IIETA International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Ingénierie des Systèmes d'Information
Vol. 28, No. 3, June, 2023, pp. 557-565
Journal homepage: http://iieta.org/journals/isi

Os-ETL: A High-Efficiency, Open-Scala Solution for Integrating Heterogeneous Data in Large-Scale Data Warehousing

El Yazid Gueddoudj[1,2*], Azeddine Chikh[1,2], Abdelouahab Attia[3,4]

[1] Computer Science Department, Faculty of Science, University of Tlemcen, Tlemcen 13000, Algeria
[2] LRIT Laboratory, University of Tlemcen, Tlemcen 13000, Algeria
[3] Computer Science Department, University Mohamed El Bachir El Ibrahimi of Bordj BouArreridj, Bordj Bou Arreridj 34000, Algeria
[4] LMSE Laboratory, University Mohamed El Bachir El Ibrahimi of Bordj BouArreridj, Bordj Bou Arreridj, Algeria

Corresponding Author Email: yazid.gueddoudj@univ-tlemcen.dz

https://doi.org/10.18280/isi.280303

Received: 15 February 2023
Accepted: 21 May 2023

Keywords:
ETL, spark, big data, RDF, partitioning, data warehouse, polystore, scalability

ABSTRACT

The surge in data volume necessitates the integration of Resource Description Framework (RDF) data within corporate environments. While Extract, Transform, Load (ETL) processes exhibit proficiency with conventional data sources, their scalability diminishes when applied to large and highly varied data sources, inclusive of RDF data. The latter constitutes a wealth of knowledge that, when harnessed via data warehouse technology, can augment corporate value in a fiercely competitive milieu. The advent of platforms like polystore offers opportunities for advanced hardware deployment. ETL processes necessitate two crucial phases: Partitioning and data allocation. Concurrently, the scientific community is spurred to innovate ETL processes that support real-time analytics. This study proposes a novel architecture for ETL processes, named Open-Scala-ETL (Os-ETL). Equipped with a method for deploying a data warehouse based on a polystore, Os-ETL enables real-time analysis. The primary objective of the Os-ETL solution is to resolve the complexities in deploying a graph structure data warehouse on a polystore—a process that involves partitioning and data allocation. Os-ETL is a distributed solution that supports both batch and streaming processing using the Spark framework. Scala scripts are executed within this framework to partition RDF graphs and distribute the resultant fragments across various sites. The implementation of Os-ETL is based on Apache Spark, with ETL deployment on a Spark SQL polystore. This solution empowers companies with data warehouse technology to improve performance, scalability, and latency between a data warehouse and its data sources. The approach has been assessed and validated using large-scale, heterogeneous data, encompassing the LUBM benchmark, CSV files, an Oracle database, and a Neo4j graph database. The results corroborate its superior performance in terms of scalability and optimization.

1. INTRODUCTION

Nowadays, organizations are increasingly generating large amounts of data in a wide variety of high-speed formats (Figure 1). We are in the age of Big Data [1]. Researchers define Big Data by the following four Vs.: Volume, Variety, Velocity and Veracity. These four dimensions characterize and distinguish big data from ordinary data. Also, the rapid growth of a graph database (RDF data) is an excellent opportunity to enrich traditional data warehouses with a new V dimension of big data: Value. Thus, allow companies to exploit these rich data to enhance their added value in a highly competitive world. A few studies have mostly focused on the integration of RDF data into a data warehouse and on the ETL deployment process based on a polystore system. The first work that used a real polystore, but with relational data, was proposed by Meehan et al. [2]. Du et al. [3] proved that a dynamic workload approach is necessary for data placement in a polystore system in order to support ingestion of low-latency data.

Berkani et al. [4], chose to deploy the data warehouse in vertical representation with Oracle DBMS which, offers a storage model to represent instances and graphs, using Oracle RDF Semantic Graph. They planned a simulation to deploy a data warehouse on a polystore.

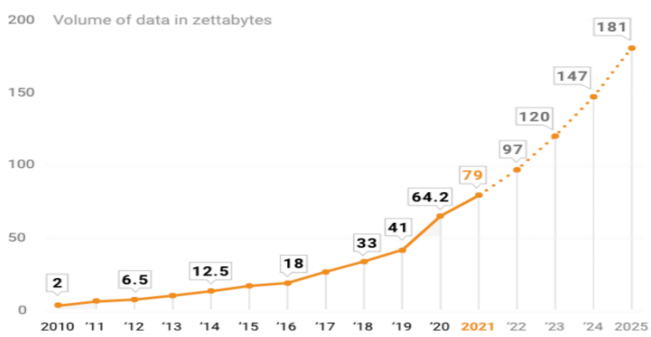

Figure 1. Explosion of data
(Source: statista.com)

Some works achieve the right degree of parallelism of ETL processes, by providing the Map and Reduce functions, known

as a MapReduce framework (i.e., partition settings, number of nodes) [5]. However, the later has some limitations (Figure 2). Indeed, it only accepts one input data stream at a time in a key/value format, also it doesn't support a real-time processing. In addition, the developer has to write custom code for the Map and Reduce functions, which is difficult to maintain and reprocess.

Furthermore, companies nowadays need to make real-time decisions based on large amounts of data, given from graph structured data sources. Thus, we argue that for modern data warehouse (Big Data warehouse) applications, in which latency and scalability are of great importance, the ETL process should be optimized using new technologies such as Spark framework. We claim that this requires a new architecture including a Spark module for building new data warehouse applications. Also, the goal of this study is to resolve the challenging process of deploying a graph structure data warehouse on a polystore, which involves two phases: data allocation and partitioning.

Thus, in this paper, we propose a new solution to solve the problem of deploying a graph structure data warehouse on a polystore, which is a difficult task, with two phases namely the partitioning and the allocation of the data. The proposed architecture for ETL named Open-Scala-ETL, which consists of three components: (i) a collector of data updates (CDC technique); (ii) a Spark ETL engine with two modules for fragmentation and assignment; and (iii) a deployment polystore. Further, to overcome limitations of MapReduce framework, we use the in-memory parallel computing (Spark framework), which uses a Direct Acyclic Graph (DAG) model, for computing components multiple times and supports in-memory data sharing between multiple tasks. Also, the use of new parallelization technologies is the key for achieving better scalability and ETL performance.

The main motivation of the proposed architecture for ETL processes is the need of improving the efficiency and performance of these processes which deal with heterogeneous and voluminous data compared to other existing ETL based optimization techniques. Longo et al. [6] present a novel approach based on stream control in ETL. Masouleh et al. [7] introduce a control-flow-based approach to ETL process modeling, and use the combination of parallelization and shared cache memory to optimize the ETL performance of the data warehouse.

Thus, the given solution involves various data sources (relational, graph and CSV, etc.) and takes advantage of the computing power and distributed storage, offered by the Spark framework to optimize ETL processes. Both academics and industry have adopted Apache Spark as a fast and scalable framework [8]. Our new framework allows the development of multidimensional data warehouses capable of managing large masses of data. We provide deployment on polystore. Therefore, our approach takes advantage of different storage systems (polystore) to increase the execution performance of the analysis algorithms.

The paper is organized as follows. After this introduction; section 2 presents the related works which summarize the main existing studies underlining ETL processes. Section 3 details the proposed framework. Section 4 introduces the background knowledge of the proposed system and presents a case study. Section 5 presents the experimental results. Finally, section 6 concludes the paper.

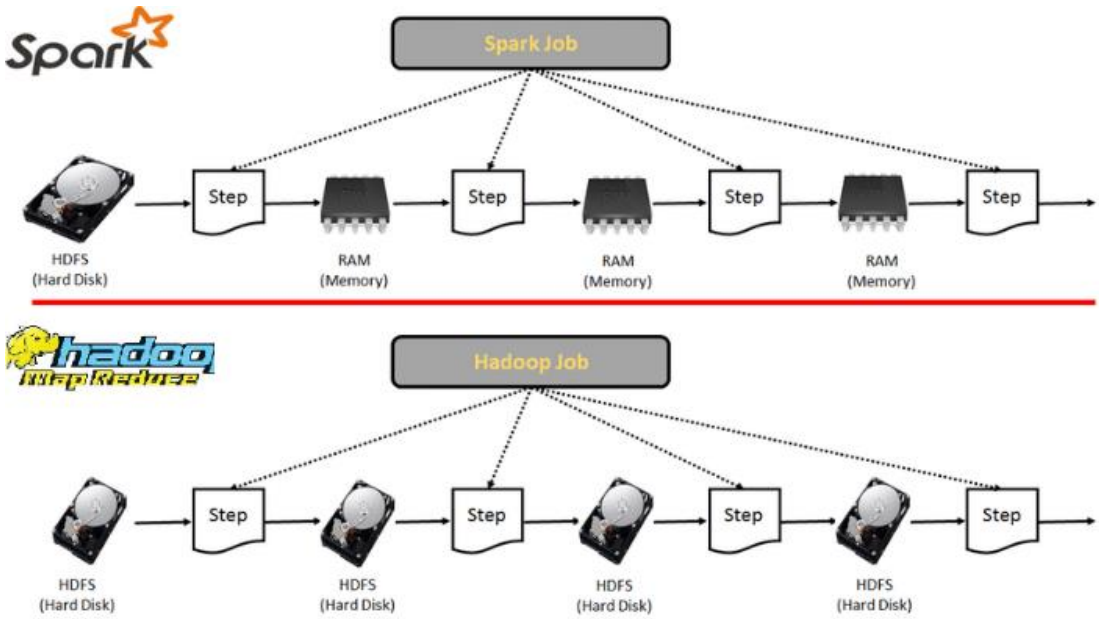

Figure 2. The MapReduce vs. Spark paradigm [9]

2. RELATED WORK

This section describes the state of the art of the main works of the ETL process dealing with the real-time, the deployment of data warehouse on polystore as well as the problem of variety in the context of integrating of Big Data Belayadi et al. [10], Alsanad et al. [11], Chen et al. [12]. Vassiliadis and Simitsis [13] present a study related to research and industry for real-time ETL. Thus, an EAI (Enterprise Application Integration) solution has been created, with a micro-batch ETL which provides a rapid transformation. However, this approach presents the challenge of consistency, which indicates the time interval between the data sources and the target data warehouse before answering queries. A semi-streaming join algorithm (SSJA) has been proposed [14]. The (SSJA) algorithm supports highspeed streams with disk and

memory access management. Muddasir and Raghuveer [15], discuss the three available techniques to achieve realtime ETL, including metadata management, change data capture (CDC) and distributed processing of Big Data. Meehanet al. [2], propose a novel generic streaming ETL architecture for new real-time data warehouse applications. The architecture contains four components: a data collector; the streaming ETL engine; the OLAP (online analytical processing) backend; and a data Migrator. But the proposed approach is not benchmarked against the state of the art to assess its performance against other ETLs. Biswas and Mondal [16], plane a cloud-based ETL solution with a new Spark-based ETL framework. The proposed framework manages the real-time data flow on the cloud platform. Berkani et al. [17], dynamically combine the ETL process and the selection of materialized views, within the framework of a near real-time data warehouse design. They consider semantic data sources, with particular emphasis on the ETL and physical design phases. Boury-Brisset [18], propose the design and implementation of a framework based on scalable multi-intelligence data integration services to facilitate the integration of heterogeneous unstructured and structured data. Pareek et al. [19] presented a new distributed architecture of ETL Striim developed to collect datasets from different sources. The ETL engine runs on a scalable and fault-tolerant cluster of compute nodes. It extracts real-time data from sources, transforms it and produces result events that are loaded/broadcasted into targets. The authors demonstrate the efficiency of Striim's ETL engine against a popular KSQL open-source streaming ETL engine built on top of Apache Kafka. Bansal [20], propose a semantic ETL framework which uses semantic technologies in order to integrate heterogeneous data. It provides a foundation for integrating and understanding knowledge from multiple sources.

Other works based on the real-time ETL, consider the optimal structures related to the traditional data warehouse. Ali and Wrembel [21], present a study of existing works on real-time ETL based optimization of ETL processes within the framework of traditional data warehouses. Besides, a theoretical framework for ETL optimization has been introduced. To improve the efficiency of ETL processes, Thomsen and Pedersen [22] have created ETL processes using Python code. Developers can treat data efficiently by using built-in features and using data access for fact and dimension tables. The proposed solution does not provide any cost model to estimate a performance benefit. Liu and Iftikhar [23], proposed an optimization framework for minimizing the time and resources required for ETL data flows. The partitioning and parallelization of ETL processes have been employed. Thus, the ETL process is partitioned into linear sub-processes, and then data parallelization is applied to each of the sub-processes. Finally, all processes are multithreaded. The framework is implemented in the open source ETL tool, Talend Open Studio for data integration. The large amount of data makes ETL extremely expensive. Liu et al. [24], presented a framework named ETLMR, which employed MapReduce to achieve scalability which easily creates dimensional ETL processes based on MapReduce to load data into the data warehouse. The user of ETLMR framework can implement parallel ETL programs with a few lines of code declaring target tables and transformation functions.

In the context of change data capture (CDC), research work based on the log-based method is introduced [25]. A trigger-based approach to capture changed data from different data sources is mentioned by Valêncio [26]. One of the most efficient ways to extract data from a database with minimal overhead is to use, Change Data Capture (CDC) [13].

A growing need for Big Data analytics requires new architectures to store data, such as a data lake or a polystore [21]. A polystore named Bigdawg has been presented by Duggan et al. [27]. While, Gurajada [28], divided the RDF graph into multiple fragments and distribute each to a different site in a distributed system. Also, Huang et al. [29], planed a graph partitioning approach on a distributed framework, where the triples are distributed across different machines using the graph partitioner [30]. Al-Ghezi and Wiese [31], considered the adaptation of a partitioning and replication layer with both the workload and the availability of storage space in a distributed RDF triple store. Without prior assumption on the query workload, the proposed system starts by performing a static partitioning of the graphs using METIS [30].

Finally, most of the aforementioned works deal only with traditional types of sources with the deployment of the data warehouse on a relational system. Furthermore, no work attempts to propose a central and uniform model to cover a wide variety of sources and complicated semantics in order to provide a framework for the optimization and scalability of ETL processes. In this work, we address this problem by proposing a pivot model that allows for the alignment of all types of sources in a common RDF model. Moreover, the proposed solution allows supporting a data warehouse, deployed on a polystore with easy scaling provided by the use of the Spark framework.

3. THE PROPOSED ARCHITECTURE

Figure 3 illustrates the architecture of the proposed approach named Open Scala ETL process (Os-ETL). It consists of six steps: (i) extract and transfer stream data; (ii) alignment of all data sources to the RDF data model; (iii) partitioning of data sources (iv) transform RDF data sources into GraphX; (v) the appropriate transformations; and (vi) allocation and distribution of the obtained RDF fragments on the Spark SQL polystore. These steps are described in details as follows:

Step 1: extract and transfer stream data, which consist an update from the data sources to the Hadoop Distributed File System (HDFS), using the change data capture technique that identifies changes to data sources via triggers implemented in each source participating in the ETL.

Step2: alignment of all data sources to the RDF data model. This step involves a manual process for analyzing the datasets. We create a pivot model to reduce the number of mappings for the different formats of the data sources involved in the ETL.

Step 3: the partitioning of data sources, in this case the adapted algorithms named path partitioning (Figure 4) introduced by Wu et al. [32], has been employed. In our data warehouse scenario, setting up a polystore entails solving graph data partitioning and allocation issues.

Step 4: transform RDF data sources into GraphX [33]. GraphX is the Spark API for graphs and graph-parallel computing.

Step 5: perform the appropriate transformations using Scala scripts. In this step the traditional ETL operators and ETL operators for managing graph representations have been used.

Step 6: allocation and distribution of the obtained RDF fragments on the Spark SQL polystore. In our solution for integrating different types of data, we use the solution of

several DBMS together.

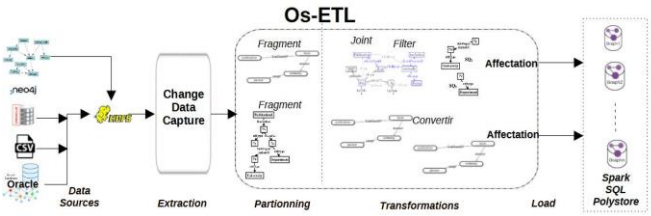

Figure 3. General architecture of the Os-ETL

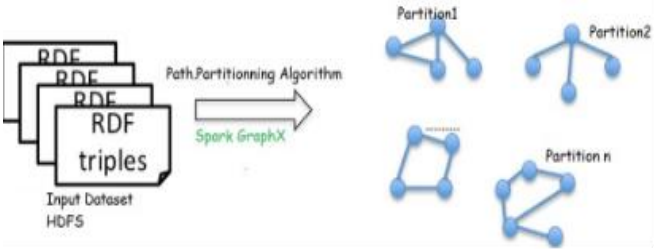

Figure 4. Path partitioning

The planed Os-ETL system involving heterogeneous data sources to ensure scalability and reduce latency between a data warehouse and its sources. In addition, for step (ii) alignment of all data sources to the RDF data model needs to modify the pivot model [34], in order to unify all data model formalisms. In this case, the alignment to the RDF data model has been employed. Thus, the problem of integrating heterogeneous data with heterogeneous data models in data warehouse applications is solved. The Os-ETL solution is implemented in Scala scripts, based on the Spark execution model shown in Figure 5.

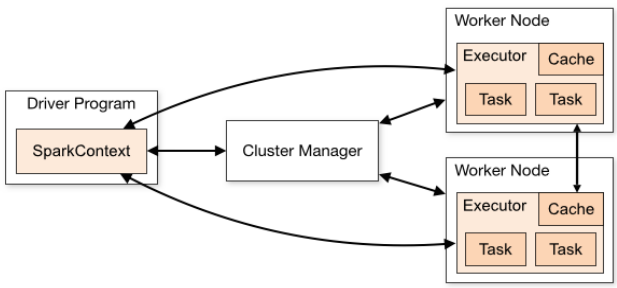

Figure 5. Spark execution model
Note: (https://spark.apache.org/docs/latest/cluster-overview.html)

The proposed algorithm Os-ETL

In the proposed architecture, we consider the schema of the data warehouse as predefined; followed by the data sources which are annotated in the tool protégé [35] according to the needs expressed in the application.

Also in Step 1, the Last updated timestamp (variable used by the technique change data capture (CDC)), which detects and collects data changes of interest requires techniques known as CDC that continuously monitor operational data sources [36]. The output of this step is represented by a delta (which represents all changes to be made to the data sources). The eighty (18) generics conceptual ETL operators [37, 38], have been used. Algorithm 1 describes the steps of the proposed Os-ETL system.

Algorithm 1 Os-ETL

Input: Relational database, CSV files, Flat files, Graph databases (NT files or OWL files): all of the data source models are aligned to RDF data model, the schema of the data warehouse (DW) and the eighty (18) conceptual ETL operators.
Output: Big Data Darehouse deployed over Spark SQL polystore.
Begin

1. - Create the control module CDC control module
2. - **if** DW = empty **then**
   Repeat for each RDF data source
   Load changes into Temp
   Filler source by last updated timestamp
   Result (delta) to ETL engine (create and load a view)
   end if
3. - Built the fragmentation by using Algorithm:P.P (Graph RDF, NBpartitions)
   begin
   Load the N-Triple dataset and apply the path partition algorithm
   Specifies end-to-end paths
   Generate path groups for each starting vertex
   Merges a vertex based on the number of paths through that vertex
   end;
4. - Make the appropriate transformations
5. - ETL Process and Deployment

End.

4. CASE STUDY

This section describes the tools and software used in this work as well as a case study is considered to illustrate the underlying principles of the proposed Os-ETL.

4.1 The ETL Spark engine

Originally Spark was developed in 2009 on UC Berkeley AMPLab. It became open source in 2010. Spark is a cluster computing platform designed to be fast and general purpose. It is highly accessible, offering simple APIs in Python, Java, R, Scala and SQL, and rich built-in libraries. It also supports advanced tools, such as Spark Streaming, MLlib for machine learning, GraphX for processing graphs, and Spark SQL for processing SQL and structured data. Spark extends the MapReduce model to support more types of computations, including interactive queries and streaming processing. One of the main features offered by Spark for speed is the ability to perform in-memory calculations [39].

The most components used in Spark are:

4.1.1 Resilient distributed datasets

A resilient distributed dataset (RDD), is the basic abstraction in Spark, is a read-only collection of objects partitioned across a number of machines that may be recreated in the event that a partition is lost. An RDD can be explicitly cached in memory between machines and used repeatedly in parallel tasks into MapReduce [40]. The ETL and RDD processes share a lot of correspondence and similarities: The activities and record sets that make up an ETL process can be seen as nodes in a directed acyclic graph (DAG), with the

input-output links between the nodes serving as the graph's edges [41]. In RDD, the created compute set, forms a DAG, it does not perform any execution, but it prepares for execution at the end. After the extraction, the ETL process performs transformations on the data such as (cleaning, filtering, converting, merging, aggregating...etc.). Thus, the ETL process fits well with Spark's RDD execution model.

4.1.2 DataFrame and DATASET

DataFrames are collections of structured records that can be manipulated using Spark's procedural API. They offer a complete set of functions (select columns, filter, join, aggregate, etc.) that solve common data analysis problems. A distributed collection of data is known as a dataset. A dataset is divided into columns called DataFrames, much like the tables of a database [40].

4.1.3 Spark's parallel processing

At a high level of abstraction, Spark applications run as independent processes that reside on clusters and are coordinated by SparkContext in the main program. Each Spark application consists of a driver program that executes the user's main function and initiates several parallel operations on the cluster. The main steps for running a spark program are:

- The first step in running a Spark program is to submit the job using spark-submit.
- The spark-submit script is used to launch the program on a cluster.
- Once the job is submitted, the SparkContext pilot program is the entry point to Spark.
- SparkContext routes the program to the modules like Cluster Master Node and RDDs are also created by these SparkContext driver programs.
- The program is then transmitted to the Cluster Master Node. Each cluster has a master node that does all the necessary processing. It then transmits the program to the worker nodes. The working node is the problem solver. Master nodes contain executors that run with the SparkContext driver.

4.2 ETL Deployment over a polystore

According to Stonebraker and Çetintemel [42], a polystore system is a database system having various heterogeneous data stores and diverse query interfaces. Due to this heterogeneity of the storage systems in the current Big Data ecosystem, many dissimilar backends or federated storage engines are required [2].

Data warehouse storage is impacted by variety, and polystore are well suited to achieve excellent data access performance [43]. So, we claim that to get the added value from business, we have to deal with variety. In our approach, we employ the Spark SQL hybrid polystore for deploying the target data warehouse, whose architecture is depicted in Figure 6.

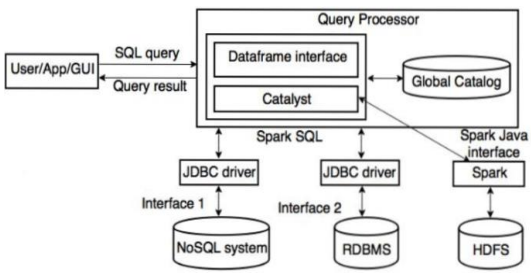

Figure 6. The Spark SQL polystore architecture [44]

4.3 Data-Sets

In order to validate the effectiveness and efficiency of the proposed system, we illustrate a case study inspired from a project of the training department of the Ministry of Higher Education and Research which wants to build a data warehouse which consists of collecting actionable information on university students, namely their performance in publications.

The LUBM (http://swat.cse.lehigh.edu/projects/lubm/.) benchmark (related to the academic field is used to generate the data source schemas. It generates data from different universities, such as each university has number of departments, professors, students and courses. We conducted a series of experiments using a data warehouse schema, which is presented in Figure 7, taken from the LUBM benchmark related to the academic domain.

A tool named UBA is provided by LUBM to generate data on the Univ-Bench ontology. These datasets were used to create simulated graphs based on outside data sources. As internal sources, we also took into account CSV data sources and a relational database. These sources are as follows:

- Source 1 is a Neo4j Graph Database with nodes, edges: Person, Student(name), Publication, Publication Author, University.
- Source 2 is a CSV file composed of attributes: Student(name), Publication, University.
- Source 3 is an Oracle database composed of tables and attributes: Student(name), Course(title), University.

We applied the Os-ETL algorithm and the Scala scripts to populate the target data warehouse schema deployed on the Spark SQL polystore.

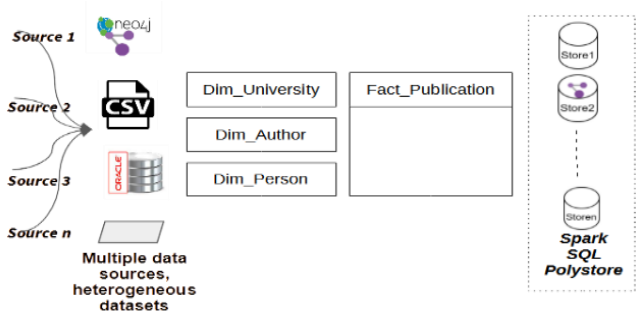

Figure 7. The data warehouse example

Example of running the Os-ETL Algorithm on CSV file: Spark ETL scripts import CSV data to several nodes. Spark begins execution after switching the transforms from RDD to DAG. A logical execution plan is transformed into a physical execution plan using a DAG. The DAG is sent to the DAG scheduler when an activity is invoked. It divided on sub-processes, which consist of a single task. The program can then be deployed on many machines through the transmission of these jobs to the task manager via a cluster manager. The Spark framework handles processing from a distribution standpoint. Each DataFrame, whether persistent or not, is a partitioned collection.

4.4 Metrics

To evaluate any ETL system efficiency, the response time is one of the important metrics. The response time after execution of the Os-ETL on the whole data involved in the

integration. Another metric is the performance time of the ETL system, where the performance is measured in term of scalability. The size of fact data is scaled and the execution time taken for each size of data is collected and plotted [45].

5. EXPERIMENT AND RESULTS

In this section, we present the evaluation of the proposed system based on the above case study. Thus, three different experiences were conducted. The first experiment is a comparison of the proposed Os-ETL with an existing approach that does not use a partitioning technique. The second experiment is also a comparison with an existing approach that uses partitioning technique. The third experiment considers the scalability of our solution.

The Hardware used to run the tests was PC having Intel(R) Core (TM) i5-8350U (1.7 GHz) processor, 8 GB main memory on a 500 GB SSD hard disc. The software used: Microsoft Windows 10x64 Professional, Software specifications: Oracle 12c, Apache Spark 2.4.4, Scala 4.7, Neo4j graph database management system, and Eclipse IDE. All of the software tools were installed on the local machine for evaluation purposes.

The data source schemas are from the LUBM benchmark. We use real world datasets: CSV file, Oracle database and Neo4j database. From this set we generated five RDF datasets in N-Triple format as shown in Table 1. The obtained data warehouse has two stores. For its deployment, we used Oracle Database 12c as the database backend for store 1 and Neo4j Graph Database for store 2.

Table 1. Datasets

Concepts (University)	RDF (N-Triples)
3	21 057
6	42 115
9	63 173
12	84 231
15	105 289

Experiment 1. We run the Os-ETL algorithm to populate the target data warehouse schema, and we compare the response time of our method with previous study [46] (no partitioning strategy). Figure 8 illustrates the reached results, where the number of instances is shown in thousands and the time performance in milliseconds. According to the results obtained, it appears that the Os-ETL was the most profitable because the response time of the ETL was divided by 4 (for example for 35000 instances the response time was 4000 ms) compared to the approach [46]. The results, clearly show that the partitioning strategy significantly improves the response time of the ETL processes.

Experiment 2. We run the Os-ETL algorithm to populate the target data warehouse schema deployed on the Spark SQL polystore over two stores (Neo4j Graph Database and Oracle Database). We measure the time spent in integrating the instances in each multidimensional concept. Figure 9 presents the obtained results, where the number of instances is shown in thousands and the time performance in milliseconds. We compare the performance of building the target data warehouse of the Os-ETL with another previous study [45] (with partitioning strategy). The results demonstrated that the proposed system is much more efficient.

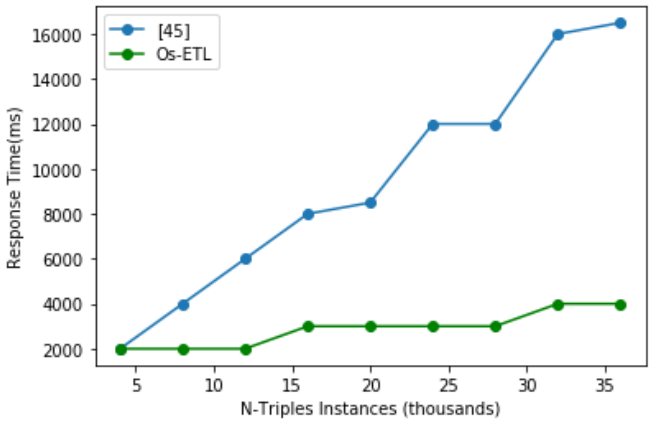

Figure 8. Os-ETL performance vs. [46] ETL

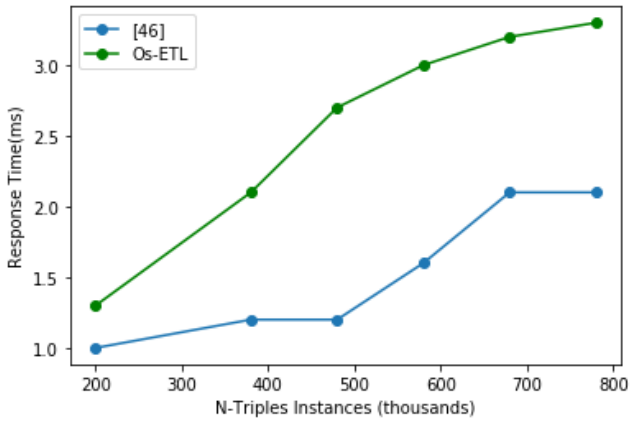

Figure 9. Os-ETL performance vs. [45] ETL

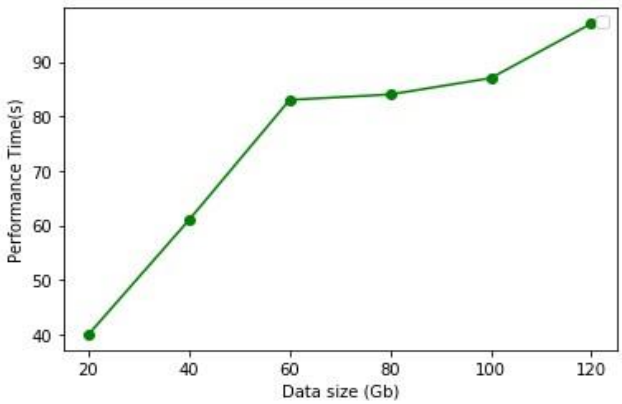

Figure 10. Os-ETL performance

Experiment 3. We considered the scalability of the Os-ETL system by varying the size of the data sources and therefore the number of tasks that run in parallel on the Spark cluster to process the data. We measure the performance of the proposed system in terms of scalability.

The size of fact data is scaled from 20 to 120 GB. Figure 10 shows the results. The combined use of CDC, partitioning and polystore techniques show that the results illustrated in Figure 10 show that the temporal performance of our system is very good compared to the size of the data set which increases. The results confirm that this is a good choice for the partitioning strategy and therefore the present study confirm the results regarding scalability.

6. CONCLUSIONS

In this paper, we have proposed a new solution called Os-ETL which consist of six steps: (i) extracting and transferring stream data; (ii) aligning all data sources to the RDF data model; (iii) partitioning data sources (iv) transforming RDF data into GraphX data; (v) appropriate transformations on GraphX data; and (vi) allocation and distribution of the obtained RDF fragments on the Spark SQL polystore. First, we have described the components used by Os-ETL which extracts heterogeneous data sources involving web-data as external data. Two techniques, the CDC method and the partitioning strategy were used and analyzed that have greatly improved the response time of ETL. Also, we proposed the usage of polystore systems as a hardware solution for deploying the target data warehouse and ETL processes. Os-ETL system significantly improves ETL processes and allows developers to focus on business logic, rather than worrying about the complex process of extracting-transforming-loading data in a highly varied environment. The most important results are related to the performance time of the ETL processes compared to the previous work, thanks to the partitioning, in-memory and pipeline strategies. The obtained results confirm the better performance of the Os-ETL system in terms of scalability and optimization. As future work, the proposed system can be extended in several directions. The system can be designed to support other types of heterogeneous data sources such as Linked Open Data (LOD). The system can be tested with several benchmarks such as TPC-DI.

ACKNOWLEDGMENT

This work has been funded and supported by the General Direction of Scientific Research and Technological Development of the Algerian Ministry of Higher Education and Scientific Research.

The authors would like to thank all the team of the LRIT (Laboratoire de recherche en informatique de Tlemcen) Laboratory of the University of Tlemcen for their help to carry out the experiments.

REFERENCES

[1] Zdravevski, E., Lameski, P., Apanowicz, C., Ślęzak, D. (2020). From Big Data to business analytics: The case study of churn prediction. Applied Soft Computing, 90: 106164. https://doi.org/10.1016/j.asoc.2020.106164

[2] Meehan, J., Aslantas, C., Zdonik, S., Tatbul, N., Du, J. (2017). Data ingestion for the connected world. In CIDR, 17: 8-11.

[3] Du, J., Meehan, J., Tatbul, N., Zdonik, S. (2017). Towards dynamic data placement for polystore ingestion. In Proceedings of the International Workshop on Real-Time Business Intelligence and Analytics, pp. 1-8. https://doi.org/10.1145/3129292.3129297

[4] Berkani, N., Bellatreche, L., Guittet, L. (2018). ETL processes in the era of variety. In: Hameurlain, A., Wagner, R., Benslimane, D., Damiani, E., Grosky, W. (eds) Transactions on Large-Scale Data- and Knowledge-Centered Systems XXXIX. Lecture Notes in Computer Science, vol. 11310. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-58415-6_4

[5] Stonebraker, M., Abadi, D., DeWitt, D.J., Madden, S., Paulson, E., Pavlo, A., Rasin, A. (2010). MapReduce and parallel DBMSs: Friends or foes? Communications of the ACM, 53(1): 64-71. https://doi.org/10.1145/1629175.1629197

[6] Longo, A., Giacovelli, S., Bochicchio, M.A. (2014). Fact–Centered ETL: A proposal for speeding business analytics up. Procedia Technology, 16: 471-480. https://doi.org/10.1016/j.protcy.2014.10.114

[7] Masouleh, M.F., Kazemi, M.A., Alborzi, M., Eshlaghy, A.T. (2016). Optimization of ETL process in data warehouse through a combination of parallelization and shared cache memory. Engineering, Technology & Applied Science Research, 6(6): 1241-1244. https://doi.org/10.48084/etasr.849

[8] Salloum, S., Dautov, R., Chen, X., Peng, P.X., Huang, J.Z. (2016). Big data analytics on Apache spark. International Journal of Data Science and Analytics, 1: 145-164. https://doi.org/10.1007/s41060-016-0027-9

[9] Bajaber, F., Elshawi, R., Batarfi, O., Altalhi, A., Barnawi, A., Sakr, S. (2016). Big data 2.0 processing systems: Taxonomy and open challenges. Journal of Grid Computing, 14: 379-405. https://doi.org/10.1007/s10723-016-9371-1

[10] Belayadi, Y., Khababa, A., Attia, A., Maza, S. (2022). An effective method based on bi-clustering and association rules for user activity analysis in location-based social network. Ingénierie des Systèmes d'Information, 27(6): 855-864. https://doi.org/10.18280/isi.270601

[11] Alsanad, A.A., Chikh, A., Mirza, A. (2019). Multilevel ontology framework for improving requirements change management in global software development. IEEE Access, 7: 71804-71812. https://doi.org/10.1109/ACCESS.2019.2916782

[12] Chen, J., Chen, Y., Du, X., Li, C., Lu, J., Zhao, S., Zhou, X. (2013). Big data challenge: A data management perspective. Frontiers of Computer Science, 7(2): 157-164. https://doi.org/10.1007/s11704-013-3903-7

[13] Vassiliadis, P., Simitsis, A. (2008). Near real time ETL. In: Kozielski, S., Wrembel, R. (eds) New Trends in Data Warehousing and Data Analysis. Annals of Information Systems, vol 3. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-87431-9_2

[14] Bornea, M.A., Deligiannakis, A., Kotidis, Y., Vassalos, V. (2011). Semi-Streamed Index Join for near-real time execution of ETL transformations. In 2011 IEEE 27th International Conference on Data Engineering, Hannover, Germany, pp. 159-170. https://doi.org/10.1109/ICDE.2011.5767906

[15] Muddasir, N.M., Raghuveer, K. (2017). Study of methods to achieve near real time ETL. In 2017 International Conference on Current Trends in Computer, Electrical, Electronics and Communication (CTCEEC), Mysore, India, pp. 436-441. https://doi.org/10.1109/CTCEEC.2017.8455002

[16] Biswas, N., Mondal, K.C. (2022). Integration of ETL in cloud using spark for streaming data. In: Mandal, J.K., De, D. (eds), Advanced Techniques for IoT Applications. EAIT 2021. Lecture Notes in Networks and Systems, vol. 292. Springer, Singapore. https://doi.org/10.1007/978-981-16-4435-1_18

[17] Berkani, N., Bellatreche, L., Ordonez, C. (2018). ETL-

aware materialized view selection in semantic data stream warehouses. In 2018 12th International Conference on Research Challenges in Information Science (RCIS), Nantes, France, pp. 1-11. https://doi.org/10.1109/RCIS.2018.8406668

[18] Boury-Brisset, A.C. (2013). Managing semantic big data for intelligence. In Stids, pp. 41-47.

[19] Pareek, A., Khaladkar, B., Sen, R., Onat, B., Nadimpalli, V., Lakshminarayanan, M. (2018). Real-time ETL in Striim. In Proceedings of the International Workshop on Real-Time Business Intelligence and Analytics, pp. 1-10. https://doi.org/10.1145/3242153.3242157

[20] Bansal, S.K. (2014). Towards a semantic extract-transform-load (ETL) framework for big data integration. In 2014 IEEE International Congress on Big Data, Anchorage, AK, USA, pp. 522-529. https://doi.org/10.1109/BigData.Congress.2014.82

[21] Ali, S.M.F., Wrembel, R. (2017). From conceptual design to performance optimization of ETL workflows: current state of research and open problems. The VLDB Journal, 26(6): 777-801. https://doi.org/10.1007/s00778-017-0477-2

[22] Thomsen, C., Pedersen, T.B. (2011). Easy and effective parallel programmable ETL. In Proceedings of the ACM 14th International Workshop on Data Warehousing and OLAP, pp. 37-44. https://doi.org/10.1145/2064676.2064684

[23] Liu, X., Iftikhar, N. (2015). An ETL optimization framework using partitioning and parallelization. In Proceedings of the 30th Annual ACM Symposium on Applied Computing, pp. 1015-1022. https://doi.org/10.1145/2695664.2695846

[24] Liu, X., Thomsen, C., Pedersen, T.B. (2013). ETLMR: A highly scalable dimensional ETL framework based on MapReduce. Transactions on Large-Scale Data-and Knowledge-Centered Systems VIII: Special Issue on Advances in Data Warehousing and Knowledge Discovery, 1-31. https://doi.org/10.1007/978-3-642-37574-3_1

[25] Ma, K., Yang, B. (2015). Log-based change data capture from schema-free document stores using MapReduce. In 2015 International conference on cloud technologies and applications (CloudTech), Marrakech, Morocco, pp. 1-6. https://doi.org/10.1109/CloudTech.2015.7336969

[26] Valêncio, C.R., Marioto, M.H., Zafalon, G.F.D., Machado, J.M., Momente, J.C. (2013). Real time delta extraction based on triggers to support data warehousing. In 2013 International Conference on Parallel and Distributed Computing, Applications and Technologies, pp. 293-297. https://doi.org/10.1109/PDCAT.2013.52

[27] Duggan, J., Elmore, A.J., Stonebraker, M., Balazinska, M., Howe, B., Kepner, J., Zdonik, S. (2015). The BigDAWG Polystore system. ACM SIGMOD Record, 44(2): 11-16. https://doi.org/10.1145/2814710.2814713

[28] Gurajada, S., Seufert, S., Miliaraki, I., Theobald, M. (2014). TriAD: A distributed shared-nothing RDF engine based on asynchronous message passing. In Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, pp. 289-300. https://doi.org/10.1145/2588555.2610511

[29] Huang, J., Abadi, D.J., Ren, K. (2011). Scalable SPARQL querying of large RDF graphs. Proceedings of the VLDB Endowment, 4(11): 1123-1134. https://doi.org/10.14778/3402707.3402747

[30] Karypis, G., Kumar, V. (1997). METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. Computer Science & Engineering (CS&E) Technical Reports [749].

[31] Al-Ghezi, A., Wiese, L. (2018). Adaptive workload-based partitioning and replication for RDF graphs. In: Hartmann, S., Ma, H., Hameurlain, A., Pernul, G., Wagner, R. (eds) Database and Expert Systems Applications. DEXA 2018. Lecture Notes in Computer Science, vol. 11030. Springer, Cham. https://doi.org/10.1007/978-3-319-98812-2_21

[32] Wu, B., Zhou, Y., Yuan, P., Liu, L., Jin, H. (2015). Scalable SPARQL querying using path partitioning. In 2015 IEEE 31st International Conference on Data Engineering, Seoul, Korea (South), pp. 795-806. https://doi.org/10.1109/ICDE.2015.7113334

[33] Gonzalez, J.E., Xin, R.S., Dave, A., Crankshaw, D., Franklin, M.J., Stoica, I. (2014). Graphx: Graph processing in a distributed dataflow framework. In 11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14), pp. 599-613.

[34] Boukhari, I., Bellatreche, L., Jean, S. (2012). An ontological pivot model to interoperate heterogeneous user requirements. In: Margaria, T., Steffen, B. (eds) Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies. ISoLA 2012. Lecture Notes in Computer Science, vol 7610. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-34032-1_35

[35] Knublauch, H., Fergerson, R.W., Noy, N.F., Musen, M.A. (2004). The Protégé OWL plugin: An open development environment for semantic web applications. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds). The Semantic Web – ISWC 2004. ISWC 2004. Lecture Notes in Computer Science, vol. 3298. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-30475-3_17

[36] Jörg, T., Deßloch, S. (2008). Towards generating ETL processes for incremental loading. In Proceedings of the 2008 International Symposium on Database Engineering & Applications, pp. 101-110. https://doi.org/10.1145/1451940.1451956

[37] Skoutas, D., Simitsis, A. (2006). Designing ETL processes using semantic web technologies. In Proceedings of the 9th ACM International Workshop on Data Warehousing and OLAP, pp. 67-74. https://doi.org/10.1145/1183512.1183526

[38] Berkani, N., Bellatreche, L., Benatallah, B. (2016). A value-added approach to design BI applications. In: Madria, S., Hara, T. (eds). Big Data Analytics and Knowledge Discovery. DaWaK 2016. Lecture Notes in Computer Science, vol. 9829. Springer, Cham. https://doi.org/10.1007/978-3-319-43946-4_24

[39] Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I. (2010). Spark: Cluster computing with working sets. HotCloud, 10(10-10): 95.

[40] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauly, M., Franklin, M.J., Shenker, S., Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12), pp. 15-28.

[41] Vassiliadis, P., Simitsis, A. (2009). Extraction, transformation, and loading. Encyclopedia of Database Systems, 10.
[42] Bimonte, S., Gallinucci, E., Marcel, P., Rizzi, S. (2022). Logical design of multi-model data warehouses. Knowledge and Information Systems, 66: 1067-1103. https://doi.org/10.1007/s10115-022-01788-0
[43] Stonebraker, M., Çetintemel, U. (2018). "One size fits all" an idea whose time has come and gone. In Making Databases Work: The Pragmatic Wisdom of Michael Stonebraker, pp. 441-462. https://doi.org/10.1145/3226595.3226636
[44] Bondiombouy, C., Valduriez, P. (2016). Query processing in multistore systems: An overview. International Journal of Cloud Computing, 5(4): 309-346. https://doi.org/10.1504/IJCC.2016.080903
[45] Berkani, N., Bellatreche, L. (2018). Streaming ETL in polystore era. In: Vaidya, J., Li, J. (eds). Algorithms and Architectures for Parallel Processing. ICA3PP 2018. Lecture Notes in Computer Science, vol 11336. Springer, Cham. https://doi.org/10.1007/978-3-030-05057-3_42
[46] Berkani, N., Bellatreche, L., Khouri, S. (2013). Towards a conceptualization of ETL and physical storage of semantic data warehouses as a service. Cluster Computing, 16(4): 915-931. https://doi.org/10.1007/s10586-013-0266-7