NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION

BSc THESIS

A parallel implementation of the tool GeoTriples using Apache Spark

Nikolaos G. Trapalis

Supervisors: Manolis Koubarakis, Professor UoA
George Stamoulis, Ph.D. Candidate UoA

ATHENS

October 2018

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Μία παράλληλη υλοποίηση του εργαλείου GeoTriples χρησιμοποιώντας το Apache Spark

Νικόλαος Γ. Τράπαλης

Επιβλέποντες: **Μανώλης Κουμπαράκης,** Καθηγητής Ε.Κ.Π.Α.
Γεώργιος Σταμούλης,Υποψήφιος για Ph.D. Ε.Κ.Π.Α.

ΑΘΗΝΑ

Οκτώβριος 2018

BSc THESIS

A parallel implementation of the tool GeoTriples using Apache Spark

Nikolaos G. Trapalis
S.N.: 1115201200180

SUPERVISORS: **Manolis Koubarakis,** Professor UoA
George Stamoulis, Ph.D. Candidate UoA

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Μία παράλληλη υλοποίηση του εργαλείου GeoTriples χρησιμοποιώντας το Apache Spark

Νικόλαος Γ. Τράπαλης
Α.Μ.: 1115201200180

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Μανώλης Κουμπαράκης,** Καθηγητής Ε.Κ.Π.Α.
Γεώργιος Σταμούλης,Υποψήφιος για Ph.D. Ε.Κ.Π.Α.

ABSTRACT

A plethora of Earth Observation data that is becoming available at no charge in Europe and the US recently reflects the strong push for more open Earth Observation data. Linked data is a paradigm which studies how one can make data available on the Web, and interconnect it with other data with the aim of making the value of the resulting “Web of data” greater than the sum of its parts. Open Earth Observation data that are currently made available by space agencies such as ESA and NASA are not following the linked data paradigm. Therefore, Earth Observation data and other kinds of geospatial data that are necessary for a user to satisfy her information needs can only be found in different data silos, where each silo may contain only part of the needed data. Publishing the content of these silos as RDF graphs, enables the development of data analytics applications with great environmental and financial value.

GeoTriples is a semi-automated tool that allows the publication of geospatial information into an RDF graph using the state of the art vocabularies like GeoSPARQL and stSPARQL, but at the same time it is not tightly coupled to a specific vocabulary. In this thesis we present a parallel implementation of the tool GeoTriples utilizing Apache Spark. Using the MapReduce technique we provide noticable improvement in time when dealing with large datasets.

SUBJECT AREA:Semantic Web,Data Transformation

KEYWORDS: MapReduce,Linked Data,RDF,GeoTriples,Shapefile

.

ΠΕΡΙΛΗΨΗ

Η πληθώρα γεωχωρικών δεδομένων που γίνονται διαθέσιμα δωρεάν στην Ευρώπη και τις Η.Π.Α. δείχνει την προσπάθεια για περισσότερα ελεύθερα γεωχωρικά δεδομένα.Τα διασυνδεδεμένα δεδομένα αποτελούν την προσπάθεια για διαθεσιμότητα δεδομένων στο Διαδίκτυο και τη διασύνδεση αυτών με άλλα δεδομένα ,ώστε η αξία του τελικού αποτελέσματος να είναι μεγαλύτερη από ένα απλό άθροισμα των δεδομένων που το αποτελούν.Τα γεωχωρικά δεδομένα που έχουν γίνει διαθέσιμα από διαστημικά πρακτορεία όπως ESA και NASA δεν ακολουθούν τη λογική των διασυνδεδεμένων δεδομένων.Επομένως ένας χρήστης που αναζητά γεωχωρικά δεδομένα μπορεί να τα βρει μόνο σε μεμονωμένες αποθήκες δεδομένων που μπορεί να περιέχουν ένα μέρος των δεδομένων που χρειάζεται.Η δημοσίευση των περιεχομένων αυτών των αποθηκών σε μορφή γράφων RDF καθιστά δυνατή την ανάπτυξη εφαρμογών ανάλυσης δεδομένων με γιγάντια περιβαντολλογική και οικονομική αξία.

Το GeoTriples είναι ένα ημι-αυτόματο εργαλείο που επιτρέπει τη δημοσίευση γεωχωρικών δεδομένων σε γράφους RDF χρησιμοποιώντας τα πλέον σύγχρονα λεξιλόγια όπως είναι τα GeoSPARQL και stSPARQL , αλλά ταυτόχρονα δεν είναι δεσμευμένο σε κάποιο συγκεκριμένο λεξιλόγιο.Σε αυτή την πτυχιακή εργασία παρουσιάζουμε μία παράλληλη υλοποίηση του εργαλείου GeoTriples αξιοποιώντας το Apache Spark.Χρησιμοποιώντας την τεχνική MapReduce η εφαρμογή μας προσφέρει σημαντική βελτίωση στο χρόνο εκτέλεσης όταν λαμβάνει σαν είσοδο μεγάλα δεδομένα.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Σημασιολογικό διαδίκτυο,Μετατροπή δεδομένων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: MapReduce,Διασυνδεδεμένα Δεδομένα,RDF,GeoTriples,Shapefile

ACKNOWLEDGMENTS

I would like to thank my supervisor Professor Manolis Koubarakis and my co-supervisor George Stamoulis for their guidance and support.Without their help the completion of this thesis wouldn’t have been possible.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1.INTRODUCTION

Geospatial data, also known as spatial data, is information about a physical object that can be represented by numerical values in a geographic coordinate system.The records in this type of information set have coordinates, an address, city, postal code or zip code, included with them. Geospatial data is all around us. Weather reports, suggested routes on Google Maps, geotagged tweets, store locations, and airline routes are an integral part of our daily lives and can be considered geospatial data. Geospatial data is also highly influential in today’s business market, and businesses that incorporate geospatial data into their analysis, reporting, and forecasting have the potential to outpace competitors through smarter use of their data [1].

“Linked Data” is the term for the collection of design principles and technologies centred around a paradigm to publish, retrieve, reuse, and integrate data on the Web (Kuhn et al. 2014). The adoption and application of Linked Data in the geospatial community have developed considerably in recent years [2] , as researchers and practitioners have started tapping the wealth of existing geospatial information and making it available on the Web.As a result, the linked open data (LOD) cloud has been rapidly populated with geospatial data. For example, Great Britain’s national mapping agency, Ordnance Survey, has been the first national mapping agency that has made various kinds of geospatial data from Great Britain available as linked open data.

In general, open EO (Earth Observation) data that are currently made available by space agencies such as ESA and NASA are not following the linked data paradigm. Therefore, EO data and other kinds of geospatial data that are necessary for a user to satisfy her information needs can only be found in different data silos, where each silo may contain only part of the needed data. Publishing the content of these silos as RDF graphs, enables the development of data analytics applications with great environmental and financial value. With the recent emphasis on open government data in many countries, the development of useful Web applications utilizing EO data and geospatial data in general is just a few SPARQL queries away[3].

The open source tool GeoTriples allows for the transformation of EO data and geospatial data in various formats, such as spatially-enabled relational databases (PostGIS and MonetDB), ESRI shapefiles, XML documents following a given schema (hence GML documents as well), KML documents, JSON and GeoJSON documents and CSV documents., into RDF graphs. GeoTriples goes beyond the state of the art by extending the R2RML mapping language to be able to deal with the specificities of geospatial data. It is a semi-automated tool that allows for the publication of geospatial information into an RDF graph using the state of the art vocabularies like GeoSPARQL but at the same time it is not tightly coupled to a specific vocabulary[4].

Our goal in this thesis was to create a parallel implementation of what GeoTriples achieves.We decided to achieve that by taking advantage of the utilities Apache Spark offers.

Spark is a cluster computing framework, which supports applications with working sets while providing similar scalability and fault tolerance properties to MapReduce.

The main abstraction in Spark is that of a resilient distributed dataset (RDD), which represents a read-only collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost. Users can explicitly cache an RDD in memory across machines and reuse it in multiple MapReduce-like parallel operations.

The parallel implementation we provide in this thesis allows the transformation of big datasets into RDF, utilizing MapReduce. The original dataset is divided into smaller pieces according to the number of mappers we are using. Then each mapper uses GeoTriples on the part of the dataset that is assigned to him and sends the partial RDF graph that is produced to the reducer. When all mappers have finished their task, we merge the RDF graphs into one output file.

2.BACKGROUND AND RELATED WORK

2.1 Apache Spark

The growth of data volumes in industry and research poses tremendous opportunities, as well as tremendous computational challenges. As data sizes have outpaced the capabilities of single machines, users have needed new systems to scale out computations to multiple nodes.From its humble beginnings in the AMPLab at U.C. Berkeley in 2009, Apache Spark has become one of the key big data distributed processing frameworks in the world. Spark can be deployed in a variety of ways, provides native bindings for the Java, Scala, Python, and R programming languages, and supports SQL, streaming data, machine learning, and graph processing. You'll find it used by banks, telecommunications companies, games companies, governments, and all of the major tech giants such as Apple, Facebook, IBM, and Microsoft[5].

2.1.1 Basic Components

Spark can be broken down to the following components[6]:

- Resilient Distributed Datasets and the Spark Core: The Spark Core is the foundation and provides basic I/O functionalities, task dispatching and scheduling. RDDs are basically a collection of partitioned data. These are generally created by referencing datasets in storages such as Cassandra, HBase et al., or by applying transformations such as map, reduce, filter etc. on existing RDDs.
- Spark SQL: Spark SQL, a component on the Core, introduces a new data abstraction called DataFrame, for providing support for structured data. It provides a language to manipulate Data Frames in Java, Python or Scala.
- Spark Streaming: Spark streaming rests on the Core as well, and levera on top of the Core which is proven to be ten times faster than Hadoop's disk-based Apache Mahout due to the distributed memory-based Spark architecture. It implements common algorithms to simplify large scale machine learning pipelines, like logistic or linear regression, decision trees or k-means clustering.
- MLlib Machine Learning Library: This is a machine learning framework on top of the Core which is proven to be ten times faster than Hadoop's disk-based Apache Mahout due to the distributed memory-based Spark architecture. It implements common algorithms to simplify large scale machine learning pipelines, like logistic or linear regression, decision trees or k-means clustering
- E.GraphX: It is a graph-processing framework on the Core, and provides an API for graph computation that can model the Pregel abstraction, providing an optimized runtime.

2.1.2 MapReduce and RDDs

A new model of cluster computing has become widely popular, in which data-parallel computations are executed on clusters of unreliable machines by systems that automatically provide locality-aware scheduling, fault tolerance, and load balancing. MapReduce pioneered this model, while systems like Dryad and Map-Reduce-Merge generalized the types of data flows supported. These systems achieve their scalability and fault tolerance by providing a programming model where the user creates acyclic data flow graphs to pass input data through a set of operators. This allows the underlying system to manage scheduling and to react to faults without user intervention. While this data flow programming model is useful for a large class of applications, there are applications that cannot be expressed efficiently as acyclic data flows. One class of these applications are the ones that reuse a working set of data across multiple parallel operations. This includes two use cases where Hadoop users were reporting that MapReduce is deficient:

- Iterative jobs: Many common machine learning algorithms apply a function repeatedly to the same dataset to optimize a parameter (e.g., through gradient descent). While each iteration can be expressed as a MapReduce/Dryad job, each job must reload the data from disk, incurring a significant performance penalty.
- Interactive analytics: Hadoop is often used to run ad-hoc exploratory queries on large datasets, through SQL interfaces such as Pig and Hive . Ideally, a user would be able to load a dataset of interest into memory across a number of machines and query it repeatedly. However, with Hadoop, each query incurs significant latency (tens of seconds) because it runs as a separate MapReduce job and reads data from disk.

Spark supports applications with working sets while providing similar scalability and fault tolerance properties to MapReduce. The main abstraction in Spark is that of a resilient distributed dataset (RDD), which represents a read-only collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost. Users can explicitly cache an RDD in memory across machines and reuse it in multiple MapReduce-like parallel operations. RDDs achieve fault tolerance through a notion of lineage: if a partition of an RDD is lost, the RDD has enough information about how it was derived from other RDDs to be able to rebuild just that partition. Although RDDs are not a general shared memory abstraction, they represent a sweet-spot between expressivity on the one hand and scalability and reliability on the other hand, and have been found well-suited for a variety of applications.

Spark provides a convenient language-integrated programming interface similar to DryadLINQ in the Scala programming language .Each RDD is represented by a Scala object. Spark lets programmers construct RDDs in four ways:

- From a file in a shared file system, such as the Hadoop Distributed File System (HDFS). • By "parallelizing" a Scala collection (e.g., an array) in the driver program, which means dividing it into a number of slices that will be sent to multiple nodes.
- By transforming an existing RDD. A dataset with elements of type A can be transformed into a dataset with elements of type B using an operation called flatMap, which passes each element through a user-provided function of type A ⇒ List[B]. 1 Other transformations can be expressed using flatMap, including map (pass elements through a function of type A ⇒ B) and filter (pick elements matching a predicate).
- By changing the persistence of an existing RDD. By default, RDDs are lazy and ephemeral. That is, partitions of a dataset are materialized on demand when they are used in a parallel operation (e.g., by passing a block of a file through a map function), and are discarded from memory after use. However, a user can alter the persistence of an RDD through two actions:

  1.The cache action leaves the dataset lazy, but hints that it should be kept in memory after the first time it is computed, because it will be reused.if there is not enough memory in the cluster to cache all partitions of a dataset, Spark will recompute them when they are used

2.The save action evaluates the dataset and writes it to a distributed file system such as HDFS.The saved version is used in future operations on it.

After the construction programmers can use these RDDs in actions, which are operations that return a value to the application or export data to a storage system. Examples of actions include:

- reduce: Combines dataset elements using an associative function to produce a result at the driver program.
- collect: Sends all elements of the dataset to the driver program. For example, an easy way to update an array in parallel is to parallelize, map and collect the array.
- foreach: Passes each element through a user provided function. This is only done for the side effects of the function (which might be to copy data to another system or to update a shared variable).
- count: Returns the number of elements in the dataset

Like DryadLINQ, Spark computes RDDs lazily the first time they are used in an action, so that it can pipeline transformations.[7]

Table 1: Comparison of RDDs with distributed shared memory

Aspect	RDDs	Distr. Shared Mem.
Reads	Coarse- or fine-grained	Fine-grained
Writes	Coarse-grained	Fine-grained
Consistency	Trivial (immutable)	Up to app / runtime
Fault recovery	Fine-grained and low-overhead using lineage	Requires checkpoints and program rollback
Straggler mitigation	Possible using backup tasks	Difficult
Work placement	Automatic based on data locality	Up to app (runtimes aim for transparency)
Behaviour if not enough RAM	Similar to existing data flow systems	Poor performance (swapping?)

To understand the benefits of RDDs as a distributed memory abstraction, we compare them against distributed shared memory (DSM) in Table 1. In DSM systems, applications read and write to arbitrary locations in a global address space. Note that under this definition, we include not only traditional shared memory systems , but also other systems where applications make fine-grained writes to shared state, including Piccolo, which provides a shared DHT, and distributed databases. DSM is a very general abstraction, but this generality makes it harder to implement in an efficient and fault-tolerant manner on commodity clusters.

The main difference between RDDs and DSM is that RDDs can only be created ("written") through coarse-grained transformations, while DSM allows reads and writes to each memory location. This restricts RDDs to applications that perform bulk writes, but allows for more efficient fault tolerance. In particular, RDDs do not need to incur the overhead of checkpointing, as they can be recovered using lineage.Furthermore, only the lost partitions of an RDD need to be recomputed upon failure, and they can be recomputed in parallel on different nodes, without having to roll back the whole program.

A second benefit of RDDs is that their immutable nature lets a system mitigate slow nodes (stragglers) by running backup copies of slow tasks as in MapReduce. Backup tasks would be hard to implement with DSM, as the two copies of a task would access the same memory locations and interfere with each other's updates.

Finally, RDDs provide two other benefits over DSM. First, in bulk operations on RDDs, a runtime can schedule tasks based on data locality to improve performance. Second, RDDs degrade gracefully when there is not enough memory to store them, as long as they are only being used in scan-based operations. Partitions that do not fit in RAM can be stored on disk and will provide similar performance to current data-parallel systems[8].

2.1.3 Spark vs Hadoop

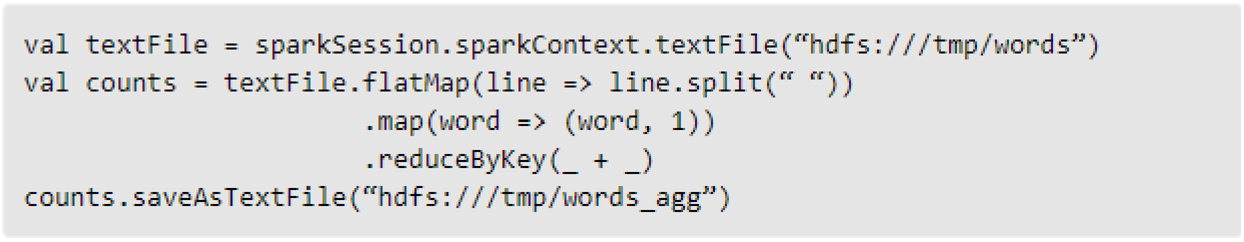
It's worth pointing out that Apache Spark vs. Apache Hadoop is a bit of a misnomer. You'll find Spark included in most Hadoop distributions these days. But due to two big advantages, Spark has become the framework of choice when processing big data, overtaking the old MapReduce paradigm that brought Hadoop to prominence.

The first advantage is speed. Spark's in-memory data engine means that it can perform tasks up to one hundred times faster than MapReduce in certain situations, particularly when compared with multi-stage jobs that require the writing of state back out to disk between stages. Even Apache Spark jobs where the data cannot be completely contained within memory tend to be around 10 times faster than their MapReduce counterpart.Table 2 demonstrates how Spark outperformed Hadoop in a sorting data experiment and set a new world record[6].

Table 2: Comparison between Hadoop MapReduce and Apache Spark

	Hadoop MR Record	**Spark Record**	**Spark 1 PB**
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
Cluster disk throughput	3150 GB/s	618 GB/s	570 GB/s
Sort Rate	1.42TB/min	4.27TB/min	4.27TB/min

The second advantage is the developer-friendly Spark API. As important as Spark's speed-up is, one could argue that the friendliness of the Spark API is even more important.In comparison to MapReduce and other Apache Hadoop components, the Apache Spark API is very friendly to developers, hiding much of the complexity of a distributed processing engine behind simple method calls. The canonical example of this is how almost 50 lines of MapReduce code to count words in a document can be reduced to just a few lines of Apache Spark (here shown in Scala):



```
val textFile = sparkSession.sparkContext.textFile("hdfs:///tmp/words")
val counts = textFile.flatMap(line => line.split(" "))
                     .map(word => (word, 1))
                     .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs:///tmp/words_agg")
```

Figure 1: Counting words in a document in Scala

By providing bindings to popular languages for data analysis like Python and R, as well as the more enterprise-friendly Java and Scala, Apache Spark allows everybody from application developers to data scientists to harness its scalability and speed in an accessible manner [5].

Table 3 demonstrates the main differences between Spark and Hadoop [9].

Table 3: Spark/Hadoop MapReduce differences

	Hadoop MapReduce	**Spark**
Storage	Disk only	In Memory or on disk
Operations	Map and Reduce	Map,Reduce,Join,Sample, etc...
Execution model	Batch	Batch, interactive, streaming
Programming environments	Java	Scala, Java, R, and Python

2.2 Linked Data

We are surrounded by data – data about the performance of our locals schools, the fuel efficiency of our cars, a multitude of products from different vendors, or the way our taxes are spent. By helping us make better decisions, this data is playing an increasingly central role in our lives and driving the emergence of a data economy. Increasing numbers of individuals and organizations are contributing to this deluge by choosing to share their data with others, including *Web-native* companies such as *Amazon* and *Yahoo!*, newspapers such as *The Guardian* and *The New York Times*, public bodies such as the UK and US governments, and research initiatives within various scientific disciplines.Third parties, in turn, are consuming this data to build new businesses, streamline online commerce, accelerate scientific progress, and enhance the democratic process [10].

The World Wide Web has radically altered the way we share knowledge by lowering the barrier to publishing and accessing documents as part of a global information space. Hypertext links allow users to traverse this information space using Web browsers, while search engines index the documents and analyse the structure of links between them to infer potential relevance to users' search queries (Brin & Page, 1998). This functionality has been enabled by the generic, open and extensible nature of the Web (Jacobs & Walsh, 2004), which is also seen as a key feature in the Web's unconstrained growth. Despite the inarguable benefits the Web provides, until recently the same principles that enabled the Web of documents to flourish have not been applied to data. Traditionally, data published on the Web has been made available as raw dumps in formats such as

CSV or XML, or marked up as HTML tables, sacrificing much of its structure and semantics. In the conventional hypertext Web, the nature of the relationship between two linked documents is implicit, as the data format, i.e. HTML, is not sufficiently expressive to enable individual entities described in a particular document to be connected by typed links to related entities. However, in recent years the Web has evolved from a global information space of linked documents to one where both documents and data are linked. Underpinning this evolution is a set of best practices for publishing and connecting structured data on the Web known as Linked Data[11].

2.2.1 Linked Data Definition

Linked Data is about using the Web to connect related data that were not previously linked, or using the Web to lower the barriers to linking data currently linked using other methods. More specifically, Wikipedia defines Linked Data as “a term used to describe a recommended best practice for exposing, sharing, and connecting pieces of data, information, and knowledge on the Semantic Web using URIs and RDF.” Linked data uses structured ontologies, or vocabularies, which are the explicit formal specifications expressing relevant terms in a domain, relationships among them, and their behavior. Software tools that discover and manipulate datasets or other information must understand the associated vocabulary in order to make full use of them.

The result, which we will refer to as the Web of Data, may more accurately be described as a web of things in the world, described by data on the Web.

Berners-Lee (2006) outlined a set of 'rules' for publishing data on the Web in a way that all published data becomes part of a single global data space:

- Use URIs as names for things
- Use HTTP URIs so that people can look up those names
- When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL)
- Include links to other URIs, so that they can discover more things

These have become known as the 'Linked Data principles', and provide a basic recipe for publishing and connecting data using the infrastructure of the Web while adhering to its architecture and standards [11].

2.2.2 The Linked Data Technology Stack

Linked Data relies on two technologies that are fundamental to the Web: Uniform Resource Identifiers (URIs) (Berners-Lee et al., 2005) and the HyperText Transfer Protocol (HTTP) (Fielding et al., 1999). While Uniform Resource Locators (URLs) have become familiar as addresses for documents and other entities that can be located on the Web, Uniform Resource Identifiers provide a more generic means to identify any entity that exists in the world. Where entities are identified by URIs that use the http:// scheme, these entities can be looked up simply by dereferencing the URI over the HTTP protocol. In this way, the HTTP protocol provides a simple yet universal mechanism for retrieving resources that can be serialised as a stream of bytes (such as a photograph of a dog), or retrieving descriptions of entities that cannot themselves be sent across the network in this way (such as the dog itself). URIs and HTTP are supplemented by a technology that is critical to the Web of Data – RDF, introduced above. Whilst HTML provides a means to structure and link documents on the Web, RDF provides a generic, graph-based data model with which to structure and link data that describes things in the world [11].

More about the RDF Data Model in the next section(2.3).

2.2.3 Forming the Web of Data

The RDF Vocabulary Definition Language (RDFS) (Brickley & Guha, 2004) and the Web Ontology Language (OWL) (McGuinness & van Harmelen, 2004) provide a basis for creating vocabularies that can be used to describe entities in the world and how they are related. Vocabularies are collections of classes and properties. Vocabularies are themselves expressed in RDF, using terms from RDFS and OWL, which provide varying degrees of expressivity in modelling domains of interest. Anyone is free to publish vocabularies to the Web of Data (Berrueta & Phipps, 2008), which in turn can be connected by RDF triples that link classes and properties in one vocabulary to those in another, thereby defining mappings between related vocabularies.

By employing HTTP URIs to identify resources, the HTTP protocol as retrieval mechanism, and the RDF data model to represent resource descriptions, Linked Data directly builds on the general architecture of the Web (Jacobs & Walsh, 2004). The Web of Data can therefore be seen as an additional layer that is tightly interwoven with the classic document Web and has many of the same properties:

- The Web of Data is generic and can contain any type of data.
- Anyone can publish data to the Web of Data.
- Data publishers are not constrained in choice of vocabularies with which to represent data.
- Entities are connected by RDF links, creating a global data graph that spans data sources and enables the discovery of new data sources.

From an application development perspective the Web of Data has the following characteristics:

Data is strictly separated from formatting and presentational aspects [11].

- Data is self-describing. If an application consuming Linked Data encounters data described with an unfamiliar vocabulary, the application can dereference the URIs that identify vocabulary terms in order to find their definition.
- The use of HTTP as a standardized data access mechanism and RDF as a standardized data model simplifies data access compared to Web APIs, which rely on heterogeneous data models and access interfaces.
- The Web of Data is open, meaning that applications do not have to be implemented against a fixed set of data sources, but can discover new data sources at run-time by following RDF links.

2.2.4 The vision of the Semantic Web

It is important to understand the benefits of Linked Data in the Semantic Web. When we think about the Web, the first issue is how to find what we are looking for. This is especially true for data, even with an increasing number of sites acting as portals for data. The discovery of data is a major challenge. And, once discovered, it is often difficult to know what the data are and how to use them – how are they organized and structured? How were they collected? What format are they in and how can they be processed? Further, for data to be really useful in research, one may want to compare them with data from other sources, so we need to understand how comparable different data from various sources are. An ideal vision of using Linked Data to produce a rich browsing environment for social science data might look something like this:

“A researcher discovers in an online journal an article about the relationship between obesity and income in a specific geographic area. She reads the article and links to a table with references to specific income and obesity variables from the underlying dataset, with links to full descriptions of each variable (name and label, associated question, responses, and frequencies) in context and links to download the data and browse additional documentation. This dataset is part of a multi-wave longitudinal study, and the user can view these variables in other years to compare them. She is able to recreate the table using data from other years. The researcher is also able to browse other variables in the dataset and to explore similar variables in other datasets ("More Like This") that might differ slightly in terms of how the questions were posed to respondents. Let's say the researcher wants to know about the relationships among income, obesity, and the proximity of fast food restaurants in the region of interest. A quick search reveals data on fast food, and the researcher is able to understand quickly whether the analysis she wants to perform is scientifically sound given the nature of the disparate datasets because the data include "smart" variables that know what they can combine with. She successfully merges the data and is presented with a visualization of the results in the form of a complex table and a map. This happens in a secure environment to minimize disclosure risk. The researcher can also follow links to other articles that used these data and specifically the variables of interest. She can link to a profile of the researcher and discover additional publications on the topic of interest from the researcher's CV. She might even initiate a collaboration with that researcher, producing new knowledge linked to existing information on the Web and adding to the rich Web of linkages in her domain."

In short, through Linked Data for the social sciences, users should be able to easily discover the existence of data, and to determine what the data contain, how they are structured, and how they can be used. The data should be well-documented. They should be of known quality and provenance, and should exist in relationship to other versions of the same dataset, so that corrections and updates can be known. If comparable/ scientifically compatible with other datasets, then that should be something that is easily determined, and it should be possible to easily merge the data[12].

2.3 RDF (Resource Description Framework)

The World Wide Web affords unprecedented access to globally distributed information. Metadata, or structured data about data, improves discovery of and access to such information. The effective use of metadata among applications, however, requires common conventions about semantics, syntax, and structure. Individual resource description communities define the semantics, or meaning, of metadata that address their particular needs. Syntax, the systematic arrangement of data elements for machine-processing, facilitates the exchange and use of metadata among multiple applications. Structure can be thought of as a formal constraint on the syntax for the consistent representation of semantics.

The Resource Description Framework (RDF), developed under the auspices of the World Wide Web Consortium (W3C) , is an infrastructure that enables the encoding, exchange, and reuse of structured metadata. This infrastructure enables metadata interoperability through the design of mechanisms that support common conventions of semantics, syntax, and structure. RDF does not stipulate semantics for each resource description community, but rather provides the ability for these communities to define metadata elements as needed. RDF uses XML (eXtensible Markup Language) as a common syntax for the exchange and processing of metadata. The XML syntax is a subset of the international text processing standard SGML (Standard Generalized Markup Language) specifically intended for use on the Web. The XML syntax provides vendor independence, user extensibility, validation, human readability, and the ability to represent complex structures. By exploiting the features of XML, RDF imposes structure that provides for the unambiguous expression of semantics and, as such, enables consistent encoding, exchange, and machine-processing of standardized metadata.

RDF supports the use of conventions that will facilitate modular interoperability among separate metadata element sets. These conventions include standard mechanisms for representing semantics that are grounded in a simple, yet powerful, data model discussed below. RDF additionally provides a means for publishing both human-readable and machine-processable vocabularies. Vocabularies are the set of properties, or metadata elements, defined by resource description communities. The ability to standardize the declaration of vocabularies is anticipated to encourage the reuse and extension of semantics among disparate information communities.RDF is designed to support semantic modularity by creating an infrastructure that supports the combination of distributed attribute registries. Thus, a central registry is not required. This permits communities to declare vocabularies which may be reused, extended and/or refined to address application or domain specific descriptive requirements [13].

2.3.1 Structure

The Resource Description Framework (RDF) is a framework for representing information in the Web.

All RDF-based languages and specifications are linked by an abstract syntax (a data model).The core structure of the abstract syntax is a set of triples, each consisting of a subject, a predicate and an object.So basically a description of a resource is represented as a number of triples. A set of such triples is called an RDF graph. An RDF graph can be visualized as a node and directed-arc diagram, in which each triple is represented as a node-arc-node link [14].

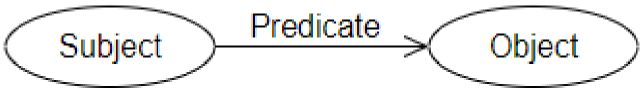


Figure 2: An RDF graph with two nodes (Subject and Object) and a triple connecting them (Predicate)

There can be three kinds of nodes in an RDF graph: IRIs, literals, and blank nodes.To be more specific :

- The subject is an IRI or a blank node
- The predicate is an IRI
- the object is an IRI, a literal or a blank node

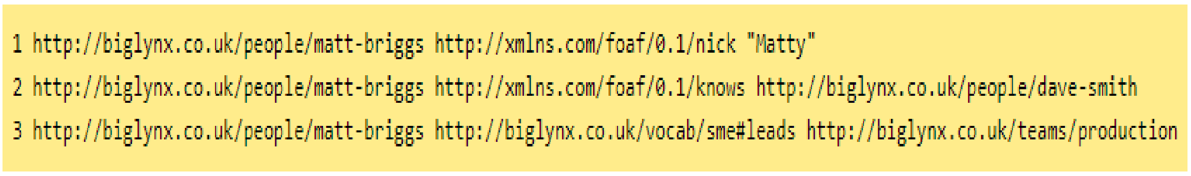
Any IRI or literal denotes something in the world (the "universe of discourse"). These things are called resources. Anything can be a resource, including physical things, documents, abstract concepts, numbers and strings; the term is synonymous with "entity" as it is used in the RDF Semantics specification. The resource denoted by an IRI is called its referent, and the resource denoted by a literal is called its literal value.

Two principal types of RDF triples can be distinguished, *Literal Triples* and *RDF Links*:

1. Literal Triples have an RDF literal such as a string, number, or date as the object. Literal triples are used to describe the properties of resources. For instance, literal triples are used to describe the name or date of birth of a person. Literals may be plain or typed: A plain literal is a string combined with an optional language tag. The language tag identifies a natural language, such as English or German. A typed literal is a string combined with a datatype URI. The datatype URI identifies the datatype of the literal. Datatype URIs for common datatypes such as integers, floating point numbers and dates are defined by the XML Schema datatypes

specification. The first triple in the code example below is a literal triple, stating that *Big Lynx* Lead Cameraman Matt Briggs has the nickname *Matty*.

2. RDF Links describe the relationship between two resources. RDF links consist of three URI references. The URIs in the subject and the object position of the link identify the related resources. The URI in the predicate position defines the type of relationship between the resources. For instance, the second triple in Figure 2 below states that Matt Briggs *knows* Dave Smith. The third triple states that he leads something identified by the URI http://biglynx.co.uk/teams/production (in this case the *Big Lynx* Production Team). A useful distinction can be made between *internal* and *external* RDF links. *Internal RDF links* connect resources within a single Linked Data source. Thus, the subject and object URIs are in the same namespace. *External RDF links* connect resources that are served by different Linked Data sources. The subject and object URIs of external RDF links are in different namespaces. External RDF links are crucial for the Web of Data as they are the glue that connects data islands into a global, interconnected data space [10].



```
http://biglynx.co.uk/people/matt-briggs http://xmlns.com/foaf/0.1/nick "Matty"
http://biglynx.co.uk/people/matt-briggs http://xmlns.com/foaf/0.1/knows http://biglynx.co.uk/people/dave-smith
http://biglynx.co.uk/people/matt-briggs http://biglynx.co.uk/vocab/sme#leads http://biglynx.co.uk/teams/production
```

Figure 3: Example of RDD triples with Links

2.3.2 RDF and change over time

The RDF data model is atemporal: RDF graphs are static snapshots of information.However, RDF graphs can express information about events and about temporal aspects of other entities, given appropriate vocabulary terms.Since RDF graphs are defined as mathematical sets, adding or removing triples from an RDF graph yields a different RDF graph.

We informally use the term RDF source to refer to a persistent yet mutable source or container of RDF graphs. An RDF source is a resource that may be said to have a state that can change over time. A snapshot of the state can be expressed as an RDF graph. For example, any web document that has an RDF-bearing representation may be considered an RDF source. Like all resources, RDF sources may be named with IRIs and therefore described in other RDF graphs [14] .

Intuitively speaking, changes in the universe of discourse can be reflected in the following ways:

- An IRI, once minted, should never change its intended referent. (See URI persistence [WEBARCH].)
- Literals, by design, are constants and never change their value.
- A relationship that holds between two resources at one time may not hold at another time.
- RDF sources may change their state over time. That is, they may provide different RDF graphs at different times.
- Some RDF sources may, however, be immutable snapshots of another RDF source, archiving its state at some point in time.

2.3.3 RDF Data Model and Linked Data

The main benefits of using the RDF data model in a Linked Data context are that:

- By using HTTP URIs as globally unique identifiers for data items as well as for vocabulary terms, the RDF data model is inherently designed for being used at global scale and enables anybody to refer to anything.
- Clients can look up any URI in an RDF graph over the Web to retrieve additional information. Thus each RDF triple is part of the global Web of Data and each RDF triple can be used as a starting point to explore this data space.
- The data model enables you to set RDF links between data from different sources.
- Information from different sources can easily be combined by merging the two sets of triples into a single graph.
- RDF allows you to represent information that is expressed using different schemata in a single graph, meaning that you can mix terms for different vocabularies to represent data.
- Combined with schema languages such as RDF-Schema and OWL , the data model allows the use of as much or as little structure as desired, meaning that tightly structured data as well as semi-structured data can be represented.

Besides the features mentioned above, the RDF Recommendation also specifies a range of other features which have not achieved widespread adoption in the Linked Data community. In order to make it easier for clients to consume data, it is recommended to use only the subset of the RDF data model described above. In particular, the following features should be avoided in a Linked Data context [10].

- RDF reification should be avoided, as reified statements are rather cumbersome to query with the SPARQL query language . Instead of using reification to publish metadata about individual RDF statements, meta-information should instead be attached to the Web document containing the relevant triples.
- RDF collections and RDF containers are also problematic if the data needs to be queried with SPARQL. Therefore, in cases where the relative ordering of items in a set is not significant, the use of multiple triples with the same predicate is recommended.
- The scope of blank nodes is limited to the document in which they appear, meaning it is not possible to create RDF links to them from external documents, reducing the potential for interlinking between different Linked Data sources. In addition, it becomes much more difficult to merge data from different sources when blank nodes are used, as there is no URI to serve as a common key. Therefore, all resources in a data set should be named using URI references.

2.4 Geotriples

In the last few years there has been significant effort on publishing EO(Earth Observation) and geospatial data sources as linked open data. However, the problem of publishing geospatial data sources into RDF graphs using a generic and extensible framework has received little attention as it has only recently emerged.

Geospatial data in general and EO data in particular, can come in vector or raster form and are usually accompanied by metadata. Vector data, available in formats such as ESRI shapefiles, KML, and GeoJSON documents, can be accessed either directly or via Web Services such as the OGC Web Feature Service or the query language of a geospatial DBMS. Raster data, available in formats such as GeoTIFF, Network Common Data Form (netCDF), Hierarchical Data Format (HDF), can be accessed either directly or via Web Services such as the OGC Web Coverage Processing Service (WCS) or the query language of an array DBMS, e.g., the array-query language SciQL4 . Metadata about EO data are encoded in various formats ranging from custom XML schemas to domain specific standards like the OGC GML Application schema for EO products and the OGC Metadata Profile of Observations and Measurements.

Automating the process of publishing linked geospatial data has not been addressed yet. For example, in the wildfire monitoring and management application that was developed in TELEIOS, custom Python scripts were used for publishing all necessary data as linked data. For this reason, the tool GeoTriples was designed and implemented in the context of the EU FP7 project LEO5 . GeoTriples allows the transformation of geospatial data stored in spatially-enabled relational databases and raw files. It is implemented as an extension to the D2RQ platform and goes beyond the state of the art by extending the R2RML mapping language to deal with the specificities of geospatial data. GeoTriples uses GeoSPARQL as the target vocabulary but the user is free to use any vocabulary he finds appropriate [3].

2.4.1 System Architecture

The system architecture of GeoTriples is depicted in Figure. 1.The input data for GeoTriples can be geospatial data and metadata stored in ESRI Shapefiles, XML, GML, KML, JSON, GeoJSON and CSV documents or spatially-enabled relational databases (e.g., PostGIS and MonetDB). GeoTriples has a connector that is responsible for providing an abstraction layer that allows the rest of the components to transparently access the input data. GeoTriples comprises three main components: the mapping generator, the mapping processor and the stSPARQL/GeoSPARQL evaluator.

The mapping generator is given as input a data source and creates automatically an R2RML/RML mapping document, depending on the type of the input. The generated mapping is enriched with subject and predicate–object maps, taking into account all transformations that are needed to produce an RDF graph that is compliant with the GeoSPARQL vocabulary. Afterwards, the user may edit the generated mapping document to make it comply with his/her requirements (e.g., use a different vocabulary). We point out that the ability of GeoTriples to use different vocabularies is a useful feature since even standardized vocabularies such as the one of GeoSPARQL can be dropped, modified or extended in the future.

The mapping processor may use either the generated mapping document or one created by the user from scratch. Based on the triples map definitions in the mapping file, the component generates the final RDF graph which can be manifested in any of the popular RDF syntaxes such as Turtle, RDF/XML, Notation3 or N-Triples. The mapping processor has been implemented in two ways. The first implementation runs on a single processor, while the second runs in a distributed manner using the Apache Hadoop framework.

The stSPARQL/GeoSPARQL evaluator is a component that evaluates an stSPARQL/GeoSPARQL query over a relational database given an R2RML mapping. The evaluator is a thin layer that integrates GeoTriples with the OBDA engine Ontop-spatial . It supports the evaluation of stSPARQL/GeoSPARQL queries over virtual RDF graphs defined through R2RML mappings to a geospatial relational database [15] .

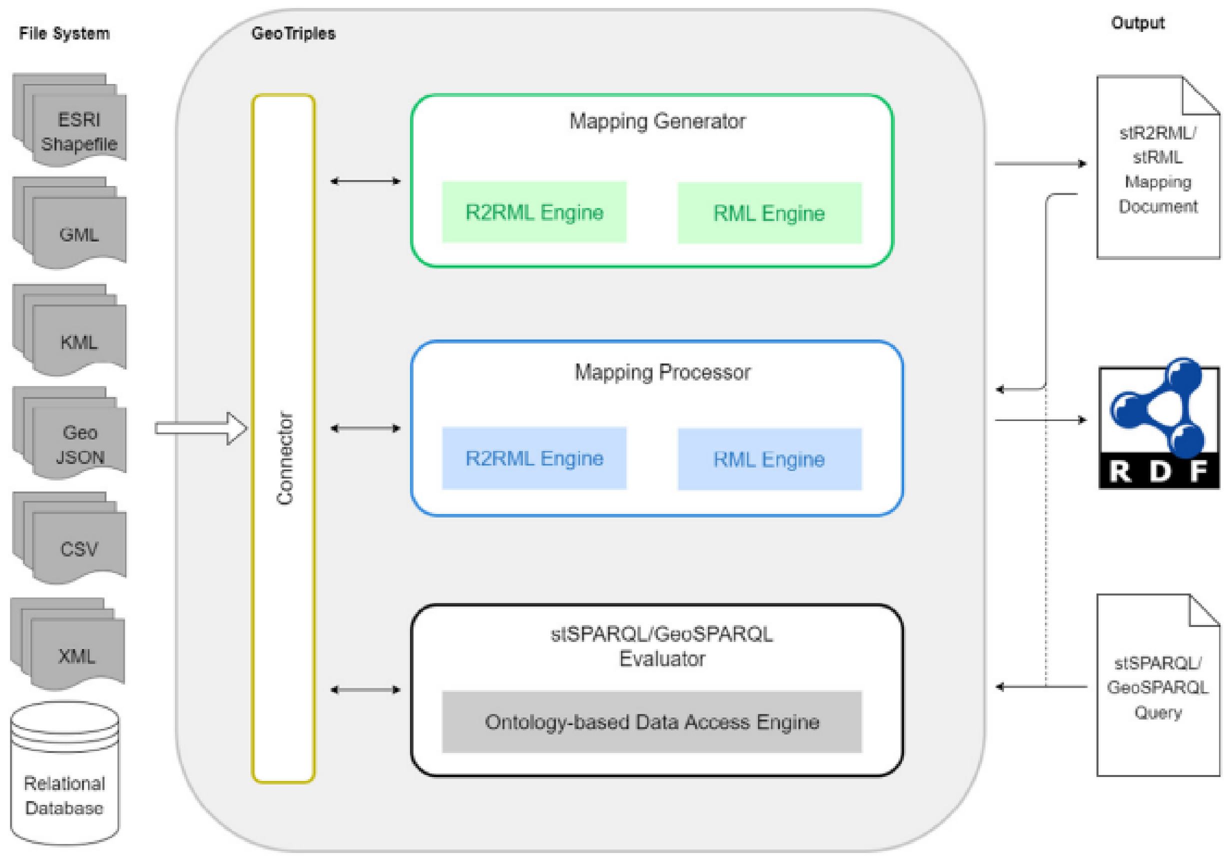


Figure 4: The system architecture of GeoTriples.

2.4.2 Transforming Geospatial Data into RDF graphs using GeoTriples

In this section the main functionality of GeoTriples is presented.It can be broken down in two main parts. First part is the automatic production of extended R2RML mappings that take into account the spatial dimension of the input data.Following in the second part is demonstrated how these mappings are processed subsequently by GeoTriples in order to generate an RDF graph.

2.4.2.1 Automatic Generation of R2RML Mappings

Much work has been done recently on extending RDF to represent and query geospatial information. The most mature results of this work are the data model stRDF and the query language stSPARQL and the OGC standard GeoSPARQL. GeoSPARQL is an OGC standard for the representation and querying of geospatial linked data. GeoSPARQL defines much of what is required for such a query language by providing vocabulary (classes, properties, and functions) that can be used in RDF graphs and SPARQL queries to represent and query geospatial data. The top level classes defined in GeoSPARQL are geo:SpatialObject that has as instances everything that can have a spatial representation and geo:Feature that represents all features and is the superclass of all classes of features that the users might want to define. To represent geometric objects, the class geo:Geometry is introduced by GeoSPARQL. Additional vocabulary is also defined by GeoSPARQL for asserting and querying information about geometries.

Given a spatially-enabled database or a raw file that contains geometric information, GeoTriples generates an R2RML mapping document. Let us take for example the Natura

2000 dataset of Germany that contains information about protected areas in Germany and is distributed in the form of an ESRI shapefile. More details on this dataset can be found in Section 4. Conceptually, each geometric object stored in the ESRI shapefile should be an instance of the class geo:Geometry, and all non-geometric attributes that characterize each geometry are thematic attributes of the corresponding feature. Following this modeling approach, we generate an instance of the class geo:Feature and an instance of the class geo:Geometry for each geometric object stored in the ESRI shapefile. Geometric and non-geometric attributes that appear at the ESRI shapefile are assigned accordingly to the appropriate instances. Part of the R2RML mapping that is generated automatically by GeoTriples to represent the features stored in the Natura 2000 ESRI shapefile is the following:

```
_:natura
    rr:logicalTable [ rr:tableName "‘natura‘"; ];
    rr:subjectMap [
            rr:class geo:Feature;
            rr:template "http://data.example.com/natura/Feature/id/{‘gid‘}"; ];

    rr:predicateObjectMap [
            rr:predicate nato:has_SITECODE;
            rr:objectMap [ rr:datatype xsd:string;
                       rr:column "‘SITECODE‘"; ];
    ];
    rr:predicateObjectMap [
            rr:predicate geo:hasGeometry ;
            rr:objectMap [
                rr:parentTriplesMap _:natura_geometry;
                rr:joinCondition [
                        rr:child  "gid";
                        rr:parent "gid"; ]; ]; ].
```

Figure 5: Part of the R2RML mapping by GeoTriples for the geo:Feature class

This mapping document contains a triples map that describes how instances of the class geo:Feature are constructed. All thematic information that is stored in the ESRI shapefile is assigned to the generated feature and a link between the feature and its geometry is also generated. However, the notion of a primary key is not defined for ESRI shapefiles. Each ESRI shapefile though is accompanied by a relational table in dBASE format that stores information about the geometries, so we define as a unique identifier for each geometric object the respective row identifier in the dBASE table. In the example above we represent this information as an extra attribute with name ’gid’.

Part of the R2RML mapping that is generated automatically by GeoTriples to represent the geometries stored in the Natura 2000 ESRI shapefile is the following:

```
_:naturaGeometry
    rr:logicalTable [ rr:tableName "‘natura‘"; ];
    rr:subjectMap [
        rr:class geo:Geometry;
        rr:template "http://data.example.com/natura/Geometry/id/{‘gid‘}"; ];

    rr:predicateObjectMap [
        rr:predicate geo:dimension;
        rr:objectMap [


          rrx:transformation [
              rrx:function geof:dimension;
               rrx:argumentMap (
                  [rr:column "‘Geom‘"] ); ]  ]; ].
```

Figure 6: Part of the R2RML mapping by GeoTriples for the geo:Geometry class

This mapping document contains a triples map that describes how instances of the class geo:Geometry are constructed. All geometric information that is stored in the ESRI shapefile is assigned to the generated geometry. In addition, object maps define that geospatial functions like geof:dimension are applied to the serialization of each geometric object in order to produce the values that will appear at the object part of the corresponding triple.

Notice that in the above example we extended the definition of an object map by allowing it to be the RDF term obtained by applying a transformation on the source data. Each transformation defines the SPARQL built-in function or the SPARQL extension function to be invoked, using as an argument the sequence of RDF terms that are produced by the respective term maps [3].

2.4.2.2 Processing of R2RML mappings for producing RDF graphs

R2RML mappings usually consist of two triples maps; one for handling thematic information and one for geospatial information. The triples map that handles thematic information defines a logical table that contains the thematic attributes and a unique identifier for the generated instances. The latter could be either the primary key of the table in case the input data is a relational database or a row number in the dBASE table of an ESRI shapefile. Combined with a URI template, the unique identifier is used to produce the URI that serve as subjects of the produced triples. The predicate object maps are also processed accordingly in order to define RDF properties the value of which originate from the value of the column of the thematic logical table.

The triples map that handles the geospatial information of the input data source, defines a logical table with unique identifier similar to the thematic one. However, according to the type of the data source, the definition of this logical table may vary. For instance, in the case of a relational database, it is defined by providing an appropriate SQL query that uses spatial functions provided by spatially enabled relational backend (e.g. utilize the function ST_Dimension). If the input source is an ESRI shapefile, then GeoTriples will perform such transformations on the fly by evaluating the SPARQL extension function using the JTS Topology Suite [3].

3.GEOTRIPLES IMPLEMENTATION WITH SPARK

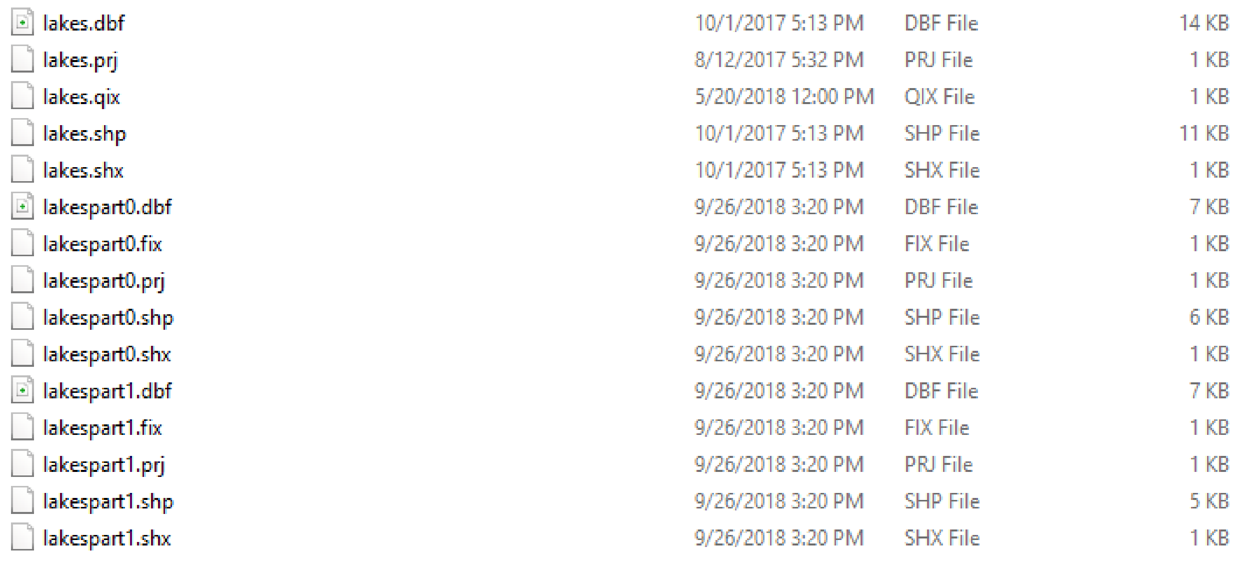
In this chapter we're going to explain each step of our application.It's main point can be boiled down to the following: Instead of using GeoTriples to transform a shapefile to RDF we divide the file into smaller parts.Each part is then used as input to GeoTriples so that the transformation process is done in parallel, utilizing Apache Spark's Map function.Finally a reduction is applied so that the smaller RDF files which were produced by GeoTriples merge into the final RDF representation of our initial shapefile.

This parallel implementation of the GeoTriples tool, shows promising results in terms of performance.Chapter 4 goes into more detail about the pros of our implementation and the edge it offers when it comes to time complexity.

The following is a description of our approach in each step of the application.

3.1 Input Data Manipulation

At first the user is prompted to choose the number of pieces in which his file will be broken into.If the user doesn't have a preference, a default value is instead used.Then the user has to pick the file he wants to use as input.Currently our application only supports shapefiles.Using the name of the original shapefile as basis and adding a suffix ("partx" where x is an Integer) to ensure uniqueness , a number of new files (pieces) are created.This number is equal to the number of pieces in which the shapefile will be broken into and each new file/piece will be a part of the original.E.g. if the user chooses as input the file "Lakes.shp" and decides to break it into 3 pieces ,3 new files will be created named "Lakespart0.shp","Lakespart1.shp" and "Lakespart2.shp" respectively.Likewise for the rest of the shapefile the .dbf .fix .prj .shx files will be created with the appropriate suffix.The new files are in the same directory as the original input shapefile.Figure 1. illustrates the result of our example.



lakes.dbf	10/1/2017 5:13 PM	DBF File	14 KB
lakes.prj	8/12/2017 5:32 PM	PRJ File	1 KB
lakes.qix	5/20/2018 12:00 PM	QIX File	1 KB
lakes.shp	10/1/2017 5:13 PM	SHP File	11 KB
lakes.shx	10/1/2017 5:13 PM	SHX File	1 KB
lakespart0.dbf	9/26/2018 3:20 PM	DBF File	7 KB
lakespart0.fix	9/26/2018 3:20 PM	FIX File	1 KB
lakespart0.prj	9/26/2018 3:20 PM	PRJ File	1 KB
lakespart0.shp	9/26/2018 3:20 PM	SHP File	6 KB
lakespart0.shx	9/26/2018 3:20 PM	SHX File	1 KB
lakespart1.dbf	9/26/2018 3:20 PM	DBF File	7 KB
lakespart1.fix	9/26/2018 3:20 PM	FIX File	1 KB
lakespart1.prj	9/26/2018 3:20 PM	PRJ File	1 KB
lakespart1.shp	9/26/2018 3:20 PM	SHP File	5 KB
lakespart1.shx	9/26/2018 3:20 PM	SHX File	1 KB

Figure 7: The result of the division of the lakes shapefile

Utilizing the geotools library we then acquire the features of the input shapefile and split them amongst the pieces.So if the shapefile has 300 features and the user decides to split it to 5 pieces , each piece will end up containing 60 features.The first piece will include the first 60 features(1-60),the second the next 60 (61-120) and so on.Finally we create the "paths" array that includes the path of each piece.

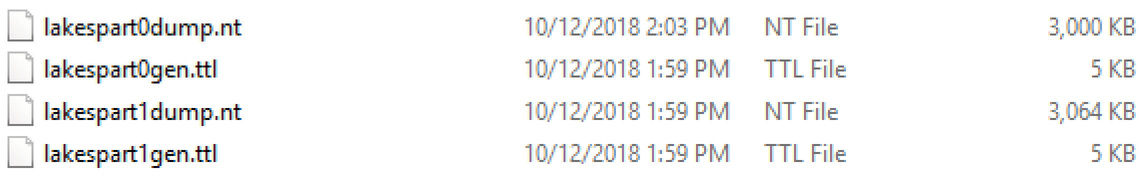
This concludes the division step of our application.We have successfully split the initial shapefile into smaller pieces which will be crucial for the steps to follow.

3.2 Mapping

In this step we take advantage of the utilities Apache Spark offers.We use Spark's parallelize function passing the "paths" array as an argument, creating an RDD.On this RDD we perform the Spark map operation.Doing so we are able to manipulate each piece in parallel , through a function of our own making.

The input for this function is a path from the "paths" array.In the function we use the GeoTriples tool on the piece(shapefile) that is located in the given path.To be more specific , at first we invoke the GeoTriples "generate_mapping" function passing the shapefile path.This function creates a .ttl file which we then pass to the GeoTriples "dump_rdf" function.This function performs the necessary transformations and returns the result in the form of a .nt file.This .nt file is the RDF representation of the given shapefile/piece.Since the function is performed on each piece in parallel we end up with a lot of .nt files which we need to merge to get the final result.This is achieved through the Spark reduce function and is described in the next section.

Figure 1. illustrates the result of the mapping stage in the Lakes example we used beforehand.The .ttl files won't be used any further but they were vital to the dump_rdf GeoTriples function.



```
lakespart0dump.nt    10/12/2018 2:03 PM   NT File    3,000 KB
lakespart0gen.ttl    10/12/2018 1:59 PM   TTL File       5 KB
lakespart1dump.nt    10/12/2018 1:59 PM   NT File    3,064 KB
lakespart1gen.ttl    10/12/2018 1:59 PM   TTL File       5 KB
```

Figure 8: Result of mapping stage

3.3 Reduction and final product

The last step of our application involves the forming of the final .nt file that is the RDF transformation of the original input shapefile.

In the previous step we mapped each part and ended up with many smaller .nt files.Since order does not matter in RDF files we can utilize the Spark Reduce function.This function receives as input pairs of .nt files originating from the array that was the Map stage output.We then append the second file of the pair to the first.That first file is returned and used as the first file of the next pair as well and so on and so forth.

When the reduction concludes we have successfully merged all the .nt files to a single one.We then proceed to copy that file to the final one whose name is the initial shapefile name with the suffix "RDF".E.g. in our previous example where we used the file "Lakes.shp" as input the final output would be stored in "LakesRDF.nt".

4.EXPERIMENT EVALUATION

4.1 System specifications

In order to compare our implementation to the original GeoTriples application we ran an extended series of experiments , using a variety of datasets as input.The specifications of the system that our program was ran on are the following:

Table 4: Environment specifications

CPU	Intel(R) Core(™) i7-6700K CPU @4.00GHz 4.00 GHz
GPU	NVIDIA GeForce GTX 1060 6GB
RAM	16.0 GB
O.S.	Windows 8.1
IDE	Eclipse

4.2 Datasets

As far as the datasets are concerned , we utilized a plethora of inputs with variations on size and origination.They are listed below.

- One shapefile with administrative divisions from all over the world.The most generic dataset that was used with a size of 1.6 Gigabytes.
- Four different shapefiles with administrative divisions from Great Britain.Each one has an approximate size of 7 Megabytes.
- Four different shapefiles with administrative divisions from Greece.Each one has an average size of 8 Megabytes.
- Finally 1 shapefile with buildings from Netherlands.The biggest dataset that we used with a size of 1.8 Gigabytes.

4.3 Results

Utilizing the GeoTriples tool we measured the time it took to transform each shapefile to RDF.We then repeated the process for our implementation experimenting with the amount of mappers used.To be more specific each shapefile was tested 3 times for 2,4 and 8 mappers respectively.Extra mappers mean the initial shapefile will be separated into extra parts before the actual transformation.The tables below illustrate the results.

4.3.1 Administrative divisions World(1 shapefile)

Table 5: Geotriples - administrative divisions World(1.6 GB)

Time (in seconds)	548.415

Table 6: Our implementation - administrative divisions World(1.6 GB)

Mappers	2	4	8
Time (in seconds)	407.025	385.554	417.145

4.3.2 Administrative divisions Great Britain(4 shapefiles)

Table 7: Geotriples - administrative divisions Great Britain(7 MB)

Shapefile(#)	0	1	2	3
Time(in seconds)	2.282	2.148	2.152	2.33

Table 8: Our implementation with 2 mappers - administrative divisions Great Britain(7 MB)

Shapefile(#)	0	1	2	3
Time(in seconds)	2.416	2.156	2.065	2.456

Table 9: Our implementation with 4 mappers - administrative divisions Great Britain(7 MB)

Shapefile(#)	0	1	2	3
Time(in seconds)	2.85	2.246	2.184	2.673

Table 10: Our implementation with 8 mappers - administrative divisions Greece(8 MB)

Shapefile(#)	0	1	2	3
Time(in seconds)	3.23	2.677	2.319	2.758

4.3.3 Administrative divisions Greece(4 shapefiles)

Table 11: Geotriples - administrative divisions Greece(8 MB)

Shapefile(#)	0	1	2	3
Time(in seconds)	2.327	2.418	2.328	4.071

Table 12: Our implementation with 2 mappers - administrative divisions Greece(8 MB)

Shapefile(#)	0	1	2	3
Time(in seconds)	3.246	2.448	2.083	2.847

Table 13: Our implementation with 4 mappers - administrative divisions Greece(8 MB)

Shapefile(#)	0	1	2	3
Time(in seconds)	2.897	2.354	2.032	2.907

Table 14: Our implementation with 8 mappers - administrative divisions Greece(8 MB)

Shapefile(#)	0	1	2	3
Time(in seconds)	3.526	2.291	2.589	3.15

4.3.4 Buildings Netherlands(1 shapefile)

Table 15: Geotriples - buildings Netherlands(1.8 GB)

Time (in seconds)	11418.635

Table 16: Our implementation - buildings Netherlands(1.8 GB)

Mappers	8
Time (in seconds)	5827.49

4.3.5 Graph Representation

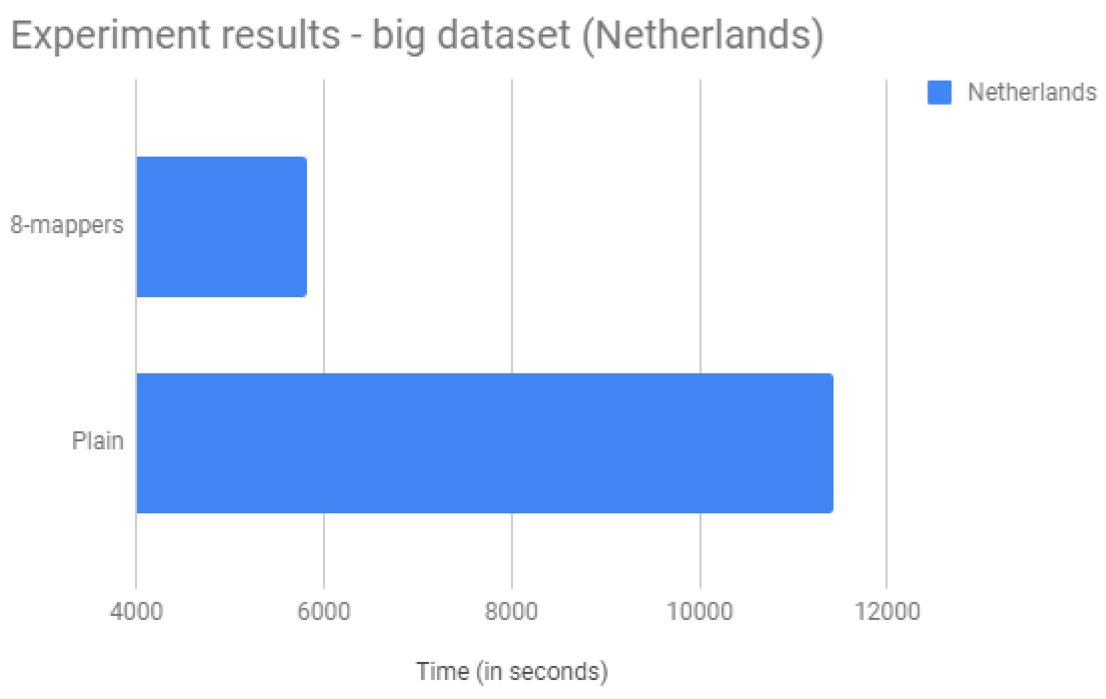


Figure 9: Bar chart of the experiment results for the Netherlands dataset (1.8 GB)

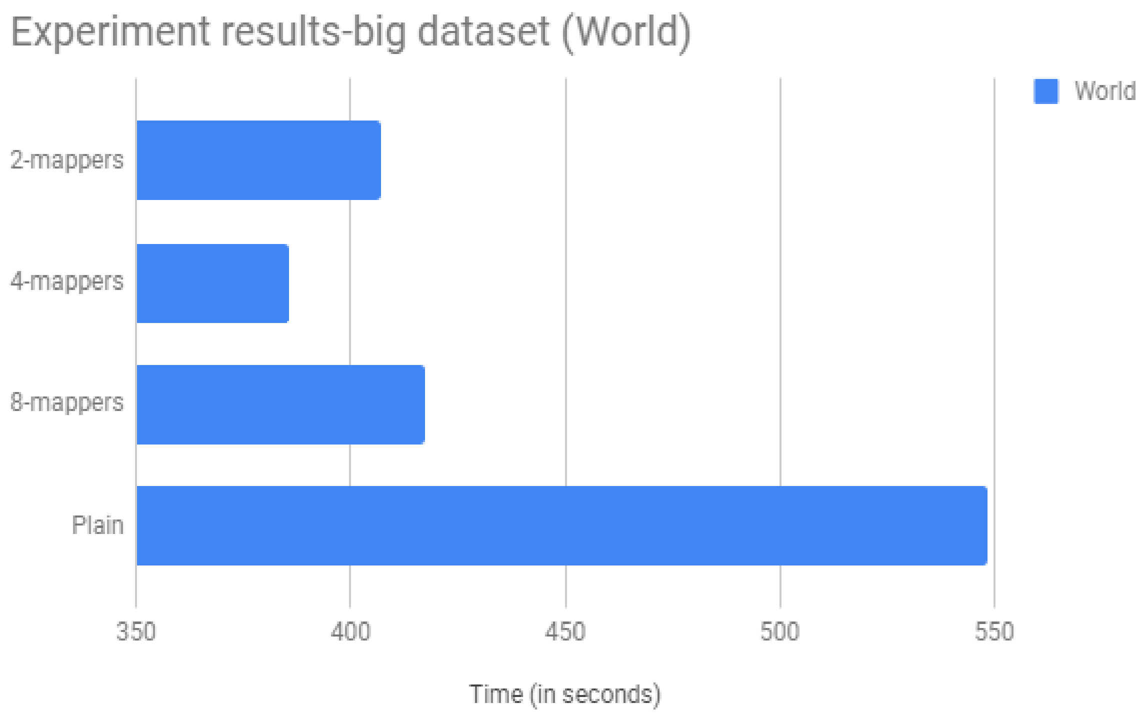


Figure 10: Bar chart of the experiment results for the World dataset (1.6 GB)

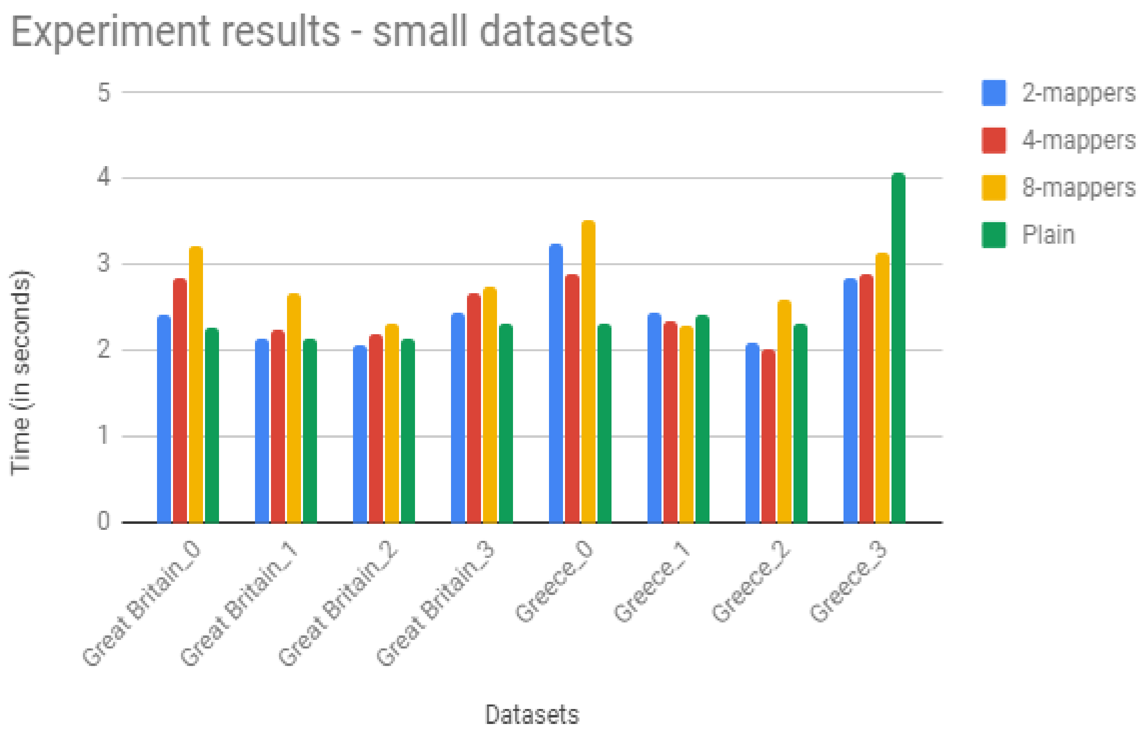


Figure 11: Column chart of the experiment results for the small datasets

4.3.6 Conclusion of the experimental evaluation

Our experiments show that the implementation we introduced in this thesis works really well when the input shapefile size is significant, since the overhead time that Apache SPARK introduces is minor compared to the completion time.

As shown in Figure 11, when the input shapefile size is small there is no benefit in using mapreduce. That is because the completion time of the program is comparable to the overhead in mapreduce and the time for preparation of the data. Though we can see that if we use only two mappers in our implementation, the time results are almost the same with the original implementation of GeoTriples.

While in small datasets we show that the increase in number of mappers only introduces overhead, that is not the case in big datasets. In Figure 10, we can see that moving to four mappers gives us the best results, while increasing the number to eight slows down the results. Picking the appropriate size for our job can radically change the performance. Increasing the number of tasks increases the framework overhead, but increases load balancing and lowers the cost of failures. In this case we see that four mappers offer the best results.

5.CONCLUSION

In this thesis we created a parallel implementation of the GeoTriples tool and ran experiments comparing our implementation with the original one.The results showcase that our implementation is especially effective when the input shapefile is of a significant size(eg. some gigabytes).In that case the time required for the operation to finish is (at best) half compared to the original Geotriples application.For smaller files GeoTriples performs better than our implementation.

We look forward to improving our application in the near future , so that it allows different input formats and displays even better results in terms of time complexity.

ABBREVIATIONS - ACRONYMS

DSM	Distributed Shared Memory
EO	Earth Observation
RDD	Resilient Distributed Datasets
RDF	Resource Description Framework
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
ΕΚΠΑ	Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

ANNEX I - APPLICATION CODE

```
public class geotriples
{
    public static class programParametres
    {
        public static Integer cores_num = 4;
        public static Integer defaultPieces = 2;
        public static Integer ramGb = 4;
        public static String getAvailableRam()
        {
            return ramGb.toString() + "g";
        }
    }

    public static void main(String[] args) throws Exception
    {

        SimpleFeatureSource featureSource;
        Integer pieces = null;
        String cores_num = programParametres.cores_num.toString();
        SparkConf conf = new 
SparkConf().setAppName("ShapefileRDDTest").setMaster("local[2]")
                                .set("spark.executor.cores", cores_num)
                                .set("spark.executor.memory", 
programParametres.getAvailableRam());

        JavaSparkContext sc = new JavaSparkContext(conf);//open the Spark 
connection

        JFrame frame = new JFrame("InputDialog Example #1");

        //User picks in how many pieces he wants the shapefile to be broken if 
no input is given a default value is used
        String userPreference = JOptionPane.showInputDialog(frame, "In how many 
pieces do you want to break the shapefile?");
        pieces = userPreference != null
                ? Integer.parseInt(userPreference)
                : programParametres.defaultPieces;

        File sourceFile = JFileDataStoreChooser.showOpenFile("shp", null);
        if (sourceFile == null)
        {
            return;
        }
        String saveDirectory = sourceFile.getParent();// the directory of the 
file
        saveDirectory=saveDirectory.replace("\\","/");
        String file_name = sourceFile.getName();//the name of the file
        file_name = FilenameUtils.removeExtension(file_name);

        FileDataStore store = FileDataStoreFinder.getDataStore(sourceFile);
        featureSource = store.getFeatureSource();

        //get the features of the given shapefile
        SimpleFeatureCollection featureCollection = featureSource.getFeatures();
        SimpleFeatureType schema = featureSource.getSchema();
        Integer remainingFeatures = featureCollection.size() % pieces;//in case 
the remainder is not 0 the last piece will get the rest of the features

        Integer attrPerPiece = featureCollection.size() / pieces;//calculate how
many features each piece of the original will get


        List<String> paths = new LinkedList<String>();//the paths of the pieces
        //iteration over each piece, so that each piece gets it's fair of 
features from the original shapefile
        long tStart = System.currentTimeMillis();
        for (int currentPiece = 0; currentPiece < pieces; currentPiece++) {
            String brokenFileName = file_name + "part" + currentPiece + 
".shp";//creating the new files/pieces by adding partX to the original name 
where X Integer
            //System.out.println(brokenFileName);
            paths.add(saveDirectory + "//" + brokenFileName);
            File file = new File(saveDirectory, brokenFileName);
            if (file.equals(sourceFile)) {
                JOptionPane.showMessageDialog(null, "Cannot replace " + file);
                return;
            }

            CoordinateReferenceSystem dataCRS = 
schema.getCoordinateReferenceSystem();
            DataStoreFactorySpi factory = new ShapefileDataStoreFactory();
            Map<String, Serializable> create = new HashMap<>();
            create.put("url", file.toURI().toURL());
            create.put("create spatial index", Boolean.TRUE);
            DataStore dataStore = factory.createNewDataStore(create);
            dataStore.createSchema(schema);

            String createdName = dataStore.getTypeNames()[0];
            Transaction transaction = new DefaultTransaction("Reproject");



            try (
                    FeatureWriter<SimpleFeatureType, SimpleFeature> writer = 
dataStore.getFeatureWriterAppend(createdName,
                    transaction);
                    SimpleFeatureIterator iterator = 
featureCollection.features()
                )
            {

                Integer attr_file_start = currentPiece * attrPerPiece;  //the 
first feature each piece will get
                Integer attr_per_file = (currentPiece == pieces - 1)  //how many 
features it will get
                                            ? attrPerPiece + remainingFeatures
                                            : attrPerPiece;

                IteratorMove(attr_file_start , iterator);       //move at the 
right position of the file    depending on which piece you're at
                WriteGeometryFilebyIterator(iterator , attr_per_file , writer
);//write the features to the piece

                transaction.commit();
                //JOptionPane.showMessageDialog(null, "Export to shapefile 
complete");
            }
            catch (Exception problem)
            {
                problem.printStackTrace();
                transaction.rollback();
                JOptionPane.showMessageDialog(null, "Export to shapefile 
failed");
            }
            finally
            {

                transaction.close();
            }
        }

        HashMap<Integer, String> hashMap = new HashMap<Integer, String>();
        JavaRDD<String> rdd = sc.parallelize(paths);//parallelize on paths


        //use the map function for each piece (each piece represented by it's 
path)

        JavaRDD<String> log_values = rdd.map( (Function<String,String>) x ->
            {
                String [] parts=x.split(".shp");
                String y=FilenameUtils.removeExtension(x);
                String gen_out=y+"gen.ttl";
                String dump_out=y+"dump.nt";

                //create a new file to use as the output of the generate_mapping 
GeoTriples function
                BufferedWriter gen_file = 
Files.newBufferedWriter(Paths.get(gen_out),
                        StandardCharsets.UTF_8);
                //create a new file to use as the output of the dump_rdf 
GeoTriples function
                BufferedWriter dump_file = 
Files.newBufferedWriter(Paths.get(dump_out),
                        StandardCharsets.UTF_8);


                //arguments for the GeoTriples functions
                String[] my_args= {"generate_mapping","-o",gen_out,"-b", 
"http://testdata.gr/",x };
                String[] my_args2= {"dump_rdf","-sh",x,"-o",dump_out,"-b", 
"http://testdata.gr/",gen_out};
                List <String> res=new ArrayList<String>() ;

                 //initialize the GeoTriples Command line and call the functions 
by passing the right arguments
                new GeoTriplesCMD();
                GeoTriplesCMD.main(my_args);
                GeoTriplesCMD.main(my_args2);
                return dump_out;//result RDF for the piece
            });

        //reduce all the RDF pieces into the final one, order does not matter
         String val=log_values.reduce((a,b)->
        {
            //append one RDF to the other and so on for all the pieces
             try {
             BufferedWriter out = new BufferedWriter(new FileWriter(a, true));

             BufferedReader in = new BufferedReader(new FileReader(b));
             String str;
             while ((str = in.readLine()) != null) {
                 out.write(str);
                 out.newLine();
                 }
             in.close();
             out.close();

            return a;
             }
             catch (IOException e) {
                 //System.out.println(e.getMessage());
                 return a;
             }});



        //antigrafoume to teliko arxeio ston teliko proorismo tou me to onoma
tou arxikou shapefile kai suffix RDF


         BufferedWriter final_file =
Files.newBufferedWriter(Paths.get(saveDirectory+"/"+file_name+"RDF.nt"),
                 StandardCharsets.UTF_8);



         BufferedReader in = new BufferedReader(new FileReader(val));
         String str;

         while ((str = in.readLine()) != null) {
             final_file.write(str);
             final_file.newLine();
             }
         in.close();
         final_file.close();


        sc.close();     //close the Spark connection
    }


    //Other functions

    //Iterates through a feature collection
    private static void IteratorMove ( Integer moveLength ,
SimpleFeatureIterator iterator )
    {
        Integer iteratorInt = 0;
        while ((iterator.hasNext()) && (iteratorInt < moveLength)) {
            iterator.next();
            iteratorInt++;
        }
    }


    //Copies features from one shapefile to another
    private static void WriteGeometryFilebyIterator ( SimpleFeatureIterator
iterator , Integer attrPerFile ,

FeatureWriter<SimpleFeatureType, SimpleFeature> writer)
                                                              throws Exception
    {
        try
        {
            Integer fileStart = 0;
            while ((iterator.hasNext()) && (fileStart < attrPerFile)) {

                fileStart++;
                SimpleFeature feature = iterator.next();

                SimpleFeature copy = writer.next();
                copy.setAttributes(feature.getAttributes());

                Geometry geometry = (Geometry) feature.getDefaultGeometry();

                copy.setDefaultGeometry(geometry);
                writer.write();
            }
            iterator.close();
            writer.close();
        }
        catch (Exception ex)
        {
            iterator.close();
            writer.close();
            throw new Exception(ex.getMessage());
        }
    }

}
```

REFERENCES

[1] Jamie Fishman, ”What is geospatial data—and why should businesses care about it?,November 23, 2015”;https://blog.westmonroepartners.com/what-is-geospatial-data-and-why-should-businesses-care-about-it/
[2] Weiming Huang,Ali Mansourian,Lars Harrie, “Geospatial data integration and visualisation using Linked Data”, AGILE 4th PhD school,At Leeds,UK, April 2018
[3] Kostis Kyzirakos,Ioannis Vlachopoulos,Dimitrianos Savva,Stefan Manegold and Manolis Koubarakis, “GeoTriples: a Tool for Publishing Geospatial Data as RDF Graphs Using R2RML Mappings”, 6th International Workshop on the Foundations, Technologies and Applications of the Geospatial Web, in conjunction with ISWC 2014. Riva del Garda, Trentino, Italy, October 19-23, 2014
[4] Kostis Kyzirakos,Ioannis Vlachopoulos,Dimitrianos Savva,Stefan Manegold and Manolis Koubarakis, “GeoTriples: a Tool for Publishing Geospatial Data as RDF Graphs Using R2RML Mappings”, In the Proceedings of the ISWC 2014 Posters & Demonstrations Track, pp 393-396. Riva del Garda, Trentino, Italy, October 21th, 2014
[5] Ian Pointer, “What is Apache Spark? The big data analytics platform explained”, Nov 13 2017; https://www.infoworld.com/article/3236869/analytics/what-is-apache-spark-the-big-data-analytics-platform-explained.html
[6] Abhishek Bhattacharya and Shefali Bhatnagar, Big Data and Apache Spark: A Review, International Journal of Engineering Research & Science (IJOER), vol. 2, Issue-5 May 2016
[7] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker and Ion Stoica, “Spark: Cluster Computing with Working Sets”, HotCloud 2010, 2010
[8] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker and Ion Stoica, “Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing”, nsdi’12, San Jose, CA, April 25-27, 2012
[9] Berkeley University of California, Introduction to Big Data with Apache Spark, 2015; https://prod-edxapp.edx-cdn.org/assets/courseware/v1/5a4e424feeb7b41d68219669833b6000/c4x/BerkeleyX/CS100.1x/asset/Week2Lec3.pdf
[10] 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. Morgan & Claypool Publishers, 2011
[11] Christian Bizer,Tom Heath and Tim Berners, “Linked Data - The Story So Far”, 2009; http://tomheath.com/papers/bizer-heath-berners-lee-ijswis-linked-data.pdf
[12] Arofan Gregory and Mary Vardigan , “The Web of Linked Data: Realizing the Potential for the Social Sciences”, 2010; http://odaf.org/papers/201010_Gregory_Arofan_186.pdf
[13] Eric Miller, An Introduction to the Resource Description Framework, D-Lib Magazine, May 1998
[14] *Richard Cyganiak,David Wood,Markus Lanthaler, RDF Concepts and Abstract Syntax,* World Wide Web Consortium (W3C) recommendation ,25 February 2014; https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/
[15] Kostis Kyzirakos,Ioannis Vlachopoulos,Dimitrianos Savva,Stefan Manegold and Manolis Koubarakis, “GeoTriples: Transforming geospatial data into RDF graphs using R2RML and RML mappings”, Journal of Web Semantics, 11 September 2018; https://www.sciencedirect.com/science/article/pii/S1570826818300428?dgcid=rss_sd_all