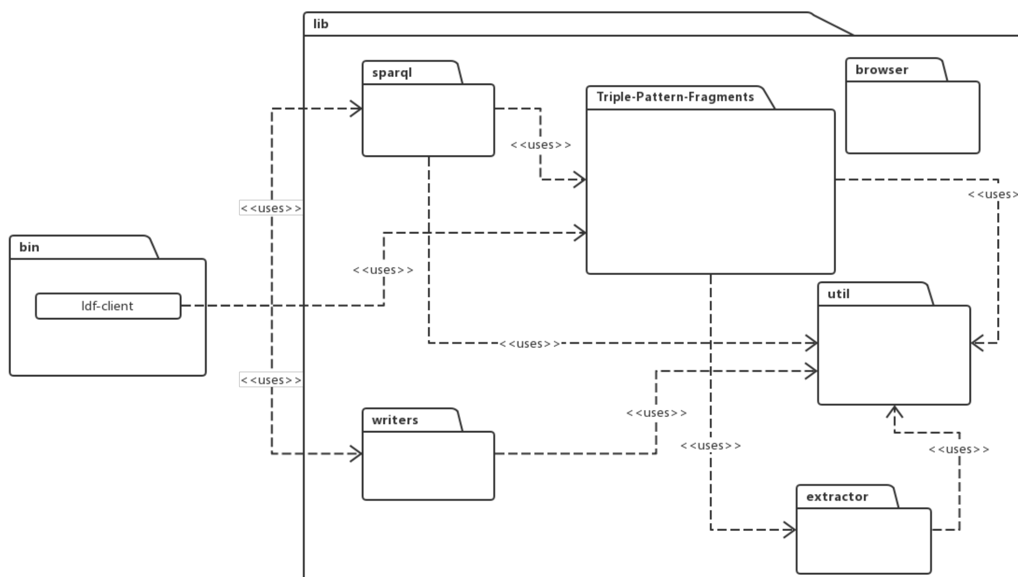


Figure 4.3: Package diagram of TPF client

diagram, three core classes are "SparqlIterator", "ldf-client" and "SparqlResultWriter" constitute the skeleton of SPARQL query processing. "ldf-client" accepts the input arguments and then make decisions based on these arguments. After this process, "ldf-client" would pass the query object along with the configuration file to the "SparqlIterator" class which would creating the cascading iterators to evaluate the query. Notably, "SparqlResultWriter" is a special iterator whose source is the "SparqlIterator". As a result, "SparqlResultWriter" can directly read the item from "SparqlIterator" and generate the result in user specified output format immediately. Rest of the classes are built around

these three core class with different purpose. But, generally they can be divided into two groups, classes that inherit from the core classes and classes that served as utility tools. The first group of classes leverages the benefits of object oriented programming that extends the "SparqlIterator" or "SparqlResultWriter" to satisfy different query needs. The second group of classes are aimed to assist this process. For example, the "HttpClient" class provides the HTTP service and the Logger provides the log service.

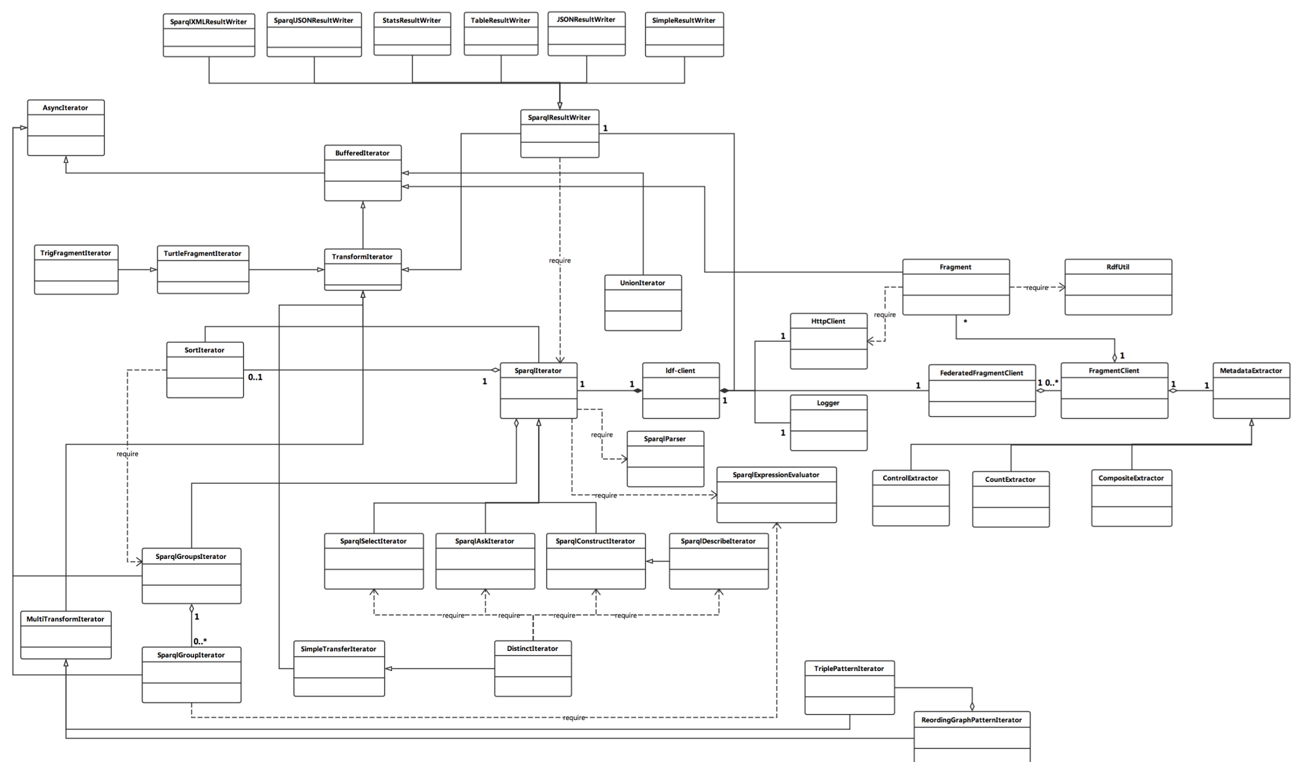


Figure 4.4: Class diagram of TPF client

4.2 Design of GeoSPARQL extension

As described in the section 2.5. The entire GeoSPARQL standard includes six components. However, some of them is optional for the vendor selection. So, we would go through these six component to find out a minimal features set to support GeoSPARQL in TPF client.

First component is the core component which defines an ontology for representing spatial objects. Which is not a concern in this dissertation. The second component is the topology vocabulary component that defines the properties for asserting and querying topological relations between spatial objects. Using spatial relationship property for querying is supported by TPF client if such relationship has already been built between the spatial objects in the dataset. Nevertheless, if such triples are not exists in the dataset, this requires the triplestore rewrite the query and generate the result accordingly. We decide to not support this feature as well as the query rewrite component in our extension which would be explained in the next section. Next, For the Geometry Components, it defines the geometry serialization as well as well as non-topological spatial analysis functions which are necessary for the new extension. The next component is the geometry topological function that defines several topological functions that serves as the filter functions, which also should be included. For the last components, RDFS entailment component is also not in the scope of this dissertation.

To conclude, the minimal set of features required to provide GeoSPARQL support in TPF client includes:

- Converting geometries between different serialization format.
- Provide support for spatial query functions that served as filter functions.
- Provide support for non-topological spatial analysis functions.

4.2.1 Properties functions vs filter functions

As described in previous section. GeoPSARQL provides two approaches to represent the spatial relationships in the query. The first one is using the spatial relation property in a triple pattern. For example, ?g1 geo:within ?g2. Another approach is using the filter functions to find out the eligible geometry pairs. Although GeoSPARQL describes the rule on how to rewrite the query to replace the spatial relations with filter function. This

feature is not required to be implemented to comply the standard. This rule is call the computed properties or the properties function.

In this dissertation, we decided not yet to support the properties function for two reasons. First of all, triplestores can not guarantee the correctness as one may assert ?x geo:within ?y whilst the two respective geometries are in fact disjoint with each other. Triplestore may not be able to refute this statement and thus lead to the conflict. Apart from this, we aimed at exploring the efficient approach to support geospatial calculation on the client to reduce server load. Enable filter function on the client can actually make client in control of the entire process.

4.2.2 Solutions

This section would briefly introduce the solutions for supporting features proposed in the section 4.2 in the TPF client. Implementation details would be introduced in chapter 5. First of all, supporting geometry serialization format conversion is widely supported by many public available JavaScript packages. we would seek a such package to support this feature.

Next, for the spatial query filter functions, these functions can be viewed as a extra subset of filter functions in the SPARQL standard. So this feature can be supported by extending the filter function sets in the TPF client. In TPF client, filter functions are stored as "operator-function" pair. When evaluate the "FILTER" clause, TPF client would pass the filter function signature to SparqlExpressionEvalustor to match the "operator" list, if such filter function exist in the list, it would return an function. This function would later be passed to the previous iterator and override the prototype filter functions. Then the iterator can filter the result based on this filter function.

Last but not the least, support the non-topological functions is slightly more complex than support filter functions as the return type of non-topological functions is not fixed and usually, it occurs either in "SELECT", "FILTER" or in the "BIND" clause. However, "BIND" is not supported in the current version of TPF client. So we would support it in the "SELECT" and "FILTER" clause. This can be achieved by extending "SparqlExpressionEvalustor.js" class and using "SparqlExpressionEvaluator" to extend the "SparqlIterator" class.

4.2.3 Requirements

Before introducing the formal content of implementation requirements, two fundamental concepts are introduced in advance to help build a better understanding. Section 4.1.1 introduces the coordinate system which is a basis for different geometries to establish the spatial connection. Next, section 4.1.2 introduces DE-9IM which is a topological model that used to describe spatial relationships between two geometries. Afterwards, a list of formal requirements definition was given at the end of the chapter.

4.2.4 Coordinate Reference Systems

Every geometry must be always georeferenced at a well-defined coordinate system, such system assists to provide a precise georeferencing location and meanwhile serve as a common bias for different geometries to establish a relationship based on their coordinates. In that respect, metadata should be maintained regarding spatial reference systems (SRS) or coordinate reference systems (CRS) with known parameterizations, which are widely adopted and allow transformations from one to another. Such systems define a set of rules that specify how coordinates are actually assigned to points. In general, a spatial reference system has four components:

- A coordinate system that describe a location relative to a center point.

- An ellipsoid, which defines an approximation for the centre and shape of the Earth as the Earth is not a perfect sphere.
- A datum, which defines the position of an ellipsoid relative to the centre of the Earth.
- A projection, that transforms positions from the curved surface of the Earth onto a plane.

In this dissertation, we adopted most widely used coordinate reference systems defined by the European Petroleum Survey Group which is short for "EPSG". For the geometries or features that represented in other coordinate reference system, we won't transform them into EPSG system but instead we would drop it for the time being.

4.2.5 DE-9IM

GeoSPARQL uses the DE-9IM to describe the spatial predicates. DE-9IM is the abbreviation of Dimensionally Extended nine-Intersection Model. It is a topological model as well as a standard that used to describe the spatial relationship between the geometries. In the area of computer spatial analysis, geometries could be viewed as a combination of three parts: interior, boundary and exterior. Equivalently, the spatial relationship between two geometries thus could be described as a set of relations across these three parts respectively which result in a intersection matrix as shown in the Form 4.5. where dim is

$$\mathrm{DE9IM}(a,b) = \begin{bmatrix} \dim(I(a) \cap I(b)) & \dim(I(a) \cap B(b)) & \dim(I(a) \cap E(b)) \\ \dim(B(a) \cap I(b)) & \dim(B(a) \cap B(b)) & \dim(B(a) \cap E(b)) \\ \dim(E(a) \cap I(b)) & \dim(E(a) \cap B(b)) & \dim(E(a) \cap E(b)) \end{bmatrix}$$

Figure 4.5: Intersection Matrix [10]

the maximum number of dimensions of the intersection of the interior (I), boundary (B), and exterior (E) of geometries a and b.

In this matrix, dimension of an empty sets is denoted as -1 or FALSE, The dimension of non-empty sets are denoted with the maximum number of dimensions of the intersection. specifically, 0 for points, 1 for lines, 2 for areas. Then, the domain of the model is 0,1,2,F. Consequently, this intersection matrix theoretically can describe 512 possible two-dimension spatial relationships.

Given the following example, a and b are two overlapping polygonal geometries which is shown in Figure 4.6 Reading in the left-to-right and top-to-bottom sequence, the DE-

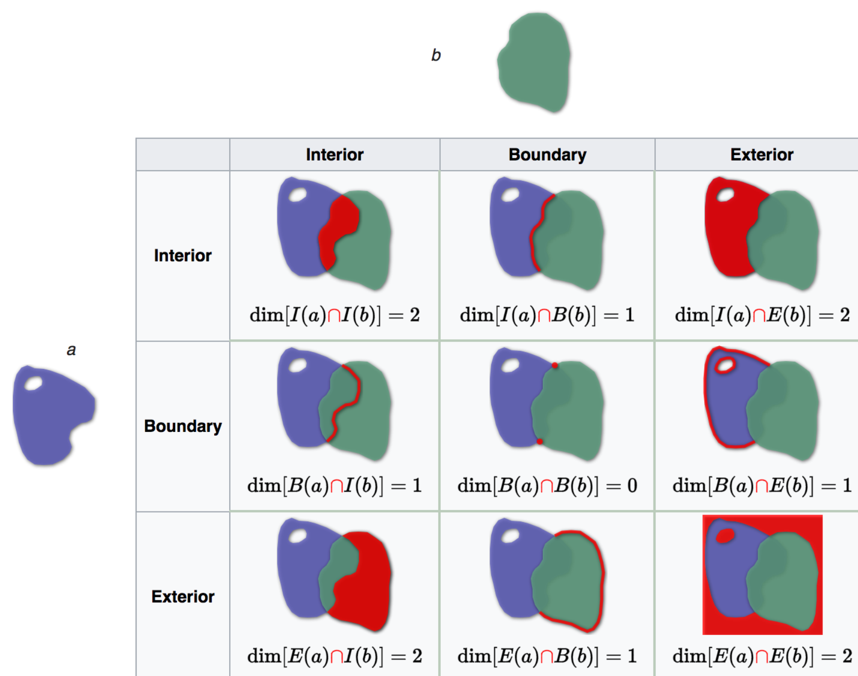


Figure 4.6: A motivating DE-9IM example [2]

9IM(a,b) string code is '212101212'. Normally, for output checking or pattern analysis, a matrix value or a string code can be checked by a "mask" which is a desired output value with optional asterisk symbols as wildcards "*" indicating output positions that the designer does not care about. Hence, the domain of mask is 0,1,2,F,*, or T,F,* for the Boolean form.

For English speakers, there are about 10 relations that have a name that reflects their semantics for example intersects. When testing two geometries against a scheme, the result of this test is a spatial predicate named by the scheme. In the GeoSPARQL there are eight spatial predicates defined in the Simple Features Relation Family. They all described using DE-9IM . Following section would give a view of their requirements

4.3 Requirements Class in GeoSPARQL

Recently released GeoSPARQL standard have defined a complete set of requirements class as well as the corresponding conformance class, as illustrate in the previous section, our implementation would focus on supporting topology geometry component, the following section would describe the formal requirements in detail.

4.3.1 Requirements for Spatial Query Filter Functions

As stated in clause-req 22, "Implementations shall support geof:sfEquals, geof:sfDisjoint, geof:sfIntersects, geof:sfTouches, geof:sfCrosses, geof:sfWithin, geof:sfContains, geof:sfOverlaps as SPARQL extension functions, consistent with their corresponding DE-9IM intersection patterns, as defined by Simple Features (ISO 19125-1)."

Filter functions defined for the Simple Features relation family, along with their DE-9IM intersection patterns, are shown in Table 4.1. Each function would take two arguments namely geom1 and geom2 of the geometry literal serialization type specified by serialization and version. Each function returns an xsd:boolean value of true if the specified relation exists between geom1 and geom2 and returns false otherwise. In each case, the spatial reference system of geom1 is used for spatial calculations.

Relation Name	Relation URI	Domain/Range	Applies To Geometry Types	DE-9IM Intersection Pattern
equals	geo:sfEquals	geo:SpatialObject	All	(TFFFTFFFT)
disjoint	geo:sfDisjoint	geo:SpatialObject	All	(FF*FF****)
intersects	geo:sfIntersects	geo:SpatialObject	All	(T******** *T******* ***T***** ****T****)
touches	geo:sfTouches	geo:SpatialObject	All except P/P	(FT******* F**T***** F***T****)
within	geo:sfWithin	geo:SpatialObject	All	(T*F**F***)
contains	geo:sfContains	geo:SpatialObject	All	(T*****FF*)
overlaps	geo:sfOverlaps	geo:SpatialObject	A/A, P/P, L/L	(T*T***T**) for A/A, P/P; (1*T***T**) for L/L
crosses	geo:sfCrosses	geo:SpatialObject	P/L, P/A, L/A, L/L	(T*T***T**) for P/L, P/A, L/A; (0********) for L/L

Table 4.1: Simple Features Topological Relations [50]

4.3.2 Requirements for Non-topological Query Functions

As stated in the clause-Req 19, "Implementations shall support geof:distance, geof:buffer, geof:convexHull, geof:intersection, geof:union, geof:difference, geof:symDifference, geof:envelope and geof:boundary as SPARQL extension functions, consistent with the definitions of the corresponding functions (distance, buffer, convexHull, intersection, difference, symDifference, envelope and boundary respectively) in Simple Features (ISO 19125-1)."

Implementation of the non-topological should be able to handle the following error raised by following invalid argument value :

- An argument of an unexpected type
- An invalid geometry literal value
- A geometry literal from a spatial reference system that is incompatible with the spatial reference system used for calculations
- An invalid units URI

Worth a note that, extension mechanism of SPARQL allows returning a value instead of raising an error.

As stated in the SPARQL specification- "SPARQL language extensions may provide additional associations between operators and operator functions; this amounts to adding rows to the table above. No additional operator may yield a result that replaces any result other than a type error in the semantics defined above. The consequence of this rule is that SPARQL extensions will produce at least the same solutions as an unextended implementation, and may, for some queries, produce more solutions."

This extension mechanism was designed to enable the compatibility between different geometry serializations. Consequently, stop the query execution exit when encounter

multiple geometry datatypes.

The last thing that needs attention is that some non-topological query functions use a unit of measure URI as an input argument. Some standard units of measure URIs are defined by OGC under the http://www.opengis.net/def/uom/OGC/1.0/ namespace.

Table 4.2 shows the non-topological query function signature as well as its description.

Function	Input	Return	Description
geof:distance	geom1: ogc:geomLiteral, geom2: ogc:geomLiteral, units: xsd:anyURI	xsd:double	Returns the shortest distance in units between any two Points in the two geometric objects as calculated in the spatial reference system of geom1.
geof:buffer	geom: ogc:geomLiteral, radius: xsd:double, units: xsd:anyURI	ogc:geomLiteral	This function returns a geometric object that represents all Points whose distance from geom1 is less than or equal to the radius measured in units. Calculations are in the spatial reference system of geom1.
geof:convexHull	geom1: ogc:geomLiteral	ogc:geomLiteral	This function returns a geometric object that represents all Points in the convex hull of geom1. Calculations are in the spatial reference system of geom1.
geof:intersection	geom1: ogc:geomLiteral, geom2: ogc:geomLiteral	ogc:geomLiteral	This function returns a geometric object that represents all Points in the intersection of geom1 with geom2. Calculations are in the spatial reference system of geom1.
geof:union	geom1: ogc:geomLiteral, geom2: ogc:geomLiteral,	ogc:geomLiteral	This function returns a geometric object that represents all Points in the union of geom1 with geom2. Calculations are in the spatial reference system of geom1.
geof:difference	geom1: ogc:geomLiteral, geom2: ogc:geomLiteral,	ogc:geomLiteral	This function returns a geometric object that represents all Points in the set difference of geom1 with geom2. Calculations are in the spatial reference system of geom1.
geof:symDifference	geom1: ogc:geomLiteral, geom2: ogc:geomLiteral,	ogc:geomLiteral	This function returns a geometric object that represents all Points in the set symmetric difference of geom1 with geom2. Calculations are in the spatial reference system of geom1.
geof:envelope	geom1: ogc:geomLiteral	ogc:geomLiteral	This function returns the minimum bounding box of geom1. Calculations are in the spatial reference system of geom1.
geof:boundary	geom1: ogc:geomLiteral	ogc:geomLiteral	This function returns the closure of the boundary of geom1. Calculations are in the spatial reference system of geom1.

Table 4.2: Non-topological functions[50]

Chapter 5

Implementation

This chapter leverages two motivating examples to illustrate the implementation details of both spatial predicates filter functions and non-topological functions.

5.1 Technology Stack

The TPF client was implemented using Node.JS and we extended the implementation of TPF version 2.2.2 [46] support GeoSPARQL. During this process, we adopted two external packages to assist our implementation; one for converting WKT into GeoJSON[8], and one for manipulating GeoJSON[32] object[9]. As stated in the previous chapter, the extended TPF client is targeted to run in a browser environment. So, we adopted browserify[12] to compress the entire TPF client into a single JavaScript file for import in an HTML document.

5.2 Establishment of the TPF server

Establishment of the TPF server was not very complex with the assistant of a server configuration file. The path of the configuration file should be passed as an argument in the server startup command. If such argument is missing, the program would instead

load a default configuration file to initialize the server. A typical look of the server configuration file is shown in the Figure 5.1. The configuration file is written in JSON which

```
{
  "title": "My Linked Data Fragments server",
  "datasources": {
    "dbpedia": {
      "title": "DBpedia 2014",
      "type": "HdtDatasource",
      "description": "DBpedia 2014 with an HDT back-end",
      "settings": { "file": "data/dbpedia2014.hdt" }
    },
    "dbpedia-sparql": {
      "title": "DBpedia 3.9 (Virtuoso)",
      "type": "SparqlDatasource",
      "description": "DBpedia 3.9 with a Virtuoso back-end",
      "settings": { "endpoint": "http://dbpedia.restdesc.org/", "defaultGraph": "http://dbpedia.org" }
    }
  }
}
```

Figure 5.1: example configuration file

is a lightweight data-interchange format. The example configuration file includes two attributes, the first one "title" is the name of TPF server and another one "datasources" is the relevant information set that used to specify datasource. TPF server supports five kinds of datasource respectively are, HDT files, N-Triples documents, Turtle documents, JSON-LD documents and SPARQL endpoints (with URL settings). Rest of the server configuration features like how to set a reverse proxy, can be found on its official website.

5.3 Implementation of Filter functions

To achieve a better understanding of the implementation, we give a motivating GeoSPARQL example with a filter function. As shown in the Figure 5.2, This query returns all the counties that touches each other. After receiving the query string, "ldf-client.js" would first parse the query string into a JavaScript object using "sparqlparser.js". Parsed query object can be found in the appendix. Then, the parsed query object would be passed to the "SparqlIterator" to create a cascading iterator chain, which is shown is the Figure 5.3. In this chain, each iterator is the source iterator of its upper iterator, for example,

```
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof:<http://www.opengis.net/def/function/geosparql/>
PREFIX osi: <http://ontologies.geohive.ie/osi#>

SELECT ?c1l ?c2l {
  ?c1 a osi:County .
  ?c1 rdfs:label ?c1l .
  ?c1 geo:hasGeometry ?g1 .

  ?c2 a osi:County .
  ?c2 rdfs:label ?c2l .
  ?c2 geo:hasGeometry ?g2 .

  FILTER (?c1 != ?c2)
  FILTER langMatches( lang(?c1l), "en" )
  FILTER langMatches( lang(?c2l), "en" )

  ?g1 geo:asWKT ?w1 .
  ?g2 geo:asWKT ?w2 .

  FILTER(geof:sfTouches(?w1, ?w2))
}
```

Figure 5.2: An example of GeoSPARQL query [23]

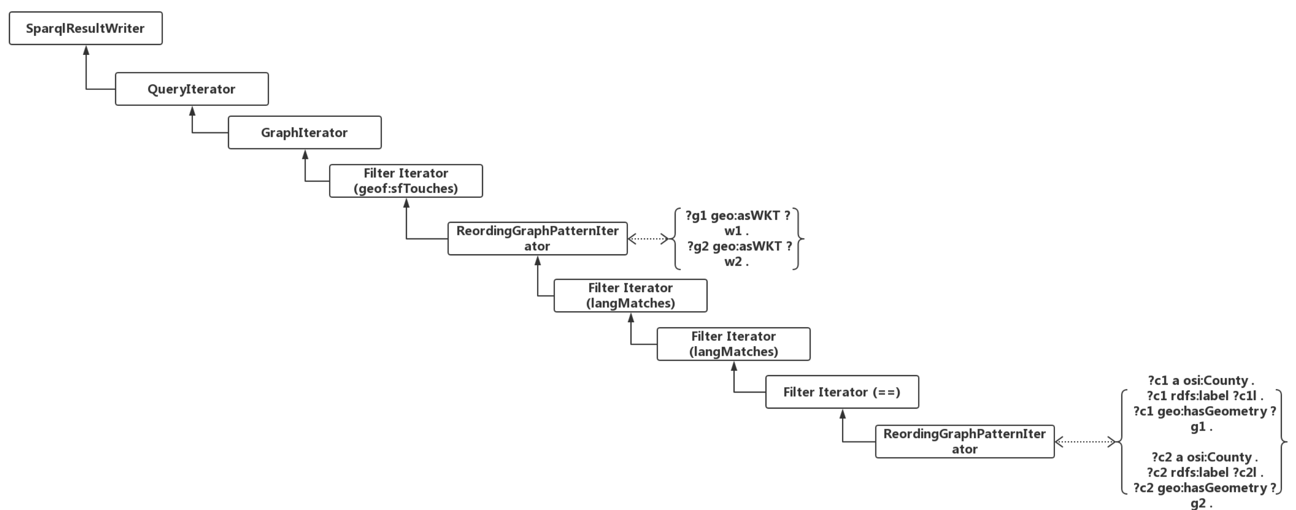


Figure 5.3: iterator chain of example query

the GraphIterator is the source iterator of QueryIterator and QueryIterator is the source iterator of SparqlResultWriter. Items are passed along this chain. "FILTER" is a special iterator in TPF client, it not just reads the item from one iterator and pass it to the next iterator, but, it would invoke the prototype filter function in the iterator and apply given filter function to all its items. The related code snippet is shown in the Figure 5.4. In

```
case 'filter':
  // A set of bindings does not match the filter
  // if it evaluates to 0/false, or errors
  var evaluate = new SparqlExpressionEvaluator(group.expression);
  return source.filter(function (bindings) {
    try {
      return !/^"false"|^"0"/.test(evaluate(bindings));
    }
    catch (error) {
      return false;
    }
  });
```

Figure 5.4: code snippet of filter iterator

this code, the returned source variable has a type of "AsyncIterator" and it invoked a prototype filter function with an anonymous function as argument. In this anonymous function, "evaluate()" is the actual filter function that take effect and it is returned by "SparqlExpressionEvaluator". The essential of "SparqlExpressionEvaluator" is a filter function object list, it returns the function object be matching given filter function signature which is also referred as "operator" in the code.

In this case, TPF client would first evaluate the BGP with first six triple pattern and then pass the result to the iterator with filter function "c1l==c2l" and so on so forth until the reach the filter with "geof:sfTouches".

The implementation of "geof:sfTouches" would strictly follow the GeSPARQL standard and so as the rest of spatial predicates. In particularly, the implementation of these handler functions should take good care of the coordinating referencing system in the Geometry String as this may cause conflict or generate an incorrect result. For the time being, current implementation only accepts the geometry that avail EPSG standard. The

rest representation would result in a UnsupportedCoordinateSystem exception. The figure 5.6 shows the final result of executing this example. A partial code snippet is shown in Figure 5.5 Notably, "ReordingGraphPatternIterator" would reorder the evaluation se-

```
function touches(a, b) {

  a = filterCoordinateSystem(a);
  b = filterCoordinateSystem(b);
  // This implementation assumes that things that touch do not contain polygons in the
  // intersection. This assumption might be naive and needs further investigating
  // TODO: check approach
  var gA = WKT.parse(N3Util.getLiteralValue(a));
  var gB = WKT.parse(N3Util.getLiteralValue(b));
  var intersection = turf.intersect(gA, gB);
  if (intersection !== undefined) {
    if (!isOrContainsPolygon(intersection))
      return XSD_TRUE;
  }
  return XSD_FALSE;
}
```

Figure 5.5: partial code snippet of geof:sfTouches

quence of BGP based on utilize the greedy strategy and then create a sub-iterator chain inside this iterator. This design would not affect our implementation, but it would provide useful insight for us to propose an optimal query pattern. But for the current state. We would focus on extending the SparqlExpressionEvaluator which is served as the key component in the filter component.


```
[
{"?c1l":"\"LAOIS\"@en","?c2l":"\"CARLOW\"@en"},
{"?c1l":"\"WEXFORD\"@en","?c2l":"\"CARLOW\"@en"},
{"?c1l":"\"KILDARE\"@en","?c2l":"\"CARLOW\"@en"},
{"?c1l":"\"KILKENNY\"@en","?c2l":"\"CARLOW\"@en"},
{"?c1l":"\"CARLOW\"@en","?c2l":"\"LAOIS\"@en"},
{"?c1l":"\"TIPPERARY\"@en","?c2l":"\"LAOIS\"@en"},
{"?c1l":"\"KILKENNY\"@en","?c2l":"\"LAOIS\"@en"},
{"?c1l":"\"SLIGO\"@en","?c2l":"\"LEITRIM\"@en"},
```


Figure 5.6: Partial result of finding touching Irish counties

5.4 Implementation of Non-topological Functions

Implementing the non-topological function generally follows the same approach as described in the previous section. However, the difference is that filter function would

return a boolean value to indicate if the condition was satisfied. In contrast, the Non-topological function would return different kinds of literals as a result. For example, the "geof:buffer" returns the WKT string while the "geof:distance" would return numerical literal. This problem can be resolved by return the entire result object and evaluating literal types at the binding process. The core code snippet is shown in the Figure 5.8.

```
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
PREFIX ontology: <http://www.geonames.org/ontology#>
PREFIX opengis: <http://www.opengis.net/def/uom/OGC/1.0/>

SELECT ?l1 {

  ?athens ontology:name "Athens" .

  ?athens ontology:hasGeometry ?g1 .

  ?g1 ontology:asWKT ?w1 .

  ?x1 ontology:name ?l1 .

  ?x1 ontology:hasGeometry ?g2 .

  ?g2 ontology:asWKT ?w2 .

  FILTER(geof:distance(?w1,?w2,opengis:metre)<5000)

}
```

Figure 5.7: Find the cities around Athens

To demonstrate the implementation of non-topological functions, we pick up a running example in Figure 5.7 that utilize the "geof:distance" function to find all the cites whose distance between Athens is less than 5000 metres and the running result is shown in the Figure 5.9

```
// Call the operator on the evaluated arguments
var result = operator.apply(null, args);
// Convert result if necessary
switch (operator.resultType) {
case 'numeric':
 // TODO: determine type instead of taking the type of the first argument
 var type = N3Util.getLiteralType(origArgs[0]) || XSD_INTEGER;
 return '"' + result + '"^^' + type;
case 'boolean':
 return result ? XSD_TRUE : XSD_FALSE;
case 'wkt' :
 return '"' + result + '"^^' + WKT_LITERAL;
default:
 return result;
}
```

Figure 5.8: handle different literal types

```
[
{"?l1":"\"Víron\""},
{"?l1":"\"Távros\""},
{"?l1":"\"Pláka\""},
{"?l1":"\"Peristérion\""},
{"?l1":"\"Periféreia Protévousis\""},
{"?l1":"\"Kallithéa\""},
{"?l1":"\"Néo Fáliro\""},
{"?l1":"\"Néa Smírni\""},
{"?l1":"\"Néa Fókaia\""},
{"?l1":"\"Plaka\""},
{"?l1":"\"Likavittós\""},
{"?l1":"\"Kopanás\""},
{"?l1":"\"Kolokinthoú\""},
{"?l1":"\"Kipséli\""},
```

Figure 5.9: Partial result of Finding cities around Athens

Chapter 6

Experiment

This chapter focuses on testing our GeoSPARQL extension of the TPF client. We adopted the Geographica benchmark to drive the implementation process and finally produce a complete result set for the entire extension. The second part focuses on using some performance analyze tools to extract the characteristics of the query process on TPF client and hence present insights gathered from this process and accordingly make some recommendations for a user of the extended TPF client.

6.1 Evaluation of Implementation

As few existing RDF benchmarks are related to the processing of geospatial data, there is no benchmark designed for the purpose of evaluating triple pattern fragments. Therefore we decided instead to use the Geographica[30] benchmark as the basis for undertaking an evaluation which is the most up-to-date benchmark developed for evaluating the geospatial RDF triplestore. All of the benchmark datasets and queries are publicly available.

6.1.1 Geographica Benchmark

Geographica is made up of two workloads along with their own datasets and queries, namely the real-word workload and the synthetic workload. Real-world workload uses

the linked datasets that publicly available on the web which covered a large range of geometry types. In addition to this, real-world workload follows the approach adopted by Jackpine[54] that defines a micro benchmark as well as a macro benchmark. Micro benchmark is used to test the primitive spatial functions while macro benchmark is used to test the performance of RDF triplestore in some real-world scenarios. As in the synthetic workload, producer of Geographica generated synthetic datasets of various size to simulate a controlled environment so that triplestores can be evaluated by queries with different thematic and spatial selectivity. Geographica did not capture the entire GeoSPARQL standard but, is sufficient enough to test our implementation and gives indication of its performance against the GeoSPARQL-enabled triplestore for future exploration.

6.1.2 Datasets

real-world workload datasets: Real-world workload datasets includes six publicly available datasets related to spatial information of Greece. Respectively, LinkedGeoData(LGD)[1], Greek Administrative Geography(GAG)[4], CORINE Land Use/Land Cover(CLC)[3] dataset for Greece, wild fire hotspots datasets produced by the National Observatory of Athens (NOA), partial datasets of DBpedia[15] as well as GeoNames referring to Greece. DBpedia and GeoNames are very significant parts of the Linked Open Data Cloud whose geospatial information are expressed by points. With regards to line strings, LGD dataset offers richer geospatial information from OpenStreetMap[5] that describes the road network and rivers of Greece. GAG and CLC dataset provided complex polygons. The CLC dataset is published by the European Environmental Agency which contains the data of entire European land cover. Finally, hotspots dataset is made of polygons representing wild fire hotspots. Some important characteristics of the datasets used can be found in Table 6.1

synthetic workload datasets: synthetic datasets of arbitrary size in Geographica was

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

Table 6.1: Real-world workload Dataset characteristics in Geographica

generated by a special generator to simulates four types of geographical features which are: states in a country, land ownership, roads and points of interest. Each of the dataset includes a minimal ontology that is built on top of the GeoSPARQL ontology and vocabulary. Synthetic workload enables the queries of different thematic and selectivity by following two designs. To ensure the thematic of each generated query, every feature is assigned with a number that represents one kind of thematic tags. Each thematic tag consists of a key-value pair of strings. As every feature is tagged with key 1, every other feature with key 2, every fourth feature with key 4 and so on so forth,until up to key 2^k where k is a positive integer. For the selectivity, each land ownership is represented by a hexagon and the entire spatial extent was constituted by $n * n$ such hexagons. Each land ownership hexagon is the size of nine states which also has a shape of hexagon. The generated road dataset consists of n features with sloping line geometries. Half of the line geometries are roughly horizontal and the rest are roughly vertical. Each line consists of $n/2 + 1$ line segments. Generated point geometries are uniformly placed across the lines. The cardinality of the point of interest geometries is thus the power of n.

Obviously, size of the generated dataset is decided by the value of n and k 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. Geographica produced a recommended dataset by setting n=512 and k=9 and proposed a list of corresponding queries. Characteristics of this dataset is shown in Table 6.2

Size	Triples	Number of land ownership	Number of states	Number of roads	Number of point of interests
745 MB	3880224	262144	28900	512	262144

Table 6.2: Synthetic workload Dataset characteristics in Geographica

6.1.3 Benchmark Queries

Real-world workload queries: Queries in micro benchmark was designed to test the primitive spatial components by using simple SPARQL queries. It first tests the simple spatial selection queries and then tests the more complex spatial join operations. Apart from these, micro benchmark also test the non-topological functions as well as some aggregation functions. Macro benchmark evaluates the performance of given RDF store in certain application scenarios. All the descriptions and expressed features of the real-world queries are summarized in the Table 6.4 and Table 6.5

Synthetic workload queries: The synthetic workload queries are instantiated from two query templates presented in Figure 6.1 which performs spatial selection and spatial join respectively. Table 6.3 lists the queries generated for this purpose.

(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))}
```

Figure 6.1: (a) spatial selections, and (b) spatial joins.

6.1.4 Supplementary Content

Geographica benchmark covers most relevant content regarding GeoSPARQL, how- ever some items remain unclear but have the same importance. First concern is that, Geo-

Spatial Selection			
Query	Operation	Tag	Percentage of Given Rectangle
Synthetic_Selection_Intersects_1_1.0	sfIntersects	1	100%
Synthetic_Selection_Intersects_512_1.0	sfIntersects	512	100%
Synthetic_Selection_Intersects_1_0.75	sfIntersects	1	75%
Synthetic_Selection_Intersects_512_0.75	sfIntersects	512	75%
Synthetic_Selection_Intersects_1_0.5	sfIntersects	1	50%
Synthetic_Selection_Intersects_512_0.5	sfIntersects	512	50%
Synthetic_Selection_Intersects_1_0.25	sfIntersects	1	25%
Synthetic_Selection_Intersects_512_0.25	sfIntersects	512	25%
Synthetic_Selection_Intersects_1_0.1	sfIntersects	1	10%
Synthetic_Selection_Intersects_512_0.1	sfIntersects	512	10%
Synthetic_Selection_Intersects_1_0.001	sfIntersects	1	0.1%
Synthetic_Selection_Intersects_512_0.001	sfIntersects	512	0.1%
Synthetic_Selection_Within_1_1.0	sfWithin	1	100%
Synthetic_Selection_Within_512_1.0	sfWithin	512	100%
Synthetic_Selection_Within_1_0.75	sfWithin	1	75%
Synthetic_Selection_Within_512_0.75	sfWithin	512	75%
Synthetic_Selection_Within_1_0.5	sfWithin	1	50%
Synthetic_Selection_Within_512_0.5	sfWithin	512	50%
Synthetic_Selection_Within_1_0.25	sfWithin	1	25%
Synthetic_Selection_Within_512_0.25	sfWithin	512	25%
Synthetic_Selection_Intersects_1_0.1	sfWithin	1	10%
Synthetic_Selection_Within_512_0.1	sfWithin	512	10%
Synthetic_Selection_Within_1_0.001	sfWithin	1	0.1%
Synthetic_Selection_Within_512_0.001	sfWithin	512	0.1%
Spatial Join			
Query	Operation	Tag 1	Tag 2
Synthetic_Join_Intersects_1_1	sfWithin	1	1
Synthetic_Join_Intersects_1_512	sfWithin	1	512
Synthetic_Join_Intersects_512_1	sfWithin	512	1
Synthetic_Join_Intersects_512_512	sfWithin	512	512
Synthetic_Join_Touches_1_1	sfTouches	1	1
Synthetic_Join_Touches_1_512	sfTouches	1	512
Synthetic_Join_Touches_512_1	sfTouches	512	1
Synthetic_Join_Touches_512_512	sfTouches	512	512
Synthetic_Join_Within_1_1	sfWithin	1	1
Synthetic_Join_Within_1_512	sfWithin	1	512
Synthetic_Join_Within_512_1	sfWithin	512	1
Synthetic_Join_Within_512_512	sfWithin	512	512

Table 6.3: Synthetic workload Queries in Geographica

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 Join		
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 6.4: Micro Benchmark Queries in Geographica

graphica can not help check how well the triplestore is compliant to the standard. Some new features introduced in SPARQL 1.1 like Negation, are not covered in the bench-

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 6.5: Macro Benchmark Queries in Geographica

mark.Table 6.5 summarizes the SPARQL features expressed in benchmark queries. The second concern is that the Geographica benchmark does not cover the entire spatial relationship set (e.g. geof:sfContains) as well as the supporting methods for spatial analysis(e.g. geof:difference) which are also referred as non-topological functions. In other words, the benchmark guaranteed the query selectivity test through the synthetic workload, whereas the real-world workload does not provide a complete query semantic test. In the experiment however, we did not extend the original benchmark with new queries to address the previously mentioned concerns. But, for the readers who do want to come up with a new benchmark for this purpose, we provide a complete set of supported features in the most recent TPF client in the appendix. We also recommend that the design work of a new benchmark follow the guidelines proposed by Kolas [43] for the spatial semantic web system. It generally states four primary types of spatial queries that must be covered: location queries, range queries, spatial joins, and nearest neighbor queries. So,

any benchmark design should take this guideline into consideration and try to achieve a balance between these four categories.

Feature	Micro Benchmark	Macro Benchmark	Synthetic Benchmark
Basic Group Pattern	✓	✓	✓
Group Graph Pattern			
Filter	✓	✓	✓
REGEX Operator			
OPTIONAL Operator		✓	
UNION Operator			
Negation			
Property Path			
BIND Operator			
VALUES Operator			
GROUP BY Modifier		✓	
HAVING Modifier			
LIMIT Modifier		✓	
OFFSET Modifier			
DINSTINCT Modifier			
REDUCED Modifier			
Aggregates			
Aggregation Projection	✓	✓	
Subqueries			
GRAPH Operator	✓		
ASK			
CONSTRUCT			
DESCRIBE			
Primary Functions		✓(bound)	
Functions on RDF Terms			
Functions on String			
Functions on Numerics			
Functions on Dates and Times			
Hash Functions			

Table 6.6: Feature Expressed in Geographica Benchmark Queries

Furthermore, there were two problems that needed to be taken good care of before running the benchmark test. First of all, most of the real-world benchmark queries using "GRAPH" modifier to specify different dataset. However, this feature was not supported in the recent version Triple Pattern Fragment client owing to the Triple Pattern Fragment

client not loading the entire dataset during the query execution. Instead it fetches the metadata of the datasets via the given URLs (address of the dataset host) and hence decides whether create a pipeline for a triple pattern to read the triples from the host. So, alternatively to the GRAPH modifier, we removed all the GRAPH clauses in the original queries and manually specify the datasets URLs as part of the query execution start-off command arguments. Another problem was that, some queries conduct the function projection in the SELECT clause, but this feature is not supported in the Triple Pattern Fragment client either. An alternative solution is using BIND keyword to bind the function results to a variable and then project this variable in the SELECT clause. Unfortunately, BIND feature is also not supported in the recent version of Triple Pattern Fragment client. So to ensure the experiment could continue, we extended the TPF client to support this feature while implementing the non-topological functions.

6.1.5 Performance Metrics

Triple pattern fragments adopted a complete different design against traditional RDF triplestore whose execution result is based on the sum of computational effort from both server side and client side, metrics applied in Geographica hence is not necessarily suitable for evaluating triple pattern fragments. Consequently, query duration time was considered as the only metric in this experiments to give an intuitively impression of how triple pattern fragments performs against GeoPSRAQL-enabled triplestore with reference to the last reported result published by Geographica. More effort would be spent on exploring the runtime characteristics of triple pattern fragments to acquire remarks for future improvement. To achieve this goal, the CPU and memory profiler would be installed in advance of the experiment and periodically make a record during the query execution. CPU profiler would tick the CPU time and report the usage statistic data in a text manner. Memory profiler can make a snapshot of the internal memory heap and accordingly list the biggest objects in a certain order. Comparing the memory snapshots can help find

out if the application is suffering from the memory leakage and give a standpoint where the program can be optimized in memory perspective.

6.1.6 Experiment Setup

For the experiment, we have setup a virtual machine running Debian OS version "Jessie" and allocated 8GB RAM, 2 virtual CPU cores on the SCSSNEBULA platform which is an elastic cloud service infrastructure hosted by Trinity College Dublin. To simulate the remote access environment, we setup a reverse proxy on the Triple Pattern Fragment server. In this experiment, we evaluate the following version of Triple Pattern Fragment server and Triple Pattern Fragment client where our GeoSPARQL extension was implemented.

- Linked Data Fragments Server version 2.2.2 (Node.js >= 4.0)
- Linked Data Fragments Client version 2.0.5 (Node.js >= 4.0)

As stated in the official documentation, Tripe Pattern Fragment server supports various kinds of datasource file format and able to initiate a couple of worker processes. From the performance point of view, HDT[29][47] was chosen as the default datasource format and two worker processes were initiated aside the main process. However, original datasets were published in N-Triple[21] format. A quick solution is manually convert the N-Triples file is to HDT file utilize rdf2hdt tool provided by[29][47]. In the appendix, we provided the final configuration file for the Triple Pattern Fragment Server. Each of the query would be executed for five times and pick up the average value as final result.

6.1.7 Benchmark Results

During the experiment, we executed queries for five times and record the highest and lowest observed value for each query and then calculated the average value as our final result. The final benchmark results are displayed in the Figure 6.2, Figure 6.3, Figure 6.4 and Figure 6.5

In the micro benchmark, every spatial selection query and non-topological queries is able to finish in a couple of minutes. These spatial selection queries are very simple queries that only test one function in one query. Hence, the result is effected mostly by the datasets size. For the spatial-join query, it is very hard to finish within a reasonable time. The last two aggregation queries are not supported, because the aggregation functions in these queries are defined under stSPARQL extension. The macro benchmark gives an intuitive impression of how the TPF performs in a real word scenario and we can observe that the query duration varies differently from each query. The fastest query would finish around ten seconds and the slowest query would take more than an hour to finish.

In the synthetic benchmark, we can observe that ,in general, the spatial-join query is much more "expensive" than the spatial selection query. That also happened in the real-world benchmark workload. More specifically perspective, for the spatial selection queries, there are three controlled variables in the benchmark: spatial functions, query selectivity and geometry size. For the first variable, geof:sfInteresects and geof:sfWithin, queries with the same selectivity and triple size but different spatial function have very close query duration. For the query with different selectivity, we observed huge performance difference. All the queries with selectivity 512 finished much faster than the queries with the selectivity of 1. The last variable geometry size, will not affect the query duration. For the spatial join queries, query selectivity of two datasets has played an important role in the performance. For example, queries whose dataset selectivity are both 512 is able to finish in seconds whereas the rest queries are not able to finish within a day.

6.1.8 Summary

In this experiment, we have tested our implementation with Geographica benchmark and it turns out that our implementation is working as expected. But, from the result set,

Name	Query Type	Query Duration(Max)	Query Duration(Min)	Query Duration(Avg)
Boundary_CLC	non-topological	93234.283	91742.381	92432.678ms
Envelope_CLC	non-topological	90183.912	85623.723	87271.975ms
ConvexHull_CLC	non-topological	97123.123	93129.469	96295.190ms
Buffer_GeoNames_2	non-topological	42938.604	40902.742	41789.390ms
Buffer_LGD_2	non-topological	9534.584	9212.238	9394.143ms
Area_CLC	non-topological	9384.395	9094.734	9117.097ms
Equals_LGD_GivenLine	spatial-selection	3284.639	2883.238	3001.341ms
Equals_GAG_GivenPolygon	spatial-selection	23749.684	18669.067	20461.527ms
Intersects_LGD_GivenPolygon	spatial-selection	4098.479	3497.846	3789.224ms
Intersects_CLC_GivenLine	spatial-selection	125786.396	109964.764	111487.346ms
Overlaps_CLC_GivenPolygon	spatial-selection	70989.789	65438.759	68661.178ms
Crosses_LGD_GivenLine	spatial-selection	7094.222	5099.374	6097.741ms
Within_GeoNames_GivenPolygon	spatial-selection	1898.123	1290.485	1430.099ms
Within_GeoNames_Point_Buffer	spatial-selection	51029.432	47223.248	48591.484ms
GeoNames_Point_Distance	spatial-selection	25748.836	20833.649	23367.913ms
Disjoint_GeoNames_GivenPolygon	spatial-selection	48920.645	42348.548	44270.685ms
Disjoint_LGD_GivenPolygon	spatial-selection	8099.637	7693.458	7722.866ms
Equals_GeoNames_DBPedia	spatial-join	>6h	>6h	>6h
Intersects_GeoNames_LGD	spatial-join	>6h	>6h	>6h
Intersects_GeoNames_GAG	spatial-join	>6h	>6h	>6h
Intersects_LGD_GAG	spatial-join	>6h	>6h	>6h
Within_GeoNames_GAG	spatial-join	>6h	>6h	>6h
Within_LGD_GAG	spatial-join	>6h	>6h	>6h
Within_CLC_GAG	spatial-join	>6h	>6h	>6h
Crosses_LGD_GAG	spatial-join	>6h	>6h	>6h
Touches_GAG_GAG	spatial-join	>6h	>6h	>6h
Overlaps_GAG_CLC	spatial-join	>6h	>6h	>6h
Extent_GAG	aggregation	not supported	not supported	not supported
Union_GAG	aggregation	not supported	not supported	not supported

Figure 6.2: real-world benchmark results (1)

Find_Closest_Populated_Place	reverse-geocoding	43827.874	39847.839	41540.838ms
Find_Closest_Motorway	reverse-geocoding	12348.984	10984.783	11964.361ms
Thematic_Search	map-search-and-browser	128993.823	120392.473	123667.067ms
Get_Around_POIs	map-search-and-browser	623489.765	597638.219	605448.186ms
Get_Around_Roads	map-search-and-browser	1497383.876	1468233.984	1478947.881ms
Get_CLC_areas	rapid-mapping	1684954.633	1603239.373	1642919.963ms
Get_highways	rapid-mapping	17832.837	15638.946	16710.921ms
Get_municipalities	rapid-mapping	17654.926	12894.846	15372.147ms
Get_hotspots	rapid-mapping	180394.735	168934.783	170652.558ms
Get_coniferous_forests_in_fire	rapid-mapping	>1h	>1h	>1h
Get_road_segments_affected_by_fire	rapid-mapping	>1h	>1h	>1h

Figure 6.3: real-world benchmark results cnt. (2)

Name	Query Type	Query Duration(Max)	Query Duration(Min)	Query Duration(Avg)
Synthetic_Selection_Intersects_1_1.0	spatial-selection	49593946.839ms	49579302.243ms	49584302.243ms
Synthetic_Selection_Intersects_512_1.0	spatial-selection	3583.8472ms	3048.293ms	3222.308ms
Synthetic_Selection_Intersects_1_0.75	spatial-selection	50637482.847ms	50593749.746ms	50615253.748ms
Synthetic_Selection_Intersects_512_0.75	spatial-selection	3027.739ms	2394.782ms	2563.182ms
Synthetic_Selection_Intersects_1_0.5	spatial-selection	49673884.382ms	49593923.648ms	49602277.531ms
Synthetic_Selection_Intersects_512_0.5	spatial-selection	3094.837ms	2578.849ms	2658.709ms
Synthetic_Selection_Intersects_1_0.25	spatial-selection	50662923.832ms	50639182.837ms	50643839.016ms
Synthetic_Selection_Intersects_512_0.25	spatial-selection	3084.836ms	2497.745ms	2701.976ms
Synthetic_Selection_Intersects_1_0.1	spatial-selection	50019323.478ms	49874284.649ms	50002671.006ms
Synthetic_Selection_Intersects_512_0.1	spatial-selection	3748.746ms	2573.083ms	2660.219ms
Synthetic_Selection_Intersects_1_0.001	spatial-selection	48647820.641ms	48608194.649ms	48625455.700ms
Synthetic_Selection_Intersects_512_0.001	spatial-selection	3128.462ms	2338.649ms	2718.028ms
Synthetic_Selection_Within_1_1.0	spatial-selection	50389237.740ms	50347283.469ms	50362432.791ms
Synthetic_Selection_Within_512_1.0	spatial-selection	3074.659ms	2673.846.537ms	2808.883ms
Synthetic_Selection_Within_1_0.75	spatial-selection	53589403.092ms	53508734.736ms	53535595.354ms
Synthetic_Selection_Within_512_0.75	spatial-selection	3278.987ms	2687.073ms	2871.658ms
Synthetic_Selection_Within_1_0.5	spatial-selection	54089763.672ms	59562340.647ms	54026457.775m
Synthetic_Selection_Within_512_0.5	spatial-selection	3238.648ms	2683.583ms	2880.122ms
Synthetic_Selection_Within_1_0.25	spatial-selection	53187639.482ms	53098793.692ms	53135162.726ms
Synthetic_Selection_Within_512_0.25	spatial-selection	2987.673ms	2678.349ms	2773.509ms
Synthetic_Selection_Within_1_0.1	spatial-selection	53038947.678ms	52998743.082ms	53019283.273ms
Synthetic_Selection_Within_512_0.1	spatial-selection	3089.632ms	2783.294ms	2960.701ms
Synthetic_Selection_Within_1_0.001	spatial-selection	50748936.193ms	50709384.603ms	50721833.172ms
Synthetic_Selection_Within_512_0.001	spatial-selection	3077.774ms	2789.391ms	2821.583ms
Synthetic_Join_Intersects_1_1	spatial-join	>24h	>24h	>24h
Synthetic_Join_Intersects_1_512	spatial-join	>24h	>24h	>24h
Synthetic_Join_Intersects_512_1	spatial-join	>24h	>24h	>24h
Synthetic_Join_Intersects_512_512	spatial-join	405893.472ms	398749.742ms	401743.043ms
Synthetic_Join_Touches_1_1	spatial-join	>24h	>24h	>24h
Synthetic_Join_Touches_1_512	spatial-join	>24h	>24h	>24h

Figure 6.4: synthetic benchmark results (1)

Synthetic_Join_Touches_512_1	spatial-join	>24h	>24h	>24h
Synthetic_Join_Touches_512_512	spatial-join	16892.087ms	14628.702ms	15208.998ms
Synthetic_Join_Within_1_1	spatial-join	>24h	>24h	>24h
Synthetic_Join_Within_1_512	spatial-join	>24h	>24h	>24h
Synthetic_Join_Within_512_1	spatial-join	>24h	>24h	>24h
Synthetic_Join_Within_512_512	spatial-join	409271.548ms	398921.764ms	404816.244ms

Figure 6.5: synthetic benchmark results cnt. (2)

we also observed that some queries would take a couple of hours to finish and we can not tell about how much system resource it has consumed during such period of time. Knowing such details are very crucial if we want running our implementation in browser environment. So, In the next section we would using the profile data(CPU, Memory) to give a deep view of the TPF client performance.

6.2 Performance analysis of TPF client

Testing our implementation with Geographica benchmark as discussed in the last section, indicated that our implementation functions as expected. Some of the queries in the Geographica benchmark take very long time to finish which is obviously not an acceptable situation for the user. For example, in macro benchmark, "Get_coniferous_forests_in_fire" query would take more than an hour to finish while time is the first concern in such fire situation. In this section, we provide a deeper view of the TPF client side query processing by analyzing the profile data collected in the previous benchmark test. This performance analysis consists of two parts, first part focuses on analyzing the memory consumption during the query execution. Second part focuses on analyzing the CPU time distribution. Last but not the least, we consider the analysis results together with the specific part of the code to provide some practical recommendations on how to form a performant query for the extended TPF client.

6.2.1 Analysis Setup

To undertake the analysis, we adopted node-inspector which is a popular Node.js debugger based on Blink Developer Tools. Node-inspector can help profile the memory as well as the CPU, also it can inspect the network client requests. In particular with the memory profile, the first memory heap snapshot would be created at the beginning of query execution, then after five minutes make the second memory heap snapshot. Compare and analyze the memory heap snapshot need special tools to cope with it. In this place,we

use the Chrome developer console to help conduct our analyze.

The following sections presents the analysis results obtained. For simplicity, we choose to only present the result of a simple BGP with only one triple pattern as it can help achieve a better understanding in comparison with a query of complex BGP. This query is shown in figure 6.6.

```
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
PREFIX datasets: <http://geographica.di.uoa.gr/dataset/>
PREFIX clc: <http://geo.linkedopendata.gr/corine/ontology#>

SELECT (geof:envelope(?o1) AS ?ret)
WHERE {
      {?s1 clc:asWKT ?o1}
}
```

Figure 6.6: envelop example for performance analyze

In this query, it fetches all the geometries from the target dataset and then calculates the envelope of these geometries and bind the result to ?ret variable. Hence the TPF client would create only one iterator for this situation and it would be easier to extract the characteristics of the query execution.

Besides that, we also set a reference group to compare the performance under local environment and remote environment. To simulate such remote environment, we set up a reverse proxy using the service provided ngrok. It would build a tunnel between our TPF server and ngrok server, as a result we can send our request to the ngrok server and the ngrok server would forward our request to the TPF server hence simulate the remote environment.

6.2.2 Memory analysis from external view

Before analyzing the complex memory heap, we would first investigate the memory consumption trend during the entire query execution. To provide a such external view of the memory. We utilized the build in process monitoring tools in the operating system. In Linux, this can be achieved by using top command. We provide the process id of TPF client and then use the pipe line command to record the statistics to a separate data file and then plot the chart.

As shown in the Figure 6.7, the y axis represents the internal memory of TPF client, x axis represents the query execution duration that measured in seconds. Series with short query duration was executed under local environment, another series indicate the remote environment. From this chart, we can conclude two points, first is the network traffic would significantly affect the query execution and the second observation is that the memory is growing in a linear way. We assume memory growth is related to the size of fetched triple from TPF server as the query only has one triple pattern. So, in the next section we would investigate the memory heap to try to find out some evidences to prove this assumption.

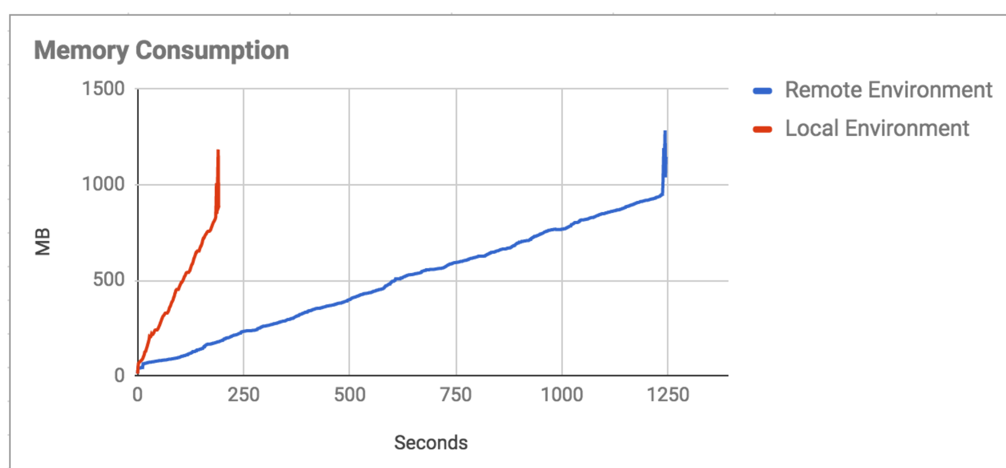


Figure 6.7: memory consumption trend of example query

6.2.3 Memory analysis from internal view

In this section, we have two memory heap snapshots with a time gap of five minutes. The comparison result of these two snapshots is shown in the table 6.7. Majority of the memory growth was contributed by the String. It takes up 99.3 percent of the memory increase. For the next step, we inspect the objects allocated between the two snapshots and particularly the string object list which is shown in the Figure 6.8. We observed that most of the new strings were geometry literals. So, we trace back the code find out where memories are allocated for the these geometries. The code trace is shown in the Figure 6.9. We observed all these geometries are reserved by the asyncIterator which is an iterator used in the TPF client to read the triples of a triple pattern.

	startup heap	heap after five minutes
Code	8942KB	9219KB
Strings	5285KB	378134KB
JS Arrays	179KB	426KB
Typed Arrays	27KB	27KB
System Objects	1352KB	1463KB
Total	30411KB	405855KB

Table 6.7: memory heap comparison of example query

To conclude, TPF client would create an iterator for each triple pattern, at the same time TPF client would create a cloned iterator of this iterator. In this cloned iterator, is has a HistoryReader that would store all the mapping values of this triple pattern. In this case, it stores all the geometries that fetched from TPF server. Code snippet of HistoryReader is shown in the Figure 6.10.

We also observed that, all the historical data are stored in an array which would not be released by the system until the iterator is released. In some cases, these historical data are never used or are partially used. In order to prove this point, we disabled the historical reader and repeated the example query, the new memory consumption chart is shown in the Figure 6.11. In this chart, TPF client with disabled HistoryReader has a

Summary ▼ Class filter Objects allocated between client_js and client_js2 ▼

Constructor	Distance	Objects Count		Shallow Size		Retained Size	
▼(string)	5	36 573	12 %	378 536 632	91 %	378 536 632	91 %
▶""<http://www.opengis.net/def/crs/EPSG/4326> MULTIPOLYGON (((24.752690015732718 41.392055270272792,24.752532871622659 41.391	16			1 163 776	0 %	1 163 776	0 %
▶""<http://www.opengis.net/def/crs/EPSG/4326> MULTIPOLYGON (((22.612307096713575 41.132378554699912,22.612105518330718 41.131	16			1 154 024	0 %	1 154 024	0 %
▶""<http://www.opengis.net/def/crs/EPSG/4326> MULTIPOLYGON (((24.752690015732718 41.392055270272792,24.752532871622659 41.391	16			1 153 480	0 %	1 153 480	0 %
▶""<http://www.opengis.net/def/crs/EPSG/4326> MULTIPOLYGON (((22.612307096713575 41.132378554699912,22.612105518330718 41.131	16			1 141 136	0 %	1 141 136	0 %
▶""<http://www.opengis.net/def/crs/EPSG/4326> POLYGON ((25.443151994398107 35.245824823690391,25.443252999861183 35.247056343	16			990 768	0 %	990 768	0 %
▶""<http://www.opengis.net/def/crs/EPSG/4326> POLYGON ((25.443151994398107 35.245824823690391,25.443252999861183 35.247056343	16			988 656	0 %	988 656	0 %
▶""<http://www.opengis.net/def/crs/EPSG/4326> POLYGON ((23.01524388900285 40.53776508364929,23.02343025797332 40.536674626875	16			659 736	0 %	659 736	0 %
▶""<http://www.opengis.net/def/crs/EPSG/4326> POLYGON ((23.01524388900285 40.53776508364929,23.02343025797332 40.536674626875	16			658 632	0 %	658 632	0 %
▶""<http://www.opengis.net/def/crs/EPSG/4326> POLYGON ((22.454945886317812 40.496508512055669,22.454098705470756 40.495721775	16			386 864	0 %	386 864	0 %
▶""<http://www.opengis.net/def/crs/EPSG/4326> POLYGON ((22.504872911288658 39.320872183881093,22.506975573706221 39.319216834	16			382 024	0 %	382 024	0 %
▶""<http://www.opengis.net/def/crs/EPSG/4326> POLYGON ((22.454945886317812 40.496508512055669,22.454098705470756 40.495721775	16			375 440	0 %	375 440	0 %
▶""<http://www.opengis.net/def/crs/EPSG/4326> POLYGON ((22.504872911288658 39.320872183881093,22.506975573706221 39.319216834	16			374 728	0 %	374 728	0 %
▶""<http://www.opengis.net/def/crs/EPSG/4326> MULTIPOLYGON (((25.301918045445767 41.248016956668906,25.300908491591652 41.246	16			371 808	0 %	371 808	0 %
▶""<http://www.opengis.net/def/crs/EPSG/4326> POLYGON ((24.542405820816573 41.262288308989277,24.546782881311447 41.260001655	16			366 688	0 %	366 688	0 %
▶""<http://www.opengis.net/def/crs/EPSG/4326> MULTIPOLYGON (((25.301918045445767 41.248016956668906,25.300908491591652 41.246	16			364 920	0 %	364 920	0 %
▶""<http://www.opengis.net/def/crs/EPSG/4326> POLYGON ((21.239341928022608 40.757573965825678,21.384276167932558 40.740879072	16			354 824	0 %	354 824	0 %
▶""<http://www.opengis.net/def/crs/EPSG/4326> POLYGON ((21.239341928022608 40.757573965825678,21.384276167932558 40.740879072	16			350 520	0 %	350 520	0 %
▶""<http://www.opengis.net/def/crs/EPSG/4326> MULTIPOLYGON (((20.40143038110876 39.834752299711781,20.40142921155368 39.83475	16			350 264	0 %	350 264	0 %
▶""<http://www.opengis.net/def/crs/EPSG/4326> POLYGON ((24.542405820816573 41.262288308989277,24.546782881311447 41.260001655	16			347 136	0 %	347 136	0 %
▶""<http://www.opengis.net/def/crs/EPSG/4326> MULTIPOLYGON (((20.40143038110876 39.834752299711781,20.40142921155368 39.83475	16			329 368	0 %	329 368	0 %
▶""<http://www.opengis.net/def/crs/EPSG/4326> POLYGON ((22.741012741856185 37.735170334939546,22.741509222265655 37.734314359	16			322 928	0 %	322 928	0 %
▶""<http://www.opengis.net/def/crs/EPSG/4326> MULTIPOLYGON (((22.327333776385085 41.153186304070296,22.327169925765844 41.152	16			322 888	0 %	322 888	0 %
▶""<http://www.opengis.net/def/crs/EPSG/4326> POLYGON ((22.44335143155082 39.669477666117935,22.44373646319729 39.66941087417	16			322 712	0 %	322 712	0 %

Figure 6.8: String object list in

Constructor	Distance	Objects Count	Shallow Size		Retained Size	
▶map :: system / Map (OneByteString) @101	4		88	0 %	88	0 %
▶""<http://www.opengis.net/def/crs/EPSG/4326> POLYGON ((25.443151994398107 35.245824823690391,25.443252999861183 35.247056343	16		988 656	0 %	988 656	0 %
▶""<http://www.opengis.net/def/crs/EPSG/4326> POLYGON ((23.01524388900285 40.53776508364929,23.02343025797332 40.536674626875	16		659 736	0 %	659 736	0 %
▶""<http://www.opengis.net/def/crs/EPSG/4326> POLYGON ((23.01524388900285 40.53776508364929,23.02343025797332 40.536674626875	16		658 632	0 %	658 632	0 %
▶""<http://www.opengis.net/def/crs/EPSG/4326> POLYGON ((22.454945886317812 40.496508512055669,22.454098705470756 40.495721775	16		386 864	0 %	386 864	0 %
▶""<http://www.opengis.net/def/crs/EPSG/4326> POLYGON ((22.504872911288658 39.320872183881093,22.506975573706221 39.319216834	16		382 024	0 %	382 024	0 %
▶""<http://www.opengis.net/def/crs/EPSG/4326> POLYGON ((22.454945886317812 40.496508512055669,22.454098705470756 40.495721775	16		375 440	0 %	375 440	0 %

Retainers

Object	Distance	Shallow Size		Retained Size	
▼parent in (sliced string) @421407	15	40	0 %	40	0 %
▼predicate in @332125	14	56	0 %	496	0 %
▼[5166] in Array @98533	13	32	0 %	383 972 648	92 %
▼history in system / Context @98519	12	96	0 %	384 109 168	92 %
▼77 in @373641	11	3 616	0 %	11 264	0 %
▼code in *readAndTransformSimple()* @117017	10	72	0 %	11 336	0 %
▼readAndTransformSimple in system / Context @116945	9	344	0 %	3 016	0 %
▼context in *AsyncIterator()* @80561	8	72	0 %	264	0 %
▼exports in Module @70565	7	80	0 %	1 064	0 %
▼/Users/momoca/Documents/Dissertation/Client.js/node_modules/asynciterator/asynciterator.js in @20313	6	24	0 %	6 312	0 %
▼_cache in *Module()* @67181	5	72	0 %	287 704	0 %
▼constructor in @4679	4	56	0 %	56	0 %
▼__proto__ in Module @20309	3	80	0 %	1 232	0 %
▼mainModule in process @923	2	24	0 %	16 116	0 %
▶process in @599	1	48	0 %	414 007 376	100 %
▶value in system / PropertyCell @45707	3	32	0 %	256	0 %
▶process in system / Context @44285	3	128	0 %	720	0 %
▶32 in @321737	9	896	0 %	2 064	0 %
▶27 in @321737	9	896	0 %	2 064	0 %
▶15 in @70133	9	416	0 %	904	0 %

Figure 6.9: function trace of geometry literal

```
function HistoryReader(source) {
 var history = [], clones;

 // Tries to read the item at the given history position
 this.readAt = function (pos) {
   var item = null;
   // Retrieve an item from history when available
   if (pos < history.length) {
     item = history[pos];
   }
   // Read a new item from the source when possible
   else if (!source.ended && (item = source.read()) !== null)
     history[pos] = item;
   return item;
 };

 // Determines whether the given position is the end of the source
 this.endsAt = function (pos) {
   return pos === history.length && source.ended;
 };
 // Registers a clone for history updates
 this.register = function (clone) {
   clones && clones.push(clone);
 };

 // Unregisters a clone for history updates
 this.unregister = function (clone) {
   var cloneIndex;
   if (clones && (cloneIndex = clones.indexOf(clone)) >= 0) {
     //console.log("-"+AsyncIterator.count--);
     clones.splice(cloneIndex, 1);
   }
 };
}
```

Figure 6.10: code snippet of historical reader

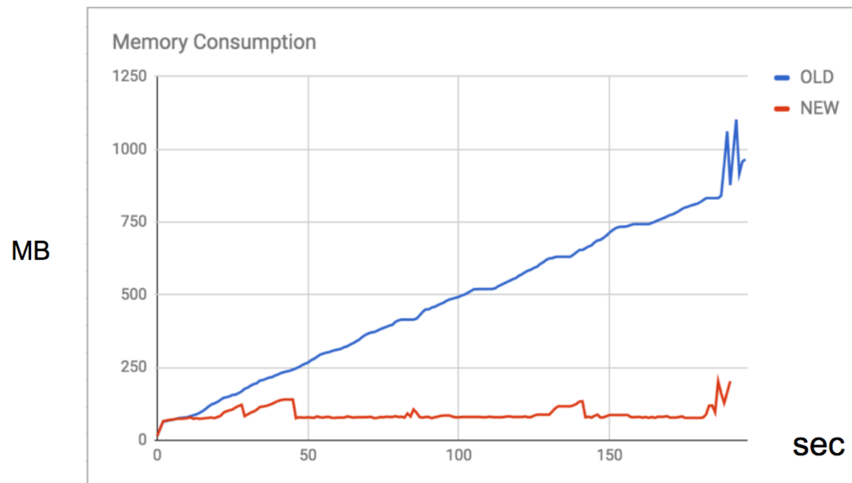


Figure 6.11: code snippet of historical reader

very small steady memory consumption around 75 MB.

6.2.4 CPU analysis from external view

This section focuses on analyzing the CPU consumption of TPF client. Node-inspector uses the CPU tick as a basic unit to generate the profile the CPU usage. The first of our concern is where these CPU time were spent. We set up four reference group in the experiment. First query was executed in the remote environment, for the second query, we removed the envelope function and executed it again in the remote environment. Same process would be repeated again in the local environment. Result is shown in the Figure 6.12. From this experiment we can conclude that, most of the CPU time were spent on handling network traffic and spatial calculation. Next section, we adopted the CPU profiler analyze tool to provide more accurate data to this respect.

6.2.5 CPU analysis from internal view

Figure 6.13 provides a top-down view of the CPU tick statistic. From this view, we observed the top call was made by the asyncIterator, this call would invoke the read the values from the buffered iterator and then transform the triples which is actually reading

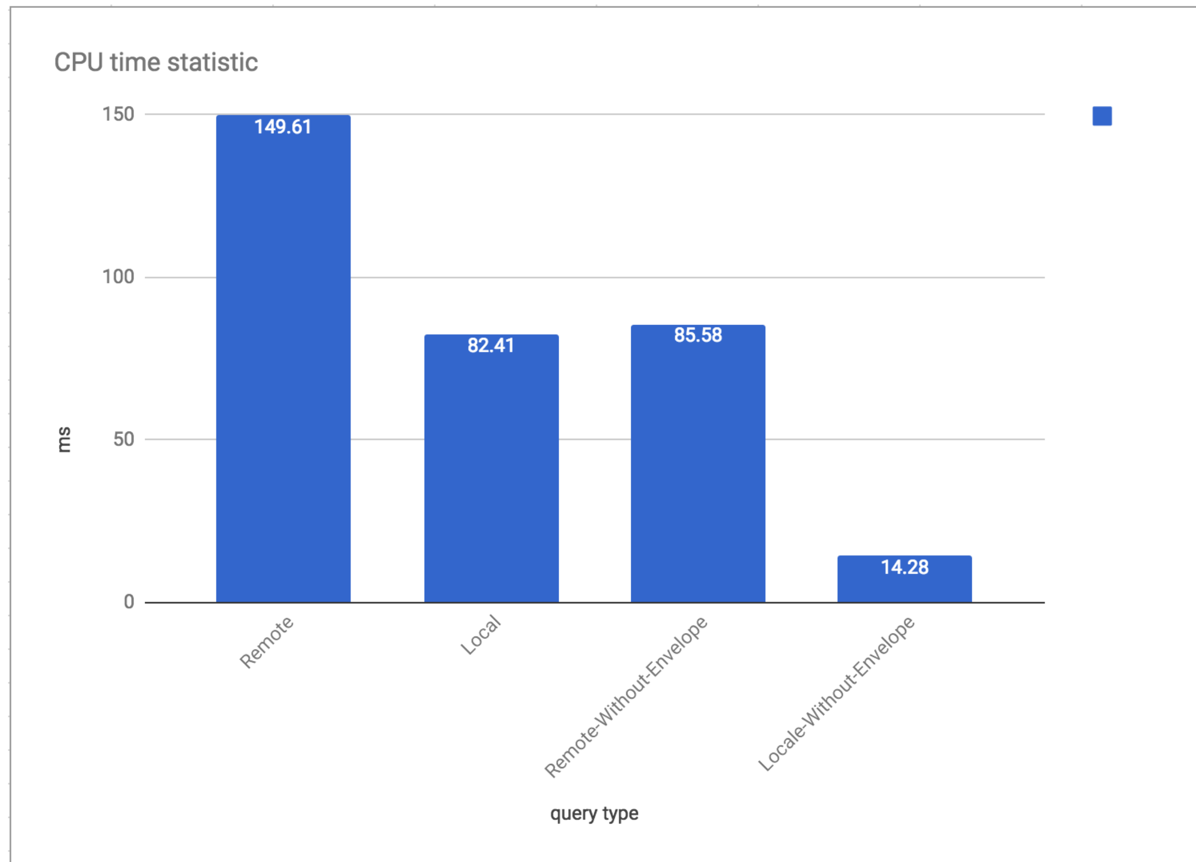


Figure 6.12: CPU time statistic of different query

the geometry literal and then pass it to the sparqlexpressionevaluator. In general, Iteration of triples take up 32.9 percent of the entire CPU time and sparqlExpressionEvaluator take up 23.8 of the entire CPU time. That proved most of the CPU were spent on the spatial calculation and handing the network traffic.

6.2.6 Summary

From the TPF client performance analysis we can conclude that the key factor that restricts performance of query execution is the number of triples fetched from the server. That is because, a smaller number of triples reduces the network traffic as well as the spatial calculation workload which would result in a better result. Second observation is that if the TPF client can provide a better historical data management in the iterator, the

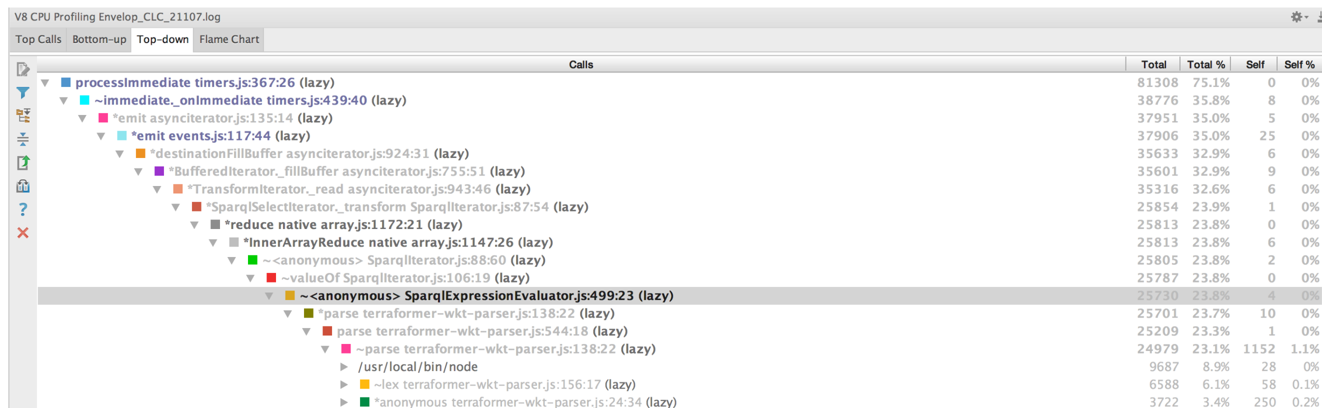


Figure 6.13: accurate CPU time statistic

internal memory consumption during the query execution would be optimized effectively.

6.2.7 Query Formulation Recommendations

As concluded from the previous section, the key thing to achieve a better query result in TPF client is reducing the number of necessary triples transmitted from TPF server. That is decided by the query evaluation strategy of the TPF client. Current version of the TPF client adopts a greedy strategy to evaluate the BGP. Greedy strategy was driven by the local optimal selection but will not guarantee a global optimal solution. We observe a carefully formed query can significantly improve the performance in TPF client.

Our key recommendation for users of the extended TPF client would be uplift any filter as much as possible to split the connected BGP into separate sub-BGPs. This is recommended as the filter operator reduces the solution mappings for next triple pattern. We outline an example to illustrate this observation.

Figure 6.14 and Figure 6.15are two equivalent queries that would return all the relevant information required, the points that intersects with a polygon. The second query puts the filter in the middle of a BGP which would split the query into two sub-BGPs. After TPF client evaluates the first BGP, filter operation would screen out all the geome-

```
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX dataset: <http://geographica.di.uoa.gr/dataset/>
PREFIX geonames: <http://www.geonames.org/ontology#>

SELECT ?f ?name ?fGeo ?code ?parent ?class ?fGeoWKT
WHERE {


            ?f geonames:hasGeometry ?fGeo.
            ?fGeo geonames:asWKT ?fGeoWKT.

            ?f geonames:name ?name.
            ?f geonames:featureCode ?code.
            ?f geonames:parentFeature ?parent.
            ?f geonames:featureClass ?class.


            FILTER(geof:sfIntersects(?fGeoWKT,
"POLYGON((21.89678192138672 38.14017245062123,21.983642578125
38.14017245062123,21.983642578125 38.120187931317325,21.89678192138672
38.120187931317325,21.89678192138672
38.14017245062123))"^^geo:wktLiteral)).


}
```

Figure 6.14: Get around POI (a)

tries that are not intersecting with a given polygon. As a result, less values would be bound to the rest triple patterns and hence the performance is improved.

Table 6.8 shows that the query performance has a significant improvement for this example. The total HTTP request reduced by 79.8% and query duration has reduced by 87.9%.

	HTTP request number	Query Duration
Original Query	110160	605448.186ms
Query with uplifted filter	22235	73184.333ms
Reduced By(%)	79.8	87.9

Table 6.8: performance comparison of equivalent query

```
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX dataset: <http://geographica.di.uoa.gr/dataset/>
PREFIX geonames: <http://www.geonames.org/ontology#>

SELECT ?f ?name ?fGeo ?code ?parent ?class ?fGeoWKT
WHERE {


            ?f geonames:hasGeometry ?fGeo.
            ?fGeo geonames:asWKT ?fGeoWKT.

            FILTER(geof:sfIntersects(?fGeoWKT,
"POLYGON((21.89678192138672 38.14017245062123,21.983642578125
38.14017245062123,21.983642578125 38.120187931317325,21.89678192138672
38.120187931317325,21.89678192138672
38.14017245062123))"^^geo:wktLiteral)).

            ?f geonames:name ?name.
            ?f geonames:featureCode ?code.
            ?f geonames:parentFeature ?parent.
            ?f geonames:featureClass ?class.

}
```

Figure 6.15: et around POI (b)

Chapter 7

Conclusions

This chapter discusses the main contribution in this dissertation and evaluate the GeoSPARQL extension from both engineer perspective and conceptual perspective. Last but not the least, this chapter also proposed some future work options and suggestions.

7.1 Conclusion

The main objective in this dissertation is enable the GeoSPARQL query on the TPF client. In our implementation, we choose to support the spatial filter functions defined in the Geometry Component as well as the spatial analysis functions defined in the Geometry topology components. The implementation process was driven by the Geographica benchmark. It finally proves that it is feasible to extend the TPF client to support GeoSPARQL query.

The second objective in this dissertation is test our extension with the Geographica benchmark. The test result shows that our implementation is work as expected but it takes more time for TPF client to finish a GeoSPARQL query than a bespoke spatial triple store. Our extension enables the server to handle the spatial data at a minimal cost and provided the client with great flexibility to explore a efficient way to finish the

GeoSPARQL query.

Consequently, we choose to conduct a performance analysis in order to find out the performance restriction of the TPF client and hence make some suggestions based on the observation. According to our analysis, a better historical data management in the TPF client iterator could help optimized the run-time internal memory consumption and better BGP evaluation strategy could help reduce the network traffic by reducing the triples that need to be transmitted to the client and finally improve the TPF client performance. We also recommend the user to uplift the filter function to separate the BGP manually which would help improve the performance.

To conclude, idea of extending the TPF client to support GeoSPARQL is a feasible as a new option for consuming the spatial data. It reduced the server cost to host a public knowledge graph which in return would encourage the organizations to publish the public available knowledge graph and fully utilized client resource. More important, the idea of TPF enables the client to split the query and explore an efficient approach to execute the both SPARQL and GeoSPARQL query which has great potential for future improvement.

7.2 Future work

From this dissertation, we realized how important the BGP evaluation strategy is for the TPF client. Current version of the TPF client adopted greedy strategy to evaluate the BGP, it works fine with most of the cases, but for some special cases, it won't yield a global optimal solution which in return would slower the query execution. Future work could investigate the potential BGP evaluation strategy. In addition to this, future work could also design a better data structure as well as a better historical data management strategy for the iterator in TPF client which would help reduce the memory consumption during the query consumption.

Appendix A

Abbreviations

Short Term	Expanded Term
W3C	World Wide Web Consortium
ISO	International Organization for Standardization
RDF	Resource Description Framework
RDFS	Resource Description Framework Scheme
OWL	Web Ontology Language
WKT	Well Known Text
SPARQL	SPARQL Protocol and RDF Query Language
DE-9IM	Dimensionally Extended nine-Intersection Model
GeoSPARQL	Geographic query language for RDF data
OGC	Open Geospatial Consortium
OSI	Ordnance Survey Ireland
LGD	LinkedGeoData
GAG	Greek Administrative Geography
CLC	CORINE Land Use/Land Cover
NOA	National Observatory of Athens

Short Term	**Expanded Term**
HDT	Header Dictionary Triples
RAM	Random Access Memory
CPU	Central Processing Unit
OS	Operating System
URI	Uniform Resource Identifier
URL	Uniform Resource Locater
HTML	Hyper-Text Markup Language
HTTP	Hyper-Text Transfer Protocol
XML	Extensible Markup Language
TPF	Triple Pattern Fragment
WKT	Well-Known-Text
GML	Geography Markup Language
BGP	Basic Group Pattern
CRS	Coordinate Reference System
SRS	Spatial Reference System
JSON	JavaScript Object Notation
JSON-LD	JSON for Linked Data
LDF	Linked Data Fragment

Appendix B

Parsed SPARQL JavaScript Object Example


```
{
  "queryType": "SELECT",
  "variables": [
    "?c1l",
    "?c2l"
  ],
  "where": [
    {
      "type": "bgp",
      "triples": [
        {
          "subject": "?c1",
          "predicate": "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
          "object": "http://ontologies.geohive.ie/osi#County"
        },
        {
```

```
            "subject": "?c1",
            "predicate": "http://www.w3.org/2000/01/rdf-schema#label",
            "object": "?c1l"
          },
          {
            "subject": "?c1",
            "predicate": "http://www.opengis.net/ont/geosparql#hasGeometry"
            "object": "?g1"
          },
          {
            "subject": "?c2",
            "predicate": "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
            "object": "http://ontologies.geohive.ie/osi#County"
          },
          {
            "subject": "?c2",
            "predicate": "http://www.w3.org/2000/01/rdf-schema#label",
            "object": "?c2l"
          },
          {
            "subject": "?c2",
            "predicate": "http://www.opengis.net/ont/geosparql#hasGeometry"
            "object": "?g2"
          }
        ]
      },
      {
```

```
    "type": "filter",
    "expression": {
      "type": "operation",
      "operator": "!=",
      "args": [
        "?c1",
        "?c2"
      ]
    }
  },
  {
    "type": "filter",
    "expression": {
      "type": "operation",
      "operator": "langmatches",
      "args": [
        {
          "type": "operation",
          "operator": "lang",
          "args": [
            "?c1l"
          ]
        },
        "\"en\""
      ]
    }
  },
```

```
{
  "type": "filter",
  "expression": {
    "type": "operation",
    "operator": "langmatches",
    "args": [
      {
        "type": "operation",
        "operator": "lang",
        "args": [
          "?c2l"
        ]
      },
      "\"en\""
    ]
  }
},
{
  "type": "bgp",
  "triples": [
    {
      "subject": "?g1",
      "predicate": "http://www.opengis.net/ont/geosparql#asWKT",
      "object": "?w1"
    },
    {
      "subject": "?g2",
```

```
          "predicate": "http://www.opengis.net/ont/geosparql#asWKT",
          "object": "?w2"
        }
      ]
    },
    {
      "type": "filter",
      "expression": {
        "type": "functionCall",
        "function": "http://www.opengis.net/def/function/geosparql/sfTouc
        "args": [
          "?w1",
          "?w2"
        ],
        "distinct": false
      }
    }
  ],
  "limit": 15,
  "type": "query",
  "prefixes": {
    "geo": "http://www.opengis.net/ont/geosparql#",
    "geof": "http://www.opengis.net/def/function/geosparql/",
    "osi": "http://ontologies.geohive.ie/osi#"
  }
}
```

Appendix C

TPF Server Configuration File

```
{
  "title": "Geo Project",
  "datasources": {

    "boundaries-counties": {
      "title": "Boundaries -- counties",
      "type": "HdtDatasource",
      "settings": {
        "file": "./data/2016-11-16-county.hdt"
      }
    },
    "boundaries-electoral-divisions": {
      "title": "Boundaries -- electoral divisioins",
      "type": "HdtDatasource",
      "settings": {
        "file": "./data/2016-11-16-ed.hdt"
      }
    },
```

```
"Greek-Administrative-Geography-Dataset": {
  "tittle": "Greek Administrative Geography Dataset",
  "type": "HdtDatasource",
  "settings": {
    "file": "./data/gag.hdt"
  }
},

"CORINE-Land-Use/Land-Cover-Dataset": {
  "tittle": "CORINE Land Use/Land Cover Dataset",
  "type": "HdtDatasource",
  "settings": {
    "file": "./data/corine.hdt"
  }
},

"LinkedGeoData-Dataset": {
  "tittle": "LinkedGeoData Dataset",
  "type": "HdtDatasource",
  "settings": {
    "file": "./data/linkedgeodata.hdt"
  }
},
"GeoNames-Dataset": {
  "tittle": "GeoNames Dataset",
  "type": "TurtleDatasource",
```

```
      "settings": {
        "file": "./data/geonames.ttl"
      }
    },
    "Hotspots-Dataset": {
      "tittle": "Hotspots Dataset",
      "type": "HdtDatasource",
      "settings": {
        "file": "./data/hotspots.hdt"
      }
    },
    "synthetic-dataset": {
      "title": "synthetic -- dataset",
      "type": "HdtDatasource",
      "settings": {
        "file": "./data/synthetic.hdt"
      }
    },
    "DBpedia-Dataset": {
      "tittle": "DBpedia Dataset",
      "type": "TurtleDatasource",
      "settings": {
        "file": "./data/dbpedia.ttl"
      }
    }

  },
```

```
  "prefixes": {
    "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
    "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
    "xsd": "http://www.w3.org/2001/XMLSchema#",
    "dc": "http://purl.org/dc/terms/",
    "foaf": "http://xmlns.com/foaf/0.1/",
    "dbpedia": "http://dbpedia.org/resource/",
    "dbpedia-owl": "http://dbpedia.org/ontology/",
    "dbpprop": "http://dbpedia.org/property/",
    "hydra": "http://www.w3.org/ns/hydra/core#",
    "void": "http://rdfs.org/ns/void#",
    "geo": "http://www.opengis.net/ont/geosparql#"
  }
}
```

Appendix D

Supported Feature in TPF Client v2.0.5

Feature	
Basic Group Pattern	✓
Group Graph Pattern	✓
Filter	✓
REGEX Operator	✓
OPTIONAL Operator	✓
UNION Operator	✓
Negation	
Property Path	✓
BIND Operator	
VALUES Operator	
GROUP BY Modifier	✓
ORDER BY Modifier	✓
HAVING Modifier	
LIMIT Modifier	✓
OFFSET Modifier	✓
DINSTINCT Modifier	✓
REDUCED Modifier	✓
Aggregates	
Aggregation Projection	✓
Subqueries	
GRAPH Operator	
ASK	✓
CONSTRUCT	✓
DESCRIBE	✓

Table D.1: Feature Expressed in GeoGraphica Benchmark Queries

Bibliography

[1] About. http://linkedgeodata.org/.

[2] Chapter 4. using postgis: Data management and queries. http://www.postgis.org/docs/using-postgis-dbmanagement.html/DE-9IM.

[3] Corine land cover of greece. http://www.linkedopendata.gr/dataset/corine-land-cover-of-greece.

[4] Greek administrative geography. http://www.linkedopendata.gr/dataset/greek-administrative-geography.

[5] Openstreetmap. http://www.openstreetmap.org/.

[6] Parliament. http://parliament.semwebcentral.org/.

[7] Query linked data on the web. http://client.linkeddatafragments.org/.

[8] Terraformer. http://terraformer.io/getting-started/.

[9] Turf.js — getting started. http://turfjs.org/getting-started/.

[10] De-9im. https://en.wikipedia.org/wiki/DE-9IM/cite-note-PostGISch4-7, Aug 2017.

[11] Jans Aasman. Allegro graph: Rdf triple database. *Cidade: Oakland Franz Incorporated*, 2006.

[12] Tim Ambler and Nicholas Cloud. Browserify. In *JavaScript Frameworks for Modern Web Dev*, pages 101–120. Springer, 2015.

[13] Grigoris Antoniou and Frank Van Harmelen. *A semantic web primer*. MIT press, 2004.

[14] Spiros Athanasiou, Daniel Hladky, Giorgos Giannopoulos, Alejandra Garcia Rojas, and Jens Lehmann. Geoknow: Making the web an exploratory place for geospatial knowledge. *ERCIM News*, 96:12–13, 2014.

[15] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. *The semantic web*, pages 722–735, 2007.

[16] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[17] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data-the story so far. *Semantic services, interoperability and web applications: emerging concepts*, pages 205–227, 2009.

[18] Harold Boley and Michael Kifer. Rif core design. *W3C Working Draft, March*, 30:2007, 2007.

[19] D Brickley. W3c semantic web interest group: Basic geo (wgs84 lat/long) vocabulary. *edited by D. Brickley*, page W3C, 2006.

[20] Carlos Buil-Aranda, Aidan Hogan, Jürgen Umbrich, and Pierre-Yves Vandenbussche. Sparql web-querying infrastructure: Ready for action? In *International Semantic Web Conference*, pages 277–293. Springer, 2013.

[21] Gavin Carothers and Andy Seaborne. Rdf 1.1 n-triples: A line-based syntax for an rdf graph. *World Wide Web Consortium. http:// www. w3. org/ TR/ n-triples/. Accessed*, 24, 2014.

[22] Artem Chebotko, Shiyong Lu, Hasan M Jamil, and Farshad Fotouhi. Semantics preserving sparql-to-sql query translation for optional graph patterns. *Wayne State University, Tech. Rep. TR-DB-052006-CLJF*, 2006.

[23] Chrdebru. chrdebru/client.js. https://github.com/chrdebru/Client.js/blob/master/queries-osi/irish-counties-touching-labels.sparql.

[24] World Wide Web Consortium et al. Rdf 1.1 concepts and abstract syntax. 2014.

[25] Gianluca Correndo, Manuel Salvadores, Ian Millard, Hugh Glaser, and Nigel Shadbolt. Sparql query rewriting for implementing data integration over linked data. In *Proceedings of the 2010 EDBT/ICDT Workshops*, page 4. ACM, 2010.

[26] Max J Egenhofer and Robert D Franzosa. Point-set topological spatial relations. *International Journal of Geographical Information System*, 5(2):161–174, 1991.

[27] Ivan Ermilov, Michael Martin, Jens Lehmann, and Sören Auer. *Linked Open Data Statistics: Collection and Exploitation*, pages 242–249. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[28] Dieter Fensel and Mark A Musen. The semantic web: a brain for humankind. *IEEE Intelligent systems*, 16(2):24–25, 2001.

[29] Javier D. Ferndez, Miguel A. Martnez-Prieto, Claudio Gutirrez, Axel Polleres, and Mario Arias. Binary rdf representation for publication and exchange (hdt). *Web Semantics: Science, Services and Agents on the World Wide Web*, 19:2241, 2013.

[30] George Garbis, Kostis Kyzirakos, and Manolis Koubarakis. Geographica: A benchmark for geospatial rdf stores (long version). In *International Semantic Web Conference*, pages 343–359. Springer, 2013.

[31] Geohive. Geohive - spatial data made easy. http://www.geohive.ie/.

[32] Sean Gillies, H Butler, M Daly, A Doyle, and T Schaub. The geojson format. *coordinates*, 102:0–5, 2016.

[33] Thomas R Gruber. Toward principles for the design of ontologies used for knowledge sharing? *International journal of human-computer studies*, 43(5-6):907–928, 1995.

[34] Alon Halevy, Peter Norvig, and Fernando Pereira. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(2):8–12, 2009.

[35] Steve Harris, Andy Seaborne, and Eric Prudhommeaux. Sparql 1.1 query language. *W3C recommendation*, 21(10), 2013.

[36] Olaf Hartig. An overview on execution strategies for linked data queries. *Datenbank-Spektrum*, 13(2):89–99, 2013.

[37] Olaf Hartig and Ralf Heese. The sparql query graph model for query optimization. *The Semantic Web: Research and Applications*, pages 564–578, 2007.

[38] Olaf Hartig, Ian Letter, and Jorge Pérez. A formal framework for comparing linked data fragments.

[39] Tom Heath and Christian Bizer. Linked data: Evolving the web into a global data space. *Synthesis lectures on the semantic web: theory and technology*, 1(1):1–136, 2011.

[40] Ordnance Survey Ireland. Adapt collaborates with ordnance survey ireland. https://www.osi.ie/news/adapt-collaborates-with-ordnance-survey-ireland/.

[41] Apache Jena. Apache jena. *jena. apache. org [Online]. Available: http://jena. apache. org [Accessed: Mar. 20, 2014]*, page 14, 2013.

[42] Michel Klein. Xml, rdf, and relatives. *IEEE Intelligent Systems*, 16(2):26–28, 2001.

[43] Dave Kolas. A benchmark for spatial semantic web systems. In *International Workshop on Scalable Semantic Web Knowledge Base Systems*, 2008.

[44] Manolis Koubarakis and Kostis Kyzirakos. Modeling and querying metadata in the semantic sensor web: The model strdf and the query language stsparql. *The semantic web: research and applications*, pages 425–439, 2010.

[45] Kostis Kyzirakos, Manos Karpathiotakis, and Manolis Koubarakis. Strabon: a semantic geospatial dbms. *The Semantic Web–ISWC 2012*, pages 295–311, 2012.

[46] LinkedDataFragments. Linkeddatafragments/client.js, Jul 2017.

[47] Miguel A. Martnez-Prieto, Mario Arias, and Javier D. Fernndez. Exchange and consumption of huge rdf data. In *The Semantic Web: Research and Applications*, pages 437–452. Springer, 2012.

[48] Brian Matthews. Semantic web technologies. *E-learning*, 6(6):8, 2005.

[49] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. In *International semantic web conference*, pages 30–43. Springer, 2006.

[50] Matthew Perry and John Herring. Ogc geosparql-a geographic query language for rdf data. *OGC Implementation Standard. Sept*, 2012.

[51] Axel Polleres, Aidan Hogan, Renaud Delbru, and Jürgen Umbrich. Rdfs and owl reasoning for linked data. In *Reasoning Web. Semantic Technologies for Intelligent Data Access*, pages 91–149. Springer, 2013.

[52] Eric Prud, Andy Seaborne, et al. Sparql query language for rdf. 2006.

[53] David A Randell, Zhan Cui, and Anthony G Cohn. A spatial logic based on regions and connection. *KR*, 92:165–176, 1992.

[54] Suprio Ray, Bogdan Simion, and Angela Demke Brown. Jackpine: A benchmark to evaluate spatial database performance. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 1139–1150. IEEE, 2011.

[55] XUL RDFS. Resource description framework pdf. 2004.

[56] Carl Reed, R Singh, R Lake, J Lieberman, and M Maron. An introduction to georss: A standards based approach for geo-enabling rss feeds. *White Paper OGC*, 2006.

[57] Markus Stocker, Andy Seaborne, Abraham Bernstein, Christoph Kiefer, and Dave Reynolds. Sparql basic graph pattern optimization using selectivity estimation. In *Proceedings of the 17th international conference on World Wide Web*, pages 595–604. ACM, 2008.

[58] Christian Strobl. Dimensionally extended nine-intersection model (de-9im). In *Encyclopedia of GIS*, pages 240–245. Springer, 2008.

[59] Joachim Van Herwegen, Laurens De Vocht, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. Substring filtering for low-cost linked data interfaces. In *International Semantic Web Conference*, pages 128–143. Springer, 2015.

[60] 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. *Web Semantics: Science, Services and Agents on the World Wide Web*, 37:184–206, 2016.