Scalable Transformation of Big Geospatial Data into Linked Data

George Mandilaras(✉) and Manolis Koubarakis(✉)

Department of Informatics and Telecommunications, National and Kapodistrian University of Athens, Athens, Greece
{gmandi,koubarak}@di.uoa.gr

Abstract. In the era of big data, a vast amount of geospatial data has become available originating from a large diversity of sources. In most cases, this data does not follow the linked data paradigm and the existing transformation tools have been proved ineffective due to the large volume and velocity of geospatial data. This is because none of the existing tools can utilize effectively the processing power of clusters of computers. We present the system GeoTriples-Spark which is able to massively transform big geospatial data into RDF graphs using Apache Spark. We evaluate GeoTriple-Spark's performance and scalability in standalone and distributed environments and show that it exhibits superior performance and scalability when compared to all of its competitors.

1 Introduction

A vast amount of geospatial data is now available on the Web, originating from a large diversity of sources like crowd-sourced projects (e.g., OpenStreetMap[1]), geospatial search engines like Google Maps, data hubs like the ESRI Open Data Hub[2] and Earth observation programs such as Copernicus[3]. As a result, researchers and practitioners working in Semantic Technologies have started transforming this big geospatial data into linked data, interlinking it with other data sources and further populating the Linked Open Data Cloud[4]. The project LinkedGeoData [3] was the first project to do this by collecting information from OpenStreetMap and converting it into linked data. Furthermore, projects such as TELEIOS [20], LEO [9], MELODIES [8] and Copernicus App Lab [4] have published several geospatial datasets that are Earth observation products, like CORINE Land Cover and the Urban Atlas[5]. This methodology has also been followed in the development of geospatial knowledge graphs like YAGO2geo [17],

1 https://www.openstreetmap.org/.
2 https://hub.arcgis.com/search.
3 https://www.copernicus.eu/.
4 https://lod-cloud.net/.
5 http://kr.di.uoa.gr/#datasets.

The present work was funded by the European Union's Horizon 2020 research and innovation project under grant agreement No 825258.

which extends YAGO2 [13] with precise geospatial information originated from multiple official government sources.

In many cases, geospatial data comes in large volumes and with high velocity. For example, this is the case in Earth observation programs such as Copernicus. Up to 2019 the Copernicus Open Access Hub[6] had published approximately 13 million Earth observation products, with a publication rate of over 30,500 products per day[7]. The size of such Earth observation products depends on the resolution of the image, and it can vary from a few megabytes (MB) to multiple gigabytes (GB). Data of such large scale requires special techniques and tools in order to process it efficiently. Despite the bulk of work on storage and querying of big RDF graphs [2,16], the scalable transformation of big geospatial data into linked data has been overlooked so far. The present paper attempts to close this gap.

Transforming geospatial data into linked data, enables users to leverage the power of ontologies for modeling the domain. Furthermore, users can interlink their data with other linked geospatial data using tools like the temporal and geospatial extension of Silk [31], RADON [30] or GIA.nt [27], pose GeoSPARQL queries by storing it into spatially-enabled triple stores such as Strabon [19] or GraphDB[8], and visualize it using visualization tools like Sextant [26]. One of the activities in the research project ExtremeEarth[9] [18] is to publish data extracted from Copernicus imagery into RDF graphs, so as to interlink it with other geospatial sources (e.g., in-situ observations), and provide it as linked open data [24]. At the moment though, there is no existing tool able to deal with the large scale of Copernicus data, and for this reason, we developed a new version of the tool GeoTriples [21] able to transform big geospatial data into linked data. In more details, the contributions of this paper are the following:

- We design and implement the system *GeoTriples-Spark*[10],[11], which is a new version of GeoTriples that runs on top of Apache Spark and enables the transformation of big geospatial data into linked data. GeoTriples-Spark is an open source project, licensed under the Apache license version 2.0.
- We evaluate our system using datasets of varying input sizes, in different scenarios, and compare its performance with its main competitors: GeoTriples-Hadoop [21] and the Spark-based implementation of TripleGeo [28]. We show that, in most cases, GeoTriples-Spark decreases the transformation time by approximately 40%. We also show that GeoTriples-Spark can transform terabytes of data in a reasonable amount of time when no other system has been proven to be able to do so.

[6] https://scihub.copernicus.eu/.
[7] https://scihub.copernicus.eu/twiki/do/view/SciHubWebPortal/AnnualReport2019.
[8] https://graphdb.ontotext.com/.
[9] http://earthanalytics.eu/.
[10] https://github.com/LinkedEOData/GeoTriples.
[11] https://zenodo.org/record/4899793.

To enable the reproduction of our experiments, all the relevant data and code is available in the repository of GeoTriples-Spark[12].

The structure of the rest of the paper is as follows. Section 2 discusses related work. Section 3 introduces the tool GeoTriples and its main components. In Sect. 4 we present GeoTriples-Spark, and in Sect. 5 we evaluate it against other systems. In Sect. 6 we sum up and present directions for future work.

2 Related Work

Geospatial data can exist in raster or vector forms. Raster data refers to images where each pixel is associated with a specific location and its colour may indicate a metric or a class. A well-known format for storing raster data is GeoTIFF, which is an industry-standard for images from GIS and satellite remote sensing applications. Vector data are made up of vertices and edges and are composed of three basic geometry types: points, lines and polygons. They are commonly available in formats such as ESRI shapefile, GeoJSON, KML and GML documents and in spatially-enabled RDBMS like PostGIS. CSV files can also store geospatial information, by containing complex geometric types expressed as Well Known Text[13] (WKT), a text markup language for representing vector geometries. In this work, we focus exclusively on vector data.

Two are the main approaches for the transformation of relational and non-relational data into RDF graphs: direct mapping and using mapping languages. Direct mapping[14] is a straightforward approach to map relational data into RDF. In direct mapping, the tables of the relational database become the classes, the column names are mapped into RDF properties that represent the relation between subject and object, the subjects of triples are formed using the primary key of each tuple, and the objects of triples are formed using the values for the rest of the columns of the table. In this approach, the generated triples are dictated by the initial schema of the relational data. Alternatively, the transformation using mapping languages allows us to define a set of mapping rules that indicate how to map the input data into RDF triples. There are two well-known mapping languages: *R2RML*[15] and *RML*[16] [11]. R2RML is also a W3C recommendation and it is used for expressing customized mappings to map relational databases into RDF graphs. RML is a more generic mapping language that can express rules able to map data from semi-structured (like XML and JSON) and structured formats into RDF graphs. Both mapping languages are very rich and enable the manipulation of the input data in numerous ways.

Historically, the first tool for transforming geospatial data into RDF was GEOMETRYtoRDF [23] which enabled users to transform data stored in spatially-enabled RDBMS into RDF graphs. GEOMETRYtoRDF mapped the

[12] https://github.com/LinkedEOData/GeoTriples/tree/master/data.
[13] https://www.ogc.org/standards/wkt-crs.
[14] https://www.w3.org/TR/rdb-direct-mapping/.
[15] https://www.w3.org/TR/r2rml/.
[16] https://rml.io/.

geometrical data into GML files which were then transformed into RDF triples using the open source libraries GeoTools[17] and Apache Jena[18]. Even though this project is no longer maintained, its code-base was the basis for the development of tool TripleGeo which is discussed below.

A different approach appears in [10] which shows how R2RML can be combined with a spatially-enabled relational database in order to transform geospatial data into RDF. However, the transformation of other geospatial data sources for vector data e.g., shapefiles is not discussed.

The closest existing tool to GeoTriples is *TripleGeo*[19] [28,29] which was developed in the project GeoKnow[20]. Similarly to GeoTriples, TripleGeo is a tool for transforming geospatial features from various sources into RDF graphs. TripleGeo supports the transformation of structured data (ESRI shapefiles, CSV, GeoJSON and GPX) or semi-structured data (XML, GML and KML), as well as from spatially-enabled DBMSs and of less standard formats such as OpenStreetMap data and certain INSPIRE data and metadata. Furthermore, recently in the project SLIPO[21], TripleGeo was further extended with several novel features and specific functionalities to efficiently support the transformation of large datasets containing Points of Interest (POIs). This was achieved by extending TripleGeo to run on top of Apache Spark. However, it was designed to run only in standalone mode and not in distributed environments, so it cannot utilize the processing power of clusters of computers. Therefore, TripleGeo cannot be used for the transformation of very big geospatial data that requires more than a single machine to process it.

We close this related work section by point out that there are applications where the data owners might not be willing to transform their geospatial data into RDF, but still want to use Semantic Technologies in their application. In this case, one cannot adopt the transformation-into-RDF paradigm of this paper, but can instead use the geospatial ontology-based data access paradigm pioneered by Ontop-spatial [5]. For example, [6] shows that you can leave geospatial data in their original vector or raster formats and still be able to query them using GeoSPARQL and the system Ontop-spatial[22].

3 GeoTriples

GeoTriples [21,22] is an open source tool developed by our team[23] in the National and Kapodistrian University of Athens, for the transformation of geospatial data into linked geospatial data. GeoTriples currently supports the transformation of spatially-enabled databases (PostGIS and MonetDB), ESRI shapefiles, XML

[17] https://geotools.org/.
[18] https://jena.apache.org/documentation/io/streaming-io.html.
[19] https://github.com/SLIPO-EU/TripleGeo.
[20] http://geoknow.eu/Welcome.html.
[21] http://slipo.eu/.
[22] http://ontop-spatial.di.uoa.gr/.
[23] http://ai.di.uoa.gr/.

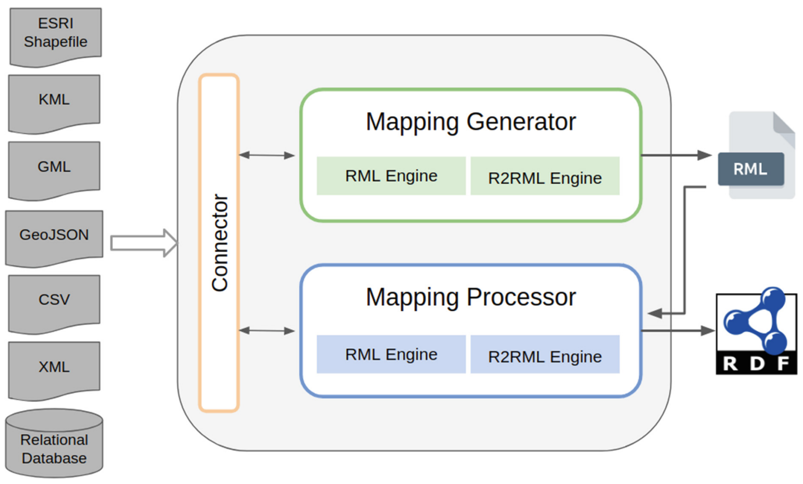


Fig. 1. The system architecture of GeoTriples

documents (hence GML documents), KML, GeoJSON and CSV documents. The produced graph is by default compliant with the GeoSPARQL vocabulary and can be manifested in any of the popular RDF syntaxes such as Turtle, RDF/XML, Notation3 or N-Triples.

GeoTriples consists of two components: a mapping generator that, given an input file, it generates a mapping file containing the mapping rules, and a mapping processor that applies the mapping rules in order to map each instance of the input data into the corresponding RDF triples. Additionally, the first component of GeoTriples is a connector that provides an abstraction layer and allows the other components to transparently access the input data regardless of the format of the source. Figure 1 displays a simplified diagram of GeoTriples system architecture.

The execution of GeoTriples comprises three steps. In the first step, we use the mapping generator to create a mapping file containing the mapping rules. Then, as a second optional step, the user can edit the mapping file so the produced triples will adopt any vocabulary or ontology she wants. Finally, the last step follows the transformation of the input file, in which the mapping processor applies the mapping rules to map the input data into RDF triples.

The mappings produced by the mapping generator consist of two triples maps: one for handling non-geometric (thematic) data, and one related to the geospatial data. The triples map that handles the thematic information defines a logical table that contains the attributes of the input data source and a unique identifier for the generated instances. Combined with a URI template, the unique identifier is used to produce the subjects of the produced triples. For each attribute of the input data, GeoTriples generates a term map that defines an RDF predicate according to the name of the attribute and a predicate-object map. The triples map that handles the geospatial information defines a logical table with a unique identifier similar to the thematic one. The logical table contains term maps that indicate the serialization of the geometric information according to the WKT representation, and the generation of all the necessary attributes for producing a GeoSPARQL compliant RDF graph. Hence, if the input is an ESRI shapefile, GeoTriples constructs RML mappings with transformations that invoke GeoSPARQL/stSPARQL extension functions. If the input is a relational database, GeoTriples constructs SQL queries that utilize the appropriate spatial functions of the Open Geospatial Consortium[24] standard [12] to generate the information required.

At the beginning of the transformation, GeoTriples parses the input mappings and extracts the content of the logical table using the appropriate way (e.g., a SELECT query). If the subject map is a template-valued term map or a column-valued term map, the related columns are extracted and stored in memory. Then, the processor iterates over all predicate-object maps, and for each one, it extracts all template- and column-valued term maps. These term maps are cached in memory along with the position that they appear in. Afterwards, the processor extracts all the features that are referenced by the term maps that appear in the subject, predicate and object positions for the current predicate map and iterates over the results. For each predicate and object value in the result row, a new RDF triple is constructed.

4 GeoTriples-Spark

Apache Spark[25] [34] is an open source, distributed, general-purpose, cluster computing framework that uses a *master/worker* architecture. There is a *Driver* process that is responsible to split the job into tasks, to schedule them to run on *Executors* and to coordinate the overall execution. Executors are distributed agents that execute tasks in parallel or sequentially. At the core of Apache Spark is the notion of data abstraction as a distributed collection of objects, known as *Resilient Distributed Datasets* (RDDs) [33]. RDDs allow the user to apply a series of transformations (i.e., map, filters, etc.), creating a lineage graph that will not be executed before calling an action (i.e., count, write to file, etc.). All transformations and actions are performed in parallel, as each Executor is assigned with a portion of the overall data known as partition and the execution of the transformation linkage graph runs inside a task.

GeoTriples-Spark is a new version of GeoTriples that runs on top of Apache Spark and enables the massive transformation of big geospatial data into RDF graphs. The input big geospatial data can be provided as multiple separate files which will be transformed concurrently or as a big single file. Currently, GeoTriples-Spark supports the transformation of CSV, GeoJSON and ESRI shapefiles and it can run in any standalone or distributed environment that supports the execution of Apache Spark jobs. Figure 2 displays the architecture of GeoTriples-Spark.

[24] https://www.ogc.org/standards/sfs.
[25] https://spark.apache.org/.

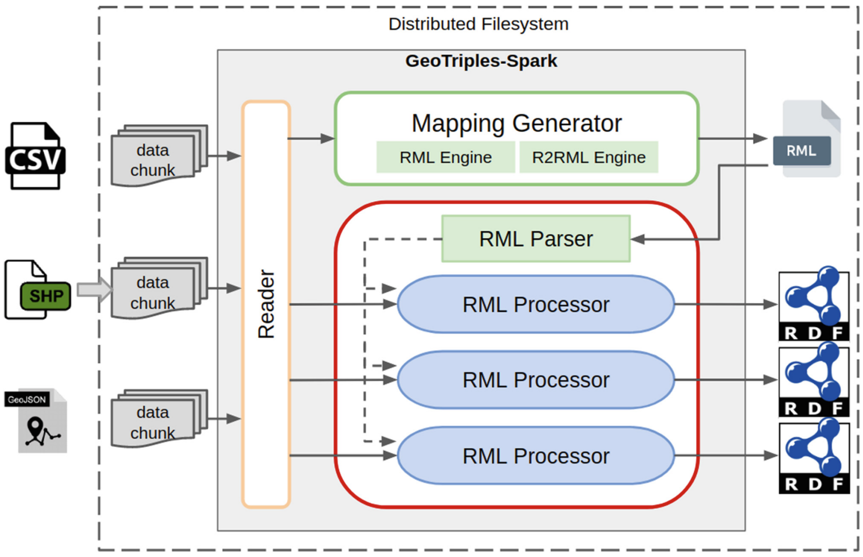


Fig. 2. The GeoTriples-Spark architecture

The mapping generator is the same as the one in the original GeoTriples system. However, the mapping processor is new and makes effective use of Apache Spark as we explain below.

The first component of GeoTriples-Spark is the Reader, which employs the appropriate libraries in order to load the input data into an Apache Spark `Dataset`, which is a distributed immutable collection of data organized into named columns. When a dataset is stored in a distributed filesystem (like HDFS), it is split into multiple chunks of constant size. When GeoTriples-Spark starts, the Reader loads these data chunks into partitions, which will be transformed as individual units in parallel. Users can change the default number of partitions in order to increase parallelization, but this can invoke data shuffling. To load ESRI shapefiles and GeoJSON, we use the library *GeoSpark*[26] [32] which is a cluster computing framework that extends Apache Spark with spatial computations. After initializing the `Dataset`, a new column is inserted containing a monotonically increasing unique index which will be combined with a URI template to form the subjects of produced triples. This is the default way of constructing subjects, but the user can overwrite it by editing the mapping file. Moreover, this generated index is not consecutive, as this would require to have counted all the records of the input data, which would add an overhead to the execution. Additionally, before the transformation starts, the Reader also loads and extracts the mapping rules from the mapping file, and broadcasts them so they will be available in all nodes.

[26] http://geospark.datasyslab.org/.

The transformation starts with a Map phase where for each partition, an RML processor is initialized, responsible for transforming all the records of the input partitions into RDF triples. Each RML processor iterates over all the records of its input partition and applies the mapping rules extracted by the mapping file, generating the corresponding triples. Note that, in most cases, the size of the produced triples exceeds the size of the initial dataset, which is sensible due to the nature of RDF triples. Hence, we avoid performing a Reduce and collect the produced triples into a single node, as this can overwhelm the targeted node and lead to memory errors. Consequently, after the transformation, the triples of each partition are stored in an individual file, thus the produced triples are stored into multiple files, one for each partition. Therefore, the whole procedure is a highly parallelized one as each process works individually from the rest and it simply loads its corresponding partitions, applies the mapping rules and stores the generated triples in a file.

Note that except for the broadcast of the mapping rules, there is no need for further data shuffling during the whole procedure as each partition already contains all the necessary information to perform the transformation. Additionally, when the produced triples of a record are generated, they are directly written in the target file. Consequently, the whole procedure is memory friendly, as it needs to maintain in the memory neither the initial data nor the produced triples. This makes GeoTriples-Spark highly scalable and able to transform a large amount of data with minimum memory requirements.

The parallelization of the whole procedure is based on the number of partitions and available resources (i.e., the physical processing units). More partitions mean the more parallelized the whole procedure can be (according to the number of the concurrently executed threads), but it also means that each process will have to transform a smaller chunk of data, as the initial dataset will have been divided into more partitions of fewer records. The number of partitions is determined by the size and the format of the source, and by the configuration settings of the filesystem, but it can also be configured by the user. However, increasing the initial number of partitions can invoke data shuffling, which in a distributed cluster can invoke network and disk I/O, which in its turn can affect negatively the performance of the system.

The transformation of CSV and GeoJSON documents is very similar. These filetypes are considered text files, and therefore they are directly loaded as multiple partitions by Spark, as they are distributedly stored across the cluster. The geometry feature in CSV files is expected to be in WKT and hence it does not require any further processing. Regarding GeoJSON, the system loads them as simple JSON files that follow the GeoJSON specification of RFC 7946[27]. In GeoJSON, the information is stored as a collection of geometric features consisting of the geometries and the thematic properties. The geometries are specified by the type (e.g., Point, LineString, Polygon, etc.) and by a list of coordinates, and the thematic properties are defined by a set of key-value pairs. So, using Spark's API, we load the properties of the geometric features into an Apache

[27] https://tools.ietf.org/html/rfc7946.

Spark `Dataset`, and then we extract and transform the geometries into WKT, which we add to the `Dataset`. Finally the partition-wise transformation of the `Dataset` into RDF triples follows.

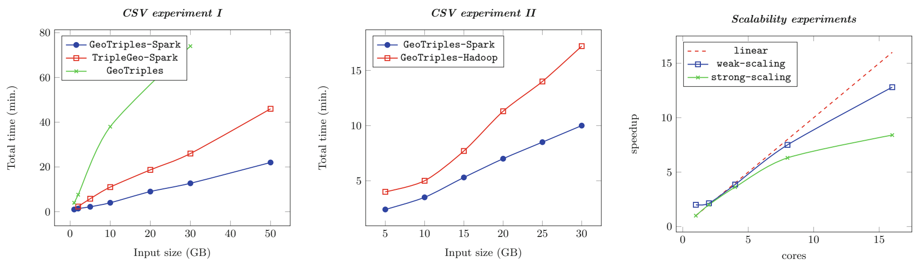
The case of ESRI shapefiles is a bit more complicated. As mentioned, we load ESRI shapefiles using GeoSpark, whose shapefile reader always loads the input shapefile into a single partition[28]. Since shapefiles are composed of multiple files, in order to load them, we first need to merge all the related component files into one. This is happening because all the related component files must be located in the same node to utilize the shapefile index and the related attributes. Loading shapefiles from distributed filesystems is a well-known problem and it has been studied extensively in [1]. Hence, if users want to parallelize the transformation of a single shapefile, they must repartition the loaded data in order to redistribute it into multiple partitions. This will probably invoke data shuffling which may have a negative effect on the performance. However, shapefiles are typically small, and actually, there is a 2 GB size limit for any of its component files[29]. Additionally, it is very common to store geospatial data as multiple shapefiles. Thus, we have enabled GeoTriples-Spark to be able to transform multiple shapefiles concurrently, by loading them as individual partitions that will be transformed in parallel.

5 Evaluation

For the evaluation of GeoTriples-Spark, we compare it with three competitors systems: the centralized version of GeoTriples, the Hadoop-based implementation of GeoTriples[30] and the Spark-based implementation of TripleGeo[31] which we refer to as TripleGeo-Spark. The following experiments concern the performance of the systems against varying input sizes, the scalability of GeoTriples-Spark and also its performance regarding the transformation of very big geospatial data into RDF. For the comparison with GeoTriples-Hadoop using shapefiles, we reproduce the same experiments presented in [21], while for the comparison using CSV files we perform large-scale experiments with bigger datasets.

For the experiments, we used three different environments, a Hadoop cluster, a standalone machine that runs Apache Spark, and a large-scale cluster that runs the Hospworks [15] platform. The main module of Hopsworks is Hops[32] , which is a next generation distribution of Apache Hadoop, using a new implementation of HDFS called HopsFS [25]. Since TripleGeo is designed to run only in standalone mode, it can neither read the configuration file nor write the output triples in a distributed filesystem. As a result, TripleGeo can run neither in Hopsworks

[28] https://github.com/DataSystemsLab/GeoSpark/issues/356.
[29] https://desktop.arcgis.com/en/arcmap/latest/manage-data/shapefiles/geoprocessing-considerations-for-shapefile-output.htm.
[30] https://github.com/dimitrianos/GeoTriples-Hadoop.
[31] https://github.com/SLIPO-EU/TripleGeo/tree/master/src/eu/slipo/athenarc/triplegeo/partitioning.
[32] https://github.com/hopshadoop/hops.



(a) Performed in standalone machine

(b) Performed in Hadoop cluster

(c) Performed in standalone machine

Fig. 3. CSV experiments

nor in the Hadoop cluster. Therefore, we compare the Spark- and Hadoop-based implementations of GeoTriples in the Hadoop cluster, and we compare the Spark-based implementations of GeoTriples and TripleGeo in the standalone machine. Last, we use the Hospworks cluster to perform large-scale experiments.

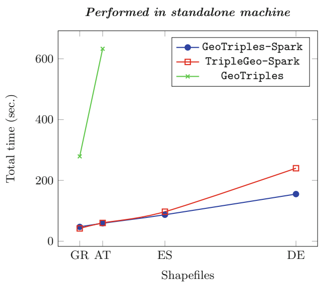
The Hadoop cluster consists of four nodes with 8 cores each with Intel(R) Xeon(R) E5-2650 v3 CPU at 2.30 GHz and 8 GB of memory. The standalone machine contains 32 virtual cores[33] at 2.20 GHz and 128 GB of memory. The large-scale cluster is a very powerful cluster provided to us by the company Logical Clocks[34], containing approximately 1000 CPU cores at 2.40 GHz, 12TB of RAM and 1PB of storage. Some of the data used in the experiments are from the Global Administrative Areas dataset[35] (GADM) while the rest are extracts of the OpenStreetMap project that are publicly available from the company GEOFABRIK[36]. Moreover, we further edit and replicate the datasets, in order to increase the input size. To enable the reproduction of the experiments, all the datasets are available in the repository of GeoTriples-Spark.

Figures 3a and 3b show the performance of GeoTriples-Spark for varying input CSV file sizes against the Hadoop-based implementation of GeoTriples and the Spark-based implementation of TripleGeo. In the experiment of Fig. 3a, both GeoTriples-Spark and TripleGeo-Spark, load the input data as 32 partitions which are transformed concurrently by 32 tasks. In the experiment of Fig. 3b, we did not change the initial number of loaded partitions of the datasets, as it would invoke network I/O which we wanted to avoid. In both experiments of Fig. 3, GeoTriples-Spark outperforms its competitors and we can also observe that as the size of input data increases, the effectiveness of GeoTriples-Spark becomes even clearer, particularly for the last datasets where the execution time decreases up to 47% compared to TripleGeo-Spark and 42% compared to GeoTriples-Hadoop. The results are similar when using GeoJSON documents as input.

[33] The system uses hyper-threading hence it has 16 physical cores.
[34] https://www.logicalclocks.com/.
[35] https://gadm.org/.
[36] http://download.geofabrik.de/.

Dataset	*Size*
GR	440MB
AT	764MB
ES	1.7GB
DE	3.7GB

(a) Shapefile datasets



(b) SHP Experiment

Fig. 4. ESRI shapefiles experiments: Transformation of big shapefiles

This difference in the performance of the two systems derives from their implementation differences. First of all, GeoTriples-Spark uses an extended RML processor, while the transformation of TripleGeo-Spark is based on StreamRDF of Apache Jena. Furthermore, the execution of GeoTriples-Spark is very straightforward as it simply reads the input data, performs the transformation by applying the mapping rules, and stores the produced triples directly in the files. All of these steps are performed natively using Apache Spark's API. On the other hand, TripleGeo-Spark performs partition-wise transformation, and stores the results after transforming batches of input records, maintaining intermediate results in memory. Moreover, the writing procedure is not implemented natively using Spark's interface, but using Jena's StreamRDF writers. Last but not least, TripleGeo-Spark computes and outputs extra attributes derived from geometries, like the area of polygons and the length of lines. This adds an extra overhead in the execution, as such computations are expensive especially for big geometries.

Figure 3c depicts the scalability experiments with regards to *strong* and *weak* scaling[37]. In *strong scaling*, we examine how the overall computational time of the job scales as we increase the number of available processing cores. In *weak scaling*, we examine the speedup while increasing both the job size and the number of processing elements. In the strong scaling experiment, the size of the job is 15 GB, while in the weak scaling, the input size is equivalent to the number of active cores (i.e., 2 cores $\rightarrow$ 2 GB, 4 cores $\rightarrow$ 4 GB). In weak scaling, we can observe that the execution is almost linear but we can notice that there is a small deceleration as the number of cores increases. We observe similar in the strong scaling experiment. The main reason for this is because the Executors read and write in the same disk, hence more active cores lead to bigger latencies in disk I/O.

[37] https://www.kth.se/blogs/pdc/2018/11/scalability-strong-and-weak-scaling/.

Table 1. ESRI Shapefile experiments: Transformation of multiple shapefiles of varying sizes

Dataset	Size (MB)	Times loaded	GeoTriples-Spark (sec.)	GeoTriples-Hadoop (sec.)
Andorra	888	15	345	370
Australia	247	60	382	499
Ukraine	2	1000	428	1002

For the experiments with ESRI shapefiles, we evaluate the performance of GeoTriples-Spark in two kinds of experiments. In the first experiment, we compare the performance of GeoTriples-Spark and TripleGeo-Spark in the transformation of big ESRI shapefiles. The shapefiles are displayed in Table 4a and contain information of the road system of four countries (Greece, Austria, Spain and Germany) originated from OSM. In most cases, ESRI shapefiles are relatively small because they are compressed database files. So, to create bigger ones, we merge multiple shapefiles into one. The largest shapefile we use (i.e., *DE*) contains the whole road-system of Germany and it was created by merging the shapefiles of the road-system of the states of Germany. In these experiments, both tools repartition the input data into 32 partitions which are all transformed in parallel. The results are presented in Fig. 4b. Both systems perform well and quite similarly, but in the last and largest dataset, GeoTriples-Spark outperforms TripleGeo-Spark, as it requires 62.5% of the time TripleGeo-Spark needs to transform it. This performance benefit becomes even more distinctive as we increase the size of the input.

In the second experiment, we examine and compare the performance of the Spark- and Hadoop-based implementations of GeoTriples regarding the transformation of multiple shapefiles concurrently. Similarly to GeoTriples-Spark, GeoTriples-Hadoop loads the data of a shapefile into a single mapper, but in contrast with the Spark implementation, GeoTriples-Hadoop cannot re-distribute the load to other mappers, as it is mentioned in [21]. Therefore, GeoTriples-Hadoop is good for transforming multiple shapefiles where each one is assigned to a different mapper, but it is incapable of transforming shapefiles where their size exceeds the available memory of mappers. In this experiment, we load three different shapefiles of varying sizes multiple times, in order to evaluate how the two systems perform when the goal is to transform multiple small (Ukraine), medium (Australia) and large files (Andorra). The results are displayed in Table 1 and we can see that both tools perform similarly regarding the big and the medium shapefiles, with GeoTriples-Spark performing slightly better. However, we observe a significant difference in the last dataset where GeoTriples-Spark is superior as it requires less than 50% of the time GeoTriples-Hadoop needs.

Let us now present our large-scale experiments shown in Tables 2 and 3. For the experiments with CSV documents, we constructed a dataset of size up to 250 GB, which we load multiple times. Likewise, for the experiments with

Table 2. Large-scale experiments with CSV documents

Dataset	Times loaded	Input Size	#Executors	Output Size	Total time (in minutes)
100 GB.csv	1	100 GB	41	840.1 GB	3.3
250 GB.csv	1	250 GB	60	2.1 TB	6.6
250 GB.csv	2	500 GB	65	4.1 TB	13
250 GB.csv	4	**1 TB**	70	8.3 TB	26
250 GB.csv	8	**2 TB**	80	16.6 TB	50

Table 3. Large-scale experiments with ESRI shapefiles

Dataset	Times loaded	Input Size	#Executors	Output Size	Total time (in minutes)
AT	153	100 GB	20	427.7 GB	4.3
AT	381	250 GB	30	1068.6 GB	9.9
DE	136	500 GB	15	2.5 TB	17
DE	258	**1 TB**	27	5.1 TB	34

ESRI shapefiles, we load the AT and DE shapefiles multiple times. The memory requirements of each Executor are the minimum, as neither the input data nor the generated triples are cached in memory. Furthermore, there is no need for large Spark execution memory[38] since there is little to none data shuffling. So, in these experiments, we equipped each Executor with 2 GB of memory. In the end, we managed to transform 2TB of CSV input in less than an hour and 1TB consisting of multiple shapefiles in less than half an hour.

An important issue that arises with very large input files is the size of the output files, as this is approximately eight times bigger than the initial input. To solve this issue, we are streaming the produced triples directly in a distributed geospatial triple store we are currently developing [7], instead of writing them on the disk. This will facilitate access to the produced graphs and will enable us to pose GeoSPARQL queries efficiently.

To ensure the quality of the output and to verify that the produced graphs are the expected ones, we performed limited quality control. To do this, we stored the produced graph of the smallest of the large-scale experiments in a spatially-enabled triple store and the initial data into a spatially enabled database. Then we posed a series of queries to both stores and we validated the correctness of the results using a Geographic Information Systems (GIS). This way we confirmed that neither the geometries nor the thematic information has altered in any way by the transformation. Additionally, we also deduced that all the necessary links/predicates of the graph were generated, as otherwise, it would have not produced the correct results.

[38] https://spark.apache.org/docs/latest/tuning.html.

6 Summary and Future Work

In this work, we presented GeoTriples-Spark, which is a new version of GeoTriples, able to transform big geospatial data into RDF. We also evaluated its performance against the original version of GeoTriples, the Spark-based implementation of TripleGeo and GeoTriples-Hadoop. GeoTriples-Spark not only outperforms its competitors, but we also show that it is capable of transforming up to terabytes of input data, in a reasonable amount of time. GeoTriples-Spark is used in the project ExtremeEarth in order to transform data extracted from satellite images into linked data.

In a use case scenario of ExtremeEarth [24], we first download satellite images that cover areas in the Polar regions. Then, using deep learning techniques, we extract information and store it as multiple shapefiles. Then, we transform these multiple shapefiles into RDF concurrently using GeoTriples-Spark and interlink them with other geospatial datasets containing in-situ observations. Finally, we store the produced triples into a distributed geospatial RDF store which is currently under development by our group. The goal is to be able to run the whole pipeline in real-time. GeoTriples-Spark is designed especially for such use cases where one needs to transform multiple CSV files or shapefiles concurrently in an efficient way.

As for future work, we plan to extend GeoTriples-Spark in order to be able to transform data from other geospatial sources like big KML and GML documents, and from systems that are built on top of Hadoop, like Apache Hive[39] and Apache Accumulo[40]. Moreover, we plan to extend both GeoTriples and GeoTriples-Spark to support the GeoSPARQL+ [14] vocabulary, which enables handling raster geospatial data.

References

1. Abdul, J., Alkathiri, M., Potdar, M.B.: Geospatial Hadoop (GS-Hadoop) an efficient mapreduce based engine for distributed processing of shapefiles. In: ICACCA (2016)
2. Ali, W., Saleem, M., Yao, B., Hogan, A., Ngomo, A.N.: A Survey of RDF Stores & SPARQL Engines for Querying Knowledge Graphs. CoRR (2021)
3. Auer, S., Lehmann, J., Hellmann, S.: LinkedGeoData: adding a spatial dimension to the web of data. In: ISWC (2009)
4. Bereta, K., et al.: The copernicus app lab project: easy access to copernicus data. In: EDBT (2019)
5. Bereta, K., Koubarakis, M.: Ontop of geospatial databases. In: ISWC (2016)
6. Bereta, K., Koubarakis, M.: Creating virtual semantic graphs on top of big data from space. In: BiDS (2017)
7. Bilidas, D., Ioannidis, T., Mamoulis, N., Koubarakis, M.: Efficient storage and querying for big linked geospatial data: the system Strabo 2. Manuscript in preparation (2021)

[39] https://hive.apache.org/.
[40] https://accumulo.apache.org/.

8. Blower, J., Clifford, D., Goncalves, P., Koubarakis, M.: The MELODIES project: integrating diverse data using linked data and cloud computing. In: BiDS (2014)
9. Burgstaller, S., et al.: LEOpatra: a mobile application for smart fertilization based on Linked Data. In: HAICTA (2017)
10. Chentout, K., Vaisman, A.A.: Adding spatial support to R2RML mappings. In: OTM Workshops (2013)
11. Dimou, A., Sande, M.V., Colpaert, P., Verborgh, R., Mannens, E., de Walle, R.V.: RML: a generic language for integrated RDF mappings of heterogeneous data. In: LDOW (2014)
12. Herring, J.R.: OpenGIS Implementation Standard for Geographic information - Simple feature access - Part 2: SQL option. Open Geospatial Consortium standard (2010). http://portal.opengeospatial.org/files/?artifact_id=25354
13. Hoffart, J., Suchanek, F.M., Berberich, K., Weikum, G.: YAGO2: A Spatially and Temporally Enhanced Knowledge Base from Wikipedia (2013)
14. Homburg, T., Staab, S., Janke, D.: GeoSPARQL+: syntax, semantics and system for integrated querying of graph, raster and vector data. In: Pan, J.Z., et al. (eds.) ISWC 2020. LNCS, vol. 12506, pp. 258–275. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-62419-4_15
15. Ismail, M., Gebremeskel, E., Kakantousis, T., Berthou, G., Dowling, J.: Hopsworks: improving user experience and development on hadoop with scalable. ICDCS, strongly consistent metadata. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS) (2017)
16. Kaoudi, Z., Manolescu, I., Zampetakis, S.: Cloud-Based RDF Data Management (2020)
17. Karalis, N., Mandilaras, G.M., Koubarakis, M.: Extending the YAGO2 knowledge graph with precise geospatial knowledge. In: ISWC (2019)
18. Koubarakis, M., et al.: From copernicus big data to extreme earth analytics. In: EDBT (2019)
19. Kyzirakos, K., et al.: The spatiotemporal RDF store strabon. In: SSTD (2013)
20. Kyzirakos, K., et al.: Wildfire monitoring using satellite images, ontologies and Linked Geospatial Data. J. Web Semant. **24**, 18–26 (2014)
21. Kyzirakos, K., et al.: GeoTriples: transforming geospatial data into RDF graphs using R2RML and RML mappings. J. Web Semant. **52**, 16–32 (2018)
22. Kyzirakos, K., Vlachopoulos, I., Savva, D., Manegold, S., Koubarakis, M.: GeoTriples: a tool for publishing geospatial data as RDF graphs using R2RML mappings. In: ISWC Posters (2014)
23. de León, A., Saquicela, V., Vilches, L.M., Villazón-Terrazas, B., Priyatna, F., Corcho, O.: Geographical linked data: a spanish use case. In: I-SEMANTICS (2010)
24. Mandilaras, G., Pantazi, D.A., Koubarakis, M., Hughes, N., Everett, A., Kiærbech, A.: Ice monitoring with extremeearth. In: LASCAR (2020)
25. Niazi, S., Ismail, M., Haridi, S., Dowling, J., Grohsschmiedt, S., Ronström, M.: HopsFS: scaling hierarchical file system metadata using NewSQL databases. In: FAST (2017)
26. Nikolaou, C., et al.: Sextant: visualizing time-evolving linked geospatial data. J. Web Semant. **35**, 35–52 (2015)
27. Papadakis, G.A., Mandilaras, G., Nikos, M., Koubarakis, M.: Progressive, holistic geospatial interlinking. In: Web Conference (2021)
28. Patroumpas, K., Alexakis, M., Giannopoulos, G., Athanasiou, S.: TripleGeo: an ETL Tool for transforming geospatial data into RDF triples. In: EDBT/ICDT (2014)

29. Patroumpas, K., Skoutas, D., Mandilaras, G.M., Giannopoulos, G., Athanasiou, S.: Exposing points of interest as linked geospatial data. In: SSTD (2019)
30. Sherif, M.A., Dreßler, K., Smeros, P., Ngomo, A.N.: Radon - rapid discovery of topological relations. In: AAAI (2017)
31. Smeros, P., Koubarakis, M.: Discovering spatial and temporal links among RDF data. In: LDOW (2016)
32. Yu, J., Wu, J., Sarwat, M.: GeoSpark: a cluster computing framework for processing large-scale spatial data. In: SIGSPATIAL (2015)
33. Zaharia, M., et al.: Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: USENIX (2012)
34. Zaharia, M., et al.: Apache spark: a unified engine for big data processing. Commun. ACM **59**(11), 56–65 (2016)