Dataspace Management with ETL and RDF Support

Marko Niinimaki[1]*, Tapio Niemi[2] and Peter Thanisch[3]

[1]Department of Computer Science, Faculty of Business and Technology, Webster University, Bangkok, 10120, Thailand

[2]Department of Operations, University of Lausanne, Lausanne, 1015, Switzerland

[3]School of Information Sciences, University of Tampere, Tampere, 33100, Finland

* Corresponding author. E-mail address: niinimakim@webster.ac.th

Received: 19 February 2020; Revised: 15 May 2020; Accepted: 26 May 2020; Available online: 11 June 2020

Abstract

Dataspaces have become popular in data modeling and business intelligence. In this paper, we introduce a dataspace management system with Extract-Transform-Load capabilities and RDF (Resource Description Framework) data export. Moreover, we demonstrate how distributed processing based on the MapReduce framework can be used in processing the exported RDF data.

Specifically, our system helps the user (i) discover potential problems with data integration and then (ii) carry out the actual integration. In the first case, the software generates a file of comma separated values that the users can load into their statistics software. In the second case the user can transform the files into the RDF format and analyze the data using Python tools, or export the final data set to a visualization package or business intelligence software. Our method can therefore be seen as constructive; building a tool for data professionals.

We demonstrate the viability of both the dataspace management system and RDF processing using a Hadoop cluster. There, using Hadoop distribution improved the processing speed by 85..86%.

Keywords: dataspace, MapReduce, RDF, cloud, Hadoop.

Introduction

Building a data collection for analysis can be a daunting task. The data needed can be "big": large, ever changing and in different formats. Moreover, it may be difficult to identify the attributes by which data sets can be linked together, or the coding of data can be different, for example ISO2 vs ISO3 country abbreviations. In this paper, we identify some of these difficulties and describe a tool that we have built for analysts to overcome them. The tool utilizes methods developed in dataspace research. Dataspaces can be informally described as collections of data sharing a theme.

Big Data has indeed been loosely defined as "the information asset characterized by such a high Volume, Velocity and Variety to require specific technology and analytical methods for its transformation into value" (Mauro, Greco, & Grimaldi, 2016). A specific challenge with Big Data is integration of data from multiple sources. Dong and Srivastava (2013) summarize different aspects of the challenge as follows: (i) the number of data sources has grown to be in the tens of thousands, (ii) many of the data sources are dynamic, (iii) the data sources are extremely heterogeneous in their structure, and (iv) the sources exhibit widely differing levels of data quality, having differences in the coverage, accuracy and timeliness of their data.

Yet, computer scientists working with statistical data have been aware of these kinds of problems for a long time. Though data integration methods have improved (Halevy, Franklin, & Maier, 2006), processing data in an integrated business intelligence platform often requires a non-trivial process that involves (i) identifying data sources, (ii) discovering the meaning of data in each of the sources, (iii) integrating the data by using a harmonized data format and (iv) loading it into a system that can be used to analyze it. This process is called

ETL (Extract-Transform-Load) (Rahm & Do, 2000), and the data resulting from it is often stored in a data warehouse server. When needed by an analyst, the data is then presented as On-Line Analytic Processing (OLAP) cubes (Chaudhuri, Dayal, & Narasayya, 2011).

Dataspaces were introduced to manage a situation where there is "some identifiable scope and control across the data and underlying systems, and hence one can identify a space of data, which, if managed in a principled way, will offer significant benefits to the organization" (Franklin, Halevy, & Maier, 2005). A dataspace support platform (DSSP) would provide services to manage these collections of data. Specifically, these services should help identify sources in a dataspace, inter-relate them, and offer basic query mechanisms over them, including the ability to introspect about their contents (Halevy, Franklin, & Maier, 2006). IMeMex (Antonio, Salles, & Dittrich, 2006) is maybe the best-known dataspace system for managing personal data, but its design does not directly address OLAP style analysis. Importing and harmonizing data for OLAP has been actively studied. Approaches range from conceptual modeling of the integration process to ontologies (Niinimaki & Niemi, 2009).

Recently, data management and analytics systems have been implemented using cloud computing resources and Big Data technologies like Hadoop (Talia, 2013). The data stored in cloud can be accessed from different locations and the owner of the data does not need to invest in hardware or software since these can be bought as a service. The cloud approach is also scalable and flexible, offering the requested capacity when it is actually needed.

Non-trivial integration processes are needed when the structure and semantics of the data are heterogeneous. In this paper, we demonstrate two different approaches to such data integration. The first method is based on using comma separated value (CSV) files of columns and rows such that the first row contains the name of the column (field). The second, more sophisticated method, is based on the Resource Description Framework (RDF) format, used for describing resources in the web.

The rest of the paper is organized as follows. First in Section Related Work, we give a review of related research. In Section Data sets, Dimension Model and Summarizability, we describe the data that we use for integration and analysis. This serves as a case study for Section System Design and Implementation, which describes the improved design of our dataspace system. In Section Distributed Analysis Using Hadoop, we describe the process of distributed analysis of RDF data using cloud computing. Finally, Section Conclusions contains a summary and conclusions.

Related Work

In related research, Shakhovska et al. (Shakhovska, Veres, Bolubash, & Bychkovska, 2015) provide a framework for big data dataspaces based on a formal model of Big Data. Klimek, Š koda, and Nečaský (2016) present an ETL framework for RDF data. Komamizu, Amagasa, and Kitagawa (2016) discuss constructing an OLAP cube by querying RDF data, but do not test the performance of the solution.

Sharma and Attar (2016) present a general framework for integrating big data with relational databases and Hadoop; Liu, Thomsen, and Pedersen (2014) demonstrate CloudETL, an ETL tool for Hadoop and Hive (a Hadoop cluster database). Queries with RDF data in Hadoop environments have been discussed by Schä tzle, Przyjaciel-Zablocki, Neu, and Lausen (2014), Kawises and Vatanawood (2016), Husain, Doshi, Khan, and

Thuraisingham (2009). Husain et al. (2009) test a limited set of queries using a 10 node cluster and RDF data with up to 1.1 billion triples. Their results are promising but the node configuration is fixed whereas our tests are performed in varying types of clusters. Similarly, Sun and Jin (2010) study bulk loading and retrieval of retrieval data in the RDF format in a Hadoop cluster but the configuration of the cluster is fixed. From a more general perspective Li, Escalona, Guo, and Offutt (2015) present a performance test framework for ETL style data integration with Hadoop, but their results are preliminary. Finally, Abello, Ferrarons, & Romero (2011) present a framework and a Google cloud implementation for constructing OLAP cubes in a cloud environment.

The World Wide Web Consortium (W3C) has published a recommendation of a Data Cube vocabulary, an RDF model for OLAP data (Cyganiak & Reynolds, 2014). The focus of this vocabulary is not data integration, instead "the Data Cube vocabulary is focused purely on the publication of multi-dimensional data on the web" (Cyganiak & Reynolds, 2014).

With respect to data correctness in the integration process, Dong and Naumann discuss resolving conflicts in data integration (Dong & Naumann, 2009). Using a survey among students, we have demonstrated that users often fail to consider if their database queries are meaningful (Thanisch, Niemi, Nummenmaa, & Niinimaki, 2019). Such mistakes are easier to detect (and avoid) if we keep track of measurement units and statistical scales (nominal, ordinal, interval, ratio) (Stevens, 1946) for the measurement values stored in the database. In general, assuring correct aggregations is known as summarizability (Lenz & Shoshani, 1997). We have proposed (Thanisch et al., 2019) a method of detecting summarizability based on statistical scales and units and additionally on the way the measured event was recorded. A direct measurement is called tally, recordings like inventory levels are called reckoning and indirect measurements are called snapshots. In a previous paper (Niinimaki & Thanisch, 2019), we presented a data management system that links structured data with Microsoft Excel's Data Models. In order to prevent errors in data integration, the system records the measurement types and they can be inspected when the data is loaded into Excel. In this paper, we enable simpler and faster connection between dataspace management and analysis. The software discussed in (Niinimaki & Thanisch, 2019) has been updated (actually re-written) to support the new functionality, too. To our knowledge, combining aggregation correctness checking with Hadoop analysis has not been studied earlier.

The main contributions of this paper can be described as (i) implementing a data integration method based on the dataspace approach. Within the process we can check that the data integration is (at least superficially) correct. The implemented system can generate a CSV file that the users can upload into their statistics software for analysis. Additionally, the data sets can also be exported in the RDF format. We further describe (ii) how the exported RDF files can be queried using the SPARQL query language (Harris, Seaborne, & Prudhommeaux, 2013) in the Hadoop framework for distributed analysis. Our test results show that in this way we can significantly reduce the required processing time.

Data Sets, the Dimensional Model and Summarizability

Data Set and Cube Description

As an example of a data collection and ETL project, let us consider the world trade of given products (like steel and aluminium) over a number of years. The analyst may be interested in imports and exports of the products between countries, the income levels of the countries, and the countries' memberships in international free trade areas like the EEA and NAFTA. We therefore need to collect data from various sources. MIT's Observatory of Economic Complexity (Simoes, 2012) contains both data and visualizations. Much of trade data uses 4-digit Standard International Trade Classification (SITC4) codes for traded goods. We have selected the SITC4 data set containing product trade by year and country. We have limited our focus to years 2000-2014. In addition to import and export figures per product, the data set contains information about export and import revealed comparative advantage. The data set contains the country names in ISO3 form. Thus, we need a few dictionary-style support files (from the MIT site) to translate SITC4 codes to product names and ISO3 codes to country names. GDP per capita figures for each country are imported from the CIA World Factbook (https://www.cia.gov/library/publications/the-world-factbook). Additionally, there is a file containing a list of countries and their participation in international free trade areas from Wikipedia. The data sets and their main characteristics are shown in Table 1.

Table 1 Data Sets

Data Set	Num lines	Num columns
year_orig_sitc_rev2.csv	1800000	5
SITC4digit_Rev2.csv	1043	2
countryname-iso3-freetrade.csv	258	5
countryname-iso3-continent.csv	258	3
cia_factbook_countries_by_gdppp.csv	231	3

A multidimensional data model contains a set of numeric measures that are the objects of analysis. Each of the numeric measures depends on a set of dimensions. This set provides the context for the measure (Chaudhuri, Dayal, & Narasayya, An overview of business intelligence technology, 2011). A multidimensional database, or a cube, is an implementation of the model. In a cube, dimensions can be hierarchical, as country - continent. The entire hierarchy is called a dimension; country and continent are called the levels of the geography dimension. Aggregation operations like sum, mean or count typically apply to dimensions: we may be interested in the sum of exports aggregated on the levels of countries or continents. Dimensions can have multiple roles. The geography dimension, for example, applies to both importers and exporters. In addition to dimension attributes, a cube contains measurements like the value of exports of a product from a country to another in given year. The unit of such measure attribute would be U.S. dollars and its statistical scale "ratio". For visualization, a cube is often represented as a "star schema" as in Fig. 1. More formally, the dimensional model can be presented as follows:

A set of dimension schemata Di ($1 \leq i \leq n$) and a measure set M are disjoint sets of attributes.

A cube schema $C = D1 \cup D2 \cup \ldots \cup Dn \cup M$, where D1 … Dn are dimension schemata, M is the set of measure attributes with their units and statistical scales, m_1, m_{1u}, m_{1s}, ..., m_p, m_{pu}, m_{ps}.

Dimension-level attribute values in each dimension must form a complete and disjoint hierarchy. This means that there is a many-to-one relationship from the lower-level attribute to the higher-level attribute. The dimension levels form a linear ordering.

A cube instance c is a relation over the OLAP cube schema C, and a relation d over D is called a dimension instance.

We assume that the measure set M is not empty and the cube has at least one dimension.

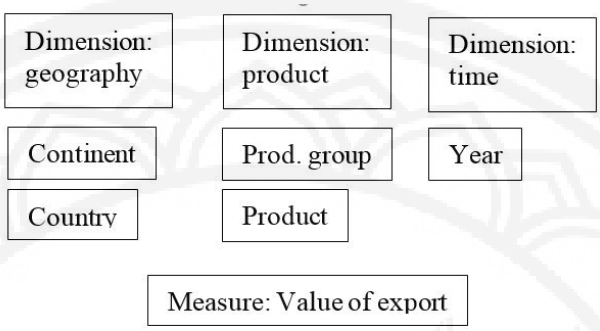

Figure 1 A Visualization of the Cube

An implementation of the OLAP data model is often based on organizing the data in a fact table surrounded by dimensional tables (Chaudhuri & Dayal, 1997). In Figure 1, is it efficient to have the fields exporter_country_id, product_id, year and value_of_export in the fact table. The dimension tables would contain the connections of each export country to their continents (country id – continent) and each product to their product groups.

As in our previous paper (Niinimaki & Thanisch, 2019), summarizability can be informally defined as "correctness of aggregate data with respect to individual observations". In practice, this is often implemented so that when a user queries a cube, the query should be checked to ensure that its result cannot violate the conditions of summarizability. We formulate the conditions as follows (Niemi, Niinimaki, Thanisch, & Nummenmaa, 2014): (i) The aggregation operation's properness for the measure, and (ii) the measure's properness for the aggregation levels in the cube's dimensions.

To validate that these conditions hold, we are specifically interested in the following:

- the statistical scales of the measure variables. The scale may determine which aggregation operation can be applied to the data
- the "eventness" type of the measure variables. The types are tally, reckoning and snapshot. Tally measures are intrinsic information about a specific event like the quantity of sale in sales data. Reckoning measures like inventory levels. Snapshot measures are indirect measurement like currency exchange rates.

It is important to include the "eventness" information about our measure variables because it can help the users avoid misleading aggregations. For example, most often we can sum up daily sales (a tally measure) to compute weekly or monthly sales. However, it is misleading to sum up inventory levels (a reckoning measure) because that would be counting the items in the inventory multiple times (though calculating the average of inventory levels over time is normally acceptable).

Instead of checking the summarizability when evaluating a query, our design (described in Section 3 below) aims at ensuring summarizability when the cube is constructed from data sets. This is not a definite guarantee

that all the user queries with the constructed cube will result in "correct" answers. This is because in rare cases the generated cube can contain multiple dimension-measure pairs (DiMi) such that D_1M_1 and D_2M_2 would ensure summarizability, but the user manages to formulate a query that returns an answer based on D_1M_2. However, in the examples of this paper, such a situation does not occur.

RDF Description

The amount of available data for business intelligence applications is potentially very large, and the data may have been stored in a geographically distributed manner, since it is profitable to run data processing in a distributed way on demand. We have successfully applied grid/cloud technologies for this purpose earlier (Niemi, Toivonen, Niinimaki, & Nummenmaa, 2007). Here, we define the data sources using an ontology-based approach to map to original data sources to an OLAP ontology describing a star schema. This approach would also enable us to take advantage of common big data processing platforms, which usually offer a higher abstraction level, an efficient distributed storage solution and greater computational power.

The applied methodology consists of the following steps:

1. Defining ontology descriptions for (big) data sources. These descriptions contain definitions of data types with their summarizability information.

2. Retrieving the data based on the ontology descriptions. This step can be performed in a sequential and distributed fashion or by applying distributed query processing using a big data platform.

3. Collecting and aggregating the results of sub-queries, if the previous step was distributed.

4. Creating the OLAP cube schema for the retrieved data and uploading the data into user's analysis server (e.g. an OLAP tool or a statistical software package).

In the first step, the available data is mapped into an OLAP ontology using RDF (Niemi, Toivonen, Niinimaki, & Nummenmaa, 2007). This means identifying data sources, and their measure and dimension attributes with their hierarchies. The ontology should also contain information (eventness, measure units, statistical scales) needed to guarantee the correctness of aggregations. In Steps 2 and 3, the data needed to answer the user's query at hand is retrieved. Finally, in Step 4, the collected OLAP data is transformed into a suitable format for uploading onto the analysis tool. This step includes mapping the data types in the ontology into the data types used in the analysis system.

System Design and Implementation

The purpose of the software is to enable its user to manage a collection of CSV files, and metadata related to them. The software lets the user upload data files and after each upload it presents the user with a form. For each measure in the data file, the user will select the measure's statistical scale and eventness using the form. For each dimension, the form has a field for its description.

An interface with the meta data form is shown in Figure 2. A list of uploaded data sets is shown in Figure 3.

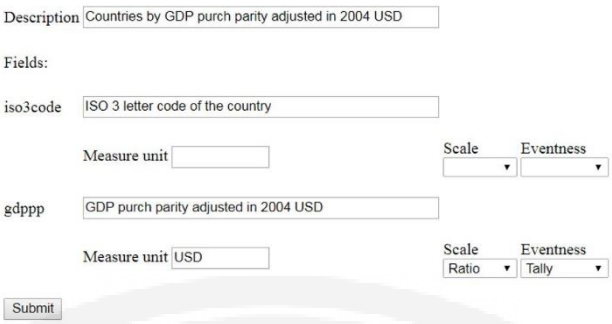

Figure 2 Meta Data Editor

Figure 3 Uploaded Data Sets

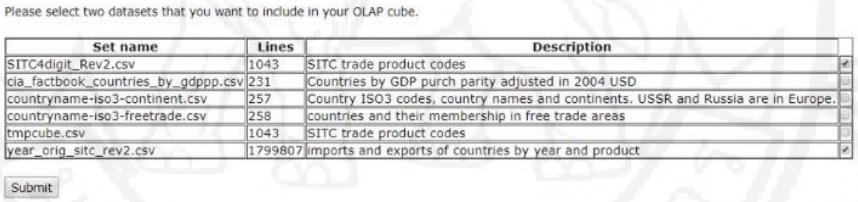

Figure 4 Cube Building: Selecting the Data Sets for the Original Cube

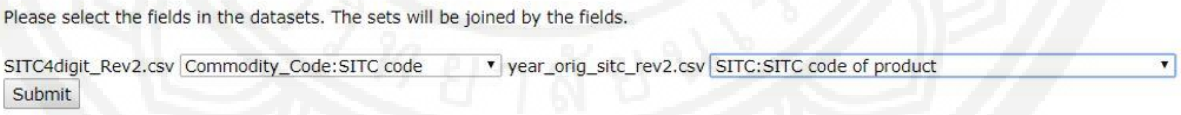

Figure 5 Cube Building: Selecting Fields for the Original Cube

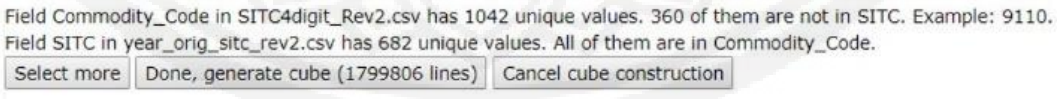

Figure 6 Cube Building: Analysis of the Selected Fields

Other than disk space, there is no limitation to the number of files that can be managed by the platform. For clarity and brevity, however, we assume that the files that the user has uploaded are those listed in Table 1. Next, we shall demonstrate the process of selecting data sets, combining them by measure and exporting the resulting cube.

The users start the cube building by clicking on the "Cube" entry in the menu of Figure 3. The cube building process is iterative in such a way that the users first select a pair of data sets and indicate the columns by which the sets will be joined. The result of this operation is an initial cube. The process continues so that the user selects more data sets that are joined with the initial cube. At each step, the system determines whether the integration is going to be successful. Figures 4, 5 and 6 illustrate the process.

In Figure 4, the user selects datasets "year_ orig_sitc_rev2" and "SITC4digit_rev2" in order to map SITC4 trade codes to product names. The process continues in Figure 5 where the user selects the corresponding columns. After this selection, the software analyses similarities of the values of the fields as in Figure 6.

As in our previous paper, we have tested the upload and download time of the dataspace software. Additionally, we can now measure the performance of cube construction with the R statistics software. The platform used was an Intel Core i5 dual-core 2.4 GHz CPU computer with 8 gigabytes of memory and a 500 gigabyte 7200 rpm hard disk. With this hardware, the dataspace application's upload speed is 5MB/s and the download speed 38 MB/s. Reading and analyzing the content of the CSV files in order to guarantee that the data can be merged (see Figure 6) is reasonably fast: our test computer was able to read the data from the files, find the unique field values and compare them at the speed of 330 000 lines per second. Writing the cube data into a file is needed to provide the users with a CSV dataset that can be loaded into a statistics package. The writing speed was about 150 000 lines per second.

With large files, memory management of the R software becomes problematic. We tested reading speed with a file with more than 100 million observations representing trade between countries for the years 2000-2014. Simply loading this data into R took more than 2 hours. In Section 5, we discuss how to improve OLAP-style analysis for large data sets using the RDF format.

Distributed Analysis Using Hadoop

There are several studies on how to process RDF (SPARQL) queries in Hadoop (Schätzle et al., 2014; Husain et al., 2009; Kawises & Vatanawood, 2016). However, our OLAP approach makes it possible to use a more straightforward and efficient query method. We divide the query processing into two steps: (i) Data selection and preparation for roll-up operations, (ii) Aggregation computing. In the first step, we find the required dimension attributes for each fact table row. An example of RDF data containing a row of a dimensional table (country – continent) and a few lines of fact data is shown below.

```
<rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
xmlns:a='http://www.olaprdf.org/'>
<Description about="bPHbx6maMqCVaIQtHR8q3tQ0" a:CountryName="Aruba" a:ISO3digit="ABW"
a:Continent="South America"/>
..
<Description about="bBWmFivgAFztQeaAFknjgKw0" a:YEAR="2000" a:ORIGIN="AGO" a:SITC="5722"
a:EXPORT_VAL="362.0" a:IMPORT_VAL="147648.0" a:EXPORT_RCA="0.000662487"
a:IMPORT_RCA="1.12053"/>
<Description about="bBWmFivgAFztQeaAFknjgKw1" a:YEAR="2000" a:ORIGIN="AGO" a:SITC="6251"
a:EXPORT_VAL="0.0" a:IMPORT_VAL="2148941.0" a:EXPORT_RCA="0.0" a:IMPORT_RCA="0.637642"/>
</rdf:RDF>
```

As an example, let us assume that the analyst wants to aggregate data on the 'continent' level (see Figure 1). In step (i), we find the corresponding continent for each country using the data in the geography dimension. The dimension data is stored in HDFS (Hadoop Distributed File System) and each mapper process accesses it directly without using the MapReduce protocol. This does not cause much overhead since the dimension data is usually small compared to the fact table data. The fact table data is then distributed to the mappers (step (ii)) by MapReduce. The data is stored in such a way that each row corresponds to an OLAP fact row containing the dimension keys and the measure values.

Since each row of the data is independent, sending rows to different mappers does not cause problems. Compared with a single process approach, Hadoop does not bring a large overhead and enables us to process large datasets using nodes that do not have large amounts of RAM. This can be demonstrated by the following example.

The SPARQL query below aggregates export values by continents. The "a" in the query is an abbreviation of the namespace in which attributes like EXPORT_VAL have been defined. We apply it to an RDF file that combines country/continent and trade data, containing 1.8 million lines (see Table 1). Using Python's RDFlib query tool, the query runs for 53 minutes (actually for this we needed to allocate more memory, 8 GB RAM is not sufficient for this amount of data).

```
SELECT
    ?continent (SUM(xsd:decimal(?export)) AS ?exp)
WHERE {
    ?country a:Continent ?continent .
    ?country a:ISO3digit ?iso .
    ?e_id a:ORIGIN ?iso .
    ?e_id a:EXPORT_VAL ?export
}
GROUP BY ?continent
```

Using Hadoop MapReduce, we avoid the problems of running out of memory, and improve the query performance. In our design, each reducer processes, i.e. aggregates, independent parts of the data, so this can be done in parallel. In this example, data on different continents can be processed by different reducers. Finally, each reducer writes its output and the final output. Thus, the requested OLAP cube is formed by combining the reducers' outputs. Our MapReduce processing software was implemented using Python and it consists of the following parts:

Mapper:

1. Initialize the RDF graph.
2. Read dimension data from HDFS and add it to the graph.
3. Read MapReduce input and add it to the graph.
4. Process a SPARQL query for rolling up the dimension keys and selecting the desired fact rows.
5. Serialize the query result and select a key. The key should be the most selective rolled-up dimension key to allow maximum parallelism in the reduce phase.

Reducer:

1. Initialize the RDF graph.
2. Read MapReduce input from mappers and add it to the graph.
3. Process a SPARQL query for aggregating the data.
4. Write the output.

We tested the method using a Hadoop cluster in an Amazon Web Service (AWS) cloud environment. The MapReducer task aggregates export values of all countries grouped by continents, an in the example above.

The mapper query was:

```
CONSTRUCT {
    ?continent a:EXPORT_VAL ?export}
WHERE {
    ?country a:Continent ?continent .
    ?country a:ISO3digit ?iso .
    ?e_id a:ORIGIN ?iso .
    ?e_id a:EXPORT_VAL ?export
    }
```

And the reducer query:

```
CONSTRUCT {
    ?continent a:EXPORT_VAL ?exp}
WHERE {
    SELECT
        ?continent (SUM(xsd:decimal(?export)) AS ?exp)
    WHERE {
        ?continent a:EXPORT_VAL ?export .}
    GROUP BY ?continent
      }
```

We run the queries in a Hadoop MapReduce cluster using the following Python codes for the mapper and reducer:

```
# The mapper code:

def main(separator='\t'):
  # Creates a graph object
  g = rdflib.Graph()
  # Reads the data from STDIN
  lines=sys.stdin.readlines()
  # Combines the lines
  allLines=''
  header="<?xml version='1.0'?> \n <rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-
  syntax-ns#' xmlns:a='http://www.olaprdf.org/' xml:base='http://www.olaprdf.org/'>\n"
  allLines += header
  allLines +=''.join(lines)
  footer = "</rdf:RDF>\n"
  allLines += footer
  g.parse(data=allLines,format='xml')

  # Reads the dimensions from HDFS to a local file
  tempfile="/tmp/temp"+str(random.randint(1,1000000))+".rdf"
```

```
  for rdffile in
  ['/user/hadoop/trade/dimensions/countries.rdf','/user/hadoop/trade/dimensions/country-
  continent.rdf', '/user/hadoop/trade/dimensions/commodity.rdf']:
       cat = subprocess.call(["/usr/bin/hadoop", "fs", "-copyToLocal", rdffile, tempfile])
       # Adds a dimension file to the graph
       g.load(tempfile)
       os.remove(tempfile)

  # Processes the mapper SPARQL query given in the previous example.
  q = g.query('CONSTRUCT …'}', initNs = { 'a' :'http://www.olaprdf.org/'})
  # Serializes the result
  qs = q.serialize(destination=None, format='nt').decode()
  # Ouputs the result to SDOUT
  print(qs)

# The reducer code:

def main(separator='\t'):
  #Creates a graph
  g = rdflib.Graph()
  # Reads the data from STDIN
  lines=sys.stdin.readlines()
  # Combines the lines and adds them into the graph
  allLines=''.join(lines)
  g.parse(data=allLines,format='nt')

  # Processes the reducer SPARQL query given in the previous example.
  q = g.query('CONSTRUCT …', initNs = { 'a' :'http://www.olaprdf.org/'})

  # Outputs the result to SDOUT
  for row in q:
     print (row)
```

The Python MapReduce code (above) can be run in the cluster using the following command:

```
# Removes possible existing results
hadoop fs -rm -r /user/hadoop/trade/trade-output

hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \
  -numReduceTasks 1 \
  -file ./mapper_rdf.py -mapper ./mapper_rdf.py \
  -file ./reducer_rdf.py -reducer ./reducer_rdf.py \
  -input /user/hadoop/trade/data/* -output /user/hadoop/trade/trade-output
```

The results are shown in Table 2. As we can see, adding more computing nodes always decreases the computing time. However, the performance improvement is not linear and heavily depends on the node types. In practical solutions, the gained performance improvements can be significant, since using large enough clusters can drop the processing time to less than a tenth compared to the single node case.

Table 2 Processing

Type of nodes	Num slave nodes	Processing time	% faster than 1 large node	Times faster than 1 large node
m4.large	1	64m 29.9s		
m4.large	2	62m 48.8s	3%	1.03
m4.large	4	29m 17.3s	55%	2.2
m4.large	8	14m 51.4s	77%	4.34
m4.large	19	9m 25.1s	85%	6.85
				Times faster than 1 xlarge node
m4.xlarge	1	33m 45.7s	48%	
m4.xlarge	2	19m 56.7s	69%	1.69
m4.xlarge	5	10m 31.7s	84%	3.21
m4.xlarge	8	9m 9.5s	86%	3.69

Conclusions

This paper presents a dataspace management software package that supports the Export-Transform-Load process in such a way that the resulting data sets can be used for analysis in the RDF format. Our design helps data analysts discover and prevent problems with data integration. For this purpose, we store meta data together with the data sets. The meta data includes scale, "eventness" and unit for measures. The dataspace application supports an OLAP-like cube construction into a file that can be used by statistics software. Our test results indicate that using a distributed MapReduce framework in a cloud computing cluster can decrease the processing time by 85% or more depending on the number and type of the nodes in the cluster.

Although the results have been encouraging, our implementation was based on RDF files in a shared Hadoop filesystem. In order to support even larger data sets, in our future research we shall look into performance of RDF databases in Hadoop analysis. Since database systems are now increasingly used with Internet of Things technologies (Rautmare & Bhalerao, 2016), their energy efficiency is another interesting field of study.

The software is available at https://sourceforge.net/ projects/simple-dataspace-management. The software has been under active development for two years. Data integration using an earlier version of the software was discussed in (Niinimaki & Thanisch, 2019), but the summarizability analysis combined with RDF processing (described in this paper) is available in the current revision of the software. The software features field compatibility checking, organizing data into collections, and search functionality. Our next development tasks are to enable Hadoop processing directly from the Web interface and to support the RDF Data Cube vocabulary in the RDF conversion of the data.

Acknowledgments

This research has been partially supported by Hasler Foundation (project number 18039) and a "Big Data Research Jan 2020" grant from Webster University.

References

Abelló, A., Ferrarons, J., & Romero, O. (2011). *Building cubes with MapReduce.* Retrieved from https://www.researchgate.net/publication/220933887_Building_Cubes_with_MapReduce

Antonio, M., Salles, V., & Dittrich, J.-P. (2006). *iMeMex: A Platform for Personal Dataspace Management.* Retrieved from https://www.researchgate.net/publication/244431278_iMeMex_A_Platform_for_Personal_Dataspace_Management

Chaudhuri, S., & Dayal, U. (1997). An overview of data warehousing and OLAP technology. *ACM SIGMOD Record, 26*(1), 65-74.

Chaudhuri, S., Dayal, U., & Narasayya, V. (2011). An overview of business intelligence technology. *Communications of The ACM, 54*(8), 88-98.

Cyganiak, R., & Reynolds, D. (2014). *The RDF Data Cube Vocabulary.* Retrieved from https://www.w3.org/TR/2012/WD-vocab-data-cube-20120405/

Dong, X., & Naumann, F. (2009). Data fusion: resolving data conflicts for integration. *Very large data bases, 2*(2), 1654-1655.

Dong, X., & Srivastava, D. (2013). Big data integration. In *2013 IEEE 29th International Conference on Data Engineering (ICDE), 8-12 April 2013* (pp. 1245-1248). Brisbane, QLD, Australia: Institute of Electrical and Electronics Engineers.

Franklin, M., Halevy, A., & Maier, D. (2005). From databases to dataspaces: a new abstraction for information management. *international conference on management of data, 34*(4), 27-33.

Halevy, A., Franklin, M., & Maier, D. (2006). *Principles of dataspace systems.* Retrieved from https://dl.acm.org/doi/10.1145/1142351.1142352

Harris, S., Seaborne, A., & Prudhommeaux, E. (2013). *SPARQL 1.1 query language.* Retrieved from https://www.w3.org/TR/sparql11-query/.

Husain, M., Doshi, P., Khan, L., & Thuraisingham, B. (2009). *Storage and Retrieval of Large RDF Graph Using Hadoop and MapReduce.* Retrieved from http://cs.utdallas.edu/semanticweb/Hadoop-RDF/Paper-SR-Large-RDF-Graphs.pdf

Kawises, J., & Vatanawood, W. (2016). A development of RDF data transfer and query on Hadoop Framework. *International Journal of Pharmacy and Technology, 8*(4), 19492-19498.

Klimek, J., Škoda, P., & Nečaský, M. (2016). LinkedPipes ETL: Evolved Linked Data Preparation. In *International Semantic Web Conference, 29 May 2016 - 2 June 2016* (pp. 95-100). Greece: Haraklion.

Komamizu, T., Amagasa, T., & Kitagawa, H. (2016). H-SPOOL: A SPARQL-based ETL framework for OLAP over linked data with dimension hierarchy extraction. *International Journal of Web Information Systems, 12*(3), 359-378.

Lenz, H.-J., & Shoshani, A. (1997). *Summarizability in OLAP and statistical data bases.* Olympia, WA: IEEE.

Li, N.-Y., Escalona, A., Guo, Y., & Offutt, A. (2015). A Scalable Big Data Test Framework. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST), 13-17 April 2015* (pp. 1-2). Graz, Austria: Institute of Electrical and Electronics Engineers.

Liu, X., Thomsen, C., & Pedersen, T. (2014). *CloudETL: scalable dimensional ETL for Hive.* Retrieved from https://www.researchgate.net/publication/266660677_CloudETL_scalable_dimensional_ETL_for_hive

Mauro, A., Greco, M., & Grimaldi, M. (2016). A formal definition of Big Data based on its essential features. *Library Review, 65*(3), 122-135.

Niemi, T., Niinimaki, M., Thanisch, P., & Nummenmaa, J. (2014). Detecting summarizability in OLAP. *data and knowledge engineering, 89*, 1-20.

Niemi, T., Toivonen, S., Niinimaki, M., & Nummenmaa, J. (2007). Ontologies with Semantic Web/Grid in Data Integration for OLAP. *International Journal on Semantic Web and Information Systems, 3*(4), 25-49.

Niinimaki, M., & Niemi, T. (2009). An ETL process for OLAP using RDF/OWL ontologies. *Journal on Data Semantics, 13*, 97-119.

Niinimaki, M., & Thanisch, P. (2019). Dataspace Management for Large Data Sets. *Innovative Computing Trends and Applications, 1*, 13-21.

Rahm, E., & Do, H. (2000). Data Cleaning: Problems and Current Approaches. *IEEE Data(base) Engineering Bulletin, 23*, 3-13.

Rautmare, S., & Bhalerao, D. (2016). MySQL and NoSQL database comparison for IoT application. In *2016 IEEE International Conference on Advances in Computer Applications (ICACA), 24-24 October 2016* (pp. 235-238). Coimbatore, India: Institute of Electrical and Electronics Engineers.

Schä tzle, A., Przyjaciel-Zablocki, M., Neu, A., & Lausen, G. (2014). *Sempala: Interactive SPARQL Query Processing on Hadoop.* Retrieved from https://www.researchgate.net/publication/270901879_Sempala_Interactive_SPARQL_Query_Processing_on_Hadoop

Shakhovska, N., Veres, O., Bolubash, Y., & Bychkovska, L. (2015). Big data information technology and data space architectur. *Sensors and Transducers, 195*(12), 69.

Sharma, K., & Attar, V. (2016). Generalized Big Data Test Framework for ETL migration. In *2016 International Conference on Computing, Analytics and Security Trends (CAST), 19-21 December 2016* (pp. 528-532). Pune, India: Institute of Electrical and Electronics Engineers.

Simoes, A. (2012). *The observatory : designing data-driven decision making tools.* Retrieved from https://www.media.mit.edu/publications/the-observatory-designing-data__driven-decision-making-tools/

Stevens, S. S. (1946). On the Theory of Scales of Measurement. *Science, 103*(2684), 677-680.

Sun, J., & Jin, Q. (2010). Scalable RDF store based on HBase and MapReduce. In *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE), 20-22 August 2010* (pp. 1). Chengdu, China: Institute of Electrical and Electronics Engineers.

Talia, D. (2013). Clouds for Scalable Big Data Analytics. *IEEE Computer, 46*(5), 98-101.

Thanisch, P., Niemi, T., Nummenmaa, J., & Niinimaki, M. (2019). Detecting measurement issues in SQL arithmetic expressions and aggregations. *Data and knowledge engineering, 122*, 116-129.