Big-Parallel-ETL: New ETL for Multidimensional NoSQL Graph Oriented Data

Nassima Soussi
LIPIM, National School of Applied Sciences, Sultan Moulay Slimane University, Khouribga, Morocco

nassima.soussi@gmail.com

Abstract. The quantitative explosion of digital data derived from social networks, smart devices, IoT sensors, etc. is eventuated by the Big Data concept considered as a very important aspect in the performance improvement of traditional decision-making systems since it reveals serious challenges to be addressed. Therefore, the main purpose of this research paper is the integration of NoSQL Graph-oriented Data into Data Warehouse to deal with Big Data challenges especially with the absence of similar approaches to the best of our knowledge. In this paper, we propose a new approach called Big-Parallel-ETL that aims to adapt the classical ETL process (Extract-Transform-Load) with Big Data technologies to accelerate data handling based on the famous MapReduce concept characterized by its efficient parallel processing feature. Our solution proposes a set of detailed Algorithms based on several rules able to conceive rapidly and efficiently the target multidimensional structure (dimensions and facts) from the NoSQL Graph oriented database.

1. Introduction
Nowadays, the decision-making systems are facing a huge volume of digital data [15] due to the increased amount of heterogeneous data sources such as social networks, smart devices, sensors, etc. The size of this exploded data can reach 2.5 trillion gigabytes per day and 600 zetabytes every year from internet of things according to IBM and Cisco respectively. These challenges create significant limitations on data warehouse building, storing, processing, and analyzing in the right time. Consequently, the concept of Big Data [16] has emerged to face this continuous demand.

The Big Data requires a set of new characteristics summarized by the 3V rules which has been enlarged subsequently to 5 V and then 7V: Volume (quantity of massive data), Variety (heterogeneity of data types and sources), Velocity (the speed of generating data), and Veracity (authenticity and accuracy of data), Value (getting value from the data), Virtualization (using charts and graphs to visualize large amounts of complex data) and Variability (different from variety: evolving behavior in data sources). Otherwise, in order to analyze and process the massive data according to the previous Vs rules, the Big Data ecosystem is founded on a set of new technologies such as NoSQL [19], Hadoop [17], Spark [20], MapReduce [18], etc.

The previous technologies offers new opportunities to enhance decision-making process considering the first main elements called ETL (Extract-Transform-Load) tool treated by the majority of data warehouse projects [21]. In fact, the ETL process is a set of multiple operations that begin with extracting required data from input sources, and then transforming the extracted data into a standard...
format in order to load them into a prepared data warehouse. However, standard operations supported by the classical ETL are not suitable to deal with the massive evolution of data which makes the improvement of ETL process a serious need in order to deal with Big Data in decision domain.

In this context, we present in the current paper a new approach for integration Big Data technologies in ETL process in order to add parallelism aspect and minimize time-consumption. In fact, we have proposed a new solution called Big-Parallel-ETL that integrates NoSQL Graph oriented data in a data warehouse by ameliorating ETL process with MapReduce paradigm characterized by its efficient parallel processing feature used to accelerate big data handling. Our solution is presented as a set of detailed Algorithms based on several rules able to conceive rapidly and efficiently the target multidimensional structure (Dimensions and facts) from NoSQL Graph oriented database.

The remainder of this paper is structured as follows: Section II analyses the most recent approaches related to the current topic. Section III introduces the context of our work by highlighting the main concepts used in the proposed solution, such as NoSQL Graph oriented databases, multidimensional model of data warehousing and the famous MapReduce paradigm. Section IV presents the functional architecture of the conceived Big-Parallel-ETL solution with detailed algorithms for each phase. Finally, section V concludes our work and suggests some future extensions of this topic.

2. Related works
In the last decades, several studies have been established to improve the performance of data warehousing with Big Data emergence. Accordingly, we present in this section the most recent approaches published in the last five years that deal with data integration by focusing on ETL modeling to achieve Big data warehousing projects.

Parallel-ETL is a conceptual modeling approach proposed in the work [8] aiming to provide a parallel ETL processes composed of five steps: Extraction Partitioning, Transformation, Reduction and Loading. The MapReduce job begins when the data source is completely loaded by P-ETL in HDFS which contribute to accelerate the ETL processing by 33%.

The work presented in [5] propose an approach called BigDimETL (Big Dimensional ETL) that focuses on enhancing classical ETL processes in order to integrate Big Data from Hbase as a NoSQL column oriented database into a target data warehouse. This process is realized using Map Reduce paradigm but they only covered the selection and projection operations in the transformation phase. However, the approaches [6] and [7] are the ameliorated versions of the previous work. In fact, in the extraction phase, they conceive a multidimensional structure from input data as a column oriented tables instead of xml format. Regarding the transformation phase, they cover join operation ignored in the initial investigation.

Paper [9] presents an architecture describing the workflow of an extensible ETL framework that address Big Data challenges. The first layer of this architecture is Workflow Designer which communicates with the intermediate extensible layer composed of four elements: UDFs, recommender component, cost model and monitoring agent. This specific framework conception contributes in the optimization and the overall automation of ETL workflow in Big Data context using recommendations, monitoring and UDFs provided by the proposed system.

D_ELT or Delayed_ELT approach [10] consists to delay transformation phase although all the raw data is loaded to Hadoop and then the transformation task is associated with the analysis task to be performed together using the same MapReduce job. In this way, a part of the transformed data can be analyzed immediately without having to wait for all the data to be transformed. However, this approach has several shortcomings when the same type of analysis must be carried out repeatedly, as well as the negative impact of delaying transformation phase on big data storage.

The work [11] proposes a cloud-based architecture in order to conceive an ETL supporting Big Data. In fact, Extraction and Transformation phases are executed in demand by Spark, and then the resulted data is loaded to a data warehouse using distributed load agents (DLAs). After that, on-demand Hadoop clusters are used by ETL process with a variable size that runs for a limited duration on Amazon AWS.
The previous research works present several interesting solutions regarding the enhancement of ETL processes in order to integrate massive data efficiently in a target data warehouse using several big data technologies such as MapReduce paradigm, Spark, cloud computing and other tools. Obviously, the current works shared some similarities with ours since they are interested in building a data warehouse based on massive data but they share the same disadvantage of ignoring to process NoSQL Graph oriented databases considered as one of the most used NoSQL systems for managing highly connected big data and supporting complex queries. In addition, the development of solutions supporting this kind of approaches is still limited [12] [13].

3. Background Context

In this section, we introduce the main concepts used in our proposed solution that covers the fundamental definitions of NoSQL Graph-oriented database taking the example of Neo4j and the multidimensional structure of Data warehouse. Furthermore, we describe the main structure of MapReduce paradigm.

3.1. NoSQL Graph-oriented Database

Graph technology has been the fastest growing category of databases in recent years. In the world where connected data represents a new source for companies, graph technology appears as the obvious option. Therefore, NoSQL Graph-oriented databases [1] are perfectly adapted to voluminous, heterogeneous and massively interconnected data due to its flexible structure capable to representing elegantly correlated and dynamic data.

This kind of databases (such as the famous one which is Neo4j [2] supported in this work) are based on graph structure composed by three fundamental components: Nodes, Relationships and Properties. Formally, we can represent a NoSQL Graph-oriented database as \( G = (N, R, P) \) where \( N = \{N_1, \ldots , N_k\} \) is a set of graph nodes, \( R = \{R_1, \ldots , R_p\} \) is a set of relationships between the nodes \( N \) and \( P = \{P_1, \ldots , P_n\} \) is a set of properties related to each component \( N \) or \( R \); each property is represented as key/value pair. The three major components cited previously are defined as follows:

- **Nodes (N)** represent the entities; Each node \( N \) can be formalized as \( N = (id_N, L_N, P_N) \) where: \( id_N \) is its identifier, \( L_N = \{L_1, \ldots , L_m\} \) is a set of labels describing its semantic and \( P_N \) is a set of its properties.
- **Relationships (R)** define the relations between connected nodes (N). Each relation \( R \) can be formalized as \( R = (id_R, L_R, N_{start}, N_{end}, P_R) \) where \( id_R \) is its identifier, \( L_R \) is its label (name), \( N_{start} \) is the identifier of the start node, \( N_{end} \) is the identifier of the end node and \( P_R \) is a set of its properties ( \( P = P_N \cup P_R \) ).

The metamodel illustrated in Figure 1 summarizes the fundamental structure of NoSQL Graph-oriented databases as detailed and formalized previously.

Neo4j supports exporting data into standard formats (JSON, CSV, GraphML, and Cypher script) based on APOC library [3]. In order to export data in these formats, Neo4j offers its users the possibility of exporting the whole database, specific nodes and relationships, a virtual graph, or the results of a Cypher query [4]. Figure 2 presents an example of Neo4j graph and its corresponding JSON format which will be considered in our proposed approach of NoSQL graph data warehousing since it is the most expressive one.
3.2. Conceptual Multidimensional Model

In the current paper, we have considered a multidimensional structure (MS) of a data warehouse as a star schema (S) composed by four fundamental concepts as described in Figure 3: Facts, Measures, Dimensions and Hierarchies. Formally, we can represent a multidimensional structure as $MS = (F, D, Link)$ where $F = \{F_1, \ldots, F_n\}$ is a set of facts in the case of constellation representation $C = \{S_1, \ldots, S_n\}$ or a single one, $D = \{D_1, \ldots, D_m\}$ is a set of dimensions and $Link$ is a link function that associates $F$ to $D$ in order to create a linked schema ready to be analyzed.

A dimension $D$ constitutes an axis of analysis that can be represented as a table. Formally, it is defined as $D = (N_D, Attr, H)$ where $N_D$ is the name of dimension, $Attr = \{attr_1, \ldots, attr_p\}$ is its set of attributes describing more details of $D$, and $H = \{H_1, \ldots, H_k\}$ is a set of hierarchies composed of several levels. The set of attributes $Attr$ of a specific dimension $D_i \in D$ is organized into one or more hierarchies. In addition, a Fact $F$ represents the main analyzed subject (such as sale, stock, …). Each Fact $F \in F$ can be formalized as $F = (N_F, FK, M)$ where $N_F$ is the fact name, $FK = \{FK_1, \ldots, FK_q\}$ is a foreign keys set of the corresponding dimensions $D$ and $M = \{M_1, \ldots, M_j\}$ is a set of measures that each one is associated with an aggregation function (COUNT, MIN, AVG, …).
3.3. MapReduce Paradigm

MapReduce [18] is a programming model developed by Google in 2004 for parallel processing and generating large data sets on clusters. It is considered as a central component of Apache Hadoop software framework, which enables resilient and distributed processing of massive unstructured data sets on clusters of computers, in which each node has its own storage space. The MapReduce mechanism works in two major phases named Map and Reduce according to a set of jobs $J=\{J_1, \ldots, J_n\}$ where each job $J_i = [M_{\text{Func}}, R_{\text{Func}}]$ ($i \in n$) operates on Map function ($M_{\text{Func}}$) and Reduce function ($R_{\text{Func}}$). In addition, all data manipulated with this program is represented as key-value pairs.

The MapReduce job operates on a split input data resulted from an horizontal partitioning by respecting 64MB as a default split size. Each split (chunk) is treated by a single Map function ($M_{\text{Func}}$) that extracts the necessary data in the form of key-value list in order to sort them by key with shuffling function. The resulted sorted data represent the input value of Reduce function ($R_{\text{Func}}$) which performs the necessary processing on each data of a specific key (sum, average, …).

4. Proposed Approach

In this section, we describe our significant contribution summarized via the functional architecture presented in Figure 4. In fact, this architecture is composed of two principal layers; the first one contains the data sources of our system which is Neo4j considered as a NoSQL Graph oriented database, alimented from social media. The second layer presents the ETL process that operates with MapReduce paradigm as described subsequently in order to analyze and process input big data in order to be integrated efficiently in the target data warehouse.

The main operations supported by our big ETL approach are presented in Algorithm-1. In fact, we operate on the user query $Q$, in order to identify the requested nodes and relationships which allow us to extract the requested sub-graph from NoSQL Graph Oriented database (G) as a JSON format, more efficient to process than a graph representation that requires a complex algorithms based on graph theory [14]. Our main idea consists to operate on this JSON file contained Neo4j massive data combined with Data Warehouse metadata (DW-metadata) so as to prepare the multidimensional schema (dimensions and facts) via MapReduce paradigm starting with horizontal splitting operation.

---

**Figure 3.** Metamodel of Multidimensional Schema Concepts.
and then converting each sub-json file into column-oriented tables in order to conceive the equivalent multidimensional structure to be integrated in the target data warehouse.

**Algorithm-1: ETL Process**

**Input**: User query (Q), NoSQL Graph Oriented Data (G), Data warehouse metadata (DW-metadata)

**Output**: List of Multidimensional Elements (Tcol)

**Begin**

1. Q ← ParsingQuery(Q)
2. List nodes ← ExtractRequestedNodes(Q)
3. List relationships ← ExtractRequestedRelationships(Q)
4. Connect to NoSQL Graph-oriented Database
5. inputData ← ExtractDataAsJson(G, nodes, relationships)
6. List sub-json-files-list<K1, V1> ← horizontalSpliting(inputData, DW-metadata)
7. List L2<K2, list V2> ← Call Map(K1, V1)
8. Call Reduce(L2)

**Figure 4.** Global Architecture of our Approach.
9. Return $T_e$
End Algorithm

4.1. Step 1: Horizontal Splitting
In order to process efficiently the input graph oriented data in our big ETL approach adopting MapReduce paradigm, the first step as defined in Algorithm-2 is splitting horizontally the input big data represented as JSON dataset (J) into smaller sub-datasets (stored in sub-json-files-list as a list of Key-Value = K1-V1) according to Data Warehouse metadata (DW-metadata). In fact, we group JSON objects (obj) which correspond to the same axe of analyse (dimension or fact) in one sub-JSON file according to the following rules:

- **Rule 1**: if the json object type is a ‘node’ (the main unit in graph structure), then we check if it represents genuinely a Dimension or a Fact of the target Data Warehouse in order to recuperate it and store it in the final list sub-json-files-list that will regroups the resulted sub json files having the same axe of analysis as a key-value format {K1,V1} where K1 is the label of analysis axe and V1 is the corresponding json nested elements.

- **Rule 2**: if the json object type is a ‘relationship’ so in this case we aim to keep the connection semantics between the two nodes linked by this relation. In fact, our proposed solution consists to integrate the label of this relation as a new property in the source node, in addition to its value composed of the relationship properties concatenated with the end node id and label.

Algorithm-2: Horizontal Splitting

**Input**: Input Json File (J), DW-metadata

**Output**: List sub-json-files-list<K1, V1>

**Begin**
1. Dimensions ← ExtractDimensions(DW-metadata)
2. Fact ← ExtractFacts(DW-metadata)
3. List json-objects ← ExtractJsonObjects(J)
4. Foreach obj in json-objects do
5. Label ← obj.getLabel()
6. If obj.getType() = ‘node’ then
7. If (Label IN Dimensions) OR (Label IN Facts) then
8. If (Label NOT IN K1) then
9. K1 ← Label
10. End If
11. V1 ← Label.addValue(obj)
12. End if
13. ElseIf obj.getType() = ‘relationship’ then
14. startNodeId ← obj.getStart().getId()
15. startNodeLabel ← obj.getStart().getLabel()
16. endNodeId ← obj.getEnd().getId()
17. endNodeLabel ← obj.getEnd().getLabel()
18. RelshipLabel ← obj.getLabel()
19. RelshipProperties ← obj.getProperties()
20. If (startNodeLabel IN K1) then
21. O ← startNodeLabel getValue(startNodeId, startNodeLabel)
22. O.addPropertieLabel(RelshipLabel)
23. O.addPropertieValue(endNodeId ∪ endNodeLabel ∪ RelshipProperties)
24. End If
25. End if
4.2. Step 2: Conversion Process

The current step is responsible of converting sub-json files generated from the previous step into column-oriented tables (Table CF) with a unique column family (CF). In this case, each job J executes one Map function (defined in Algorithm-3) which operates on a specific sub-json file represented as \{K1, V1\} where its label is K1 and its content is V1. In fact, we operate on each sub-json object Obj in order to extract its id and properties to conceive the output list L2<K2, V2> where K2 includes the different combinations of attributes label for each object’s properties while V2 contains the corresponding K2 values.

In order to determine the target column family label of each Table CF, that will receive L2 list in reducing step, we have to index this list by K1 that refers to its corresponding dimension D or Fact F.

**Algorithm-3: Map Function**

**Input**: Name of a sub-json file (K1), sub-json file’s content (V1)

**Output**: List L2<K2, list V2>

**Variables**: List JsonObj, K, V, Children

**Begin**

1. JsonObjects ← ExtractSubJsonObjects(V1)
2. Foreach Obj in JsonObjects Do
3. Id ← Obj.getId()
4. Properties ← Obj.getProperties()
5. K.add('id') V.add(Id)
6. Foreach prop in Properties Do
7. Children ← prop.getChildren()
8. Foreach Child in Children Do
9. Child_key ← Child.getAttributeLabel()
10. Child_value ← Child.getAttributeValue()
11. K.add(Child_key)
12. V.add(Child_value)
13. End Foreach
14. If (K in K2) Then
15. getKey2(K).addValue(V)
16. Else
17. K2.add(K)
18. V2.add(V)
19. End if
20. End Foreach
21. End Foreach
22. Indexing L2 with K1
23. Return L2

**End Algorithm**

The next step is considered as the major one in the conversion operation using Reduce function described in Algorithm-4. In fact, it operates on L2 lists generated from Map functions in order to conceive column-oriented tables based on several conversion rules bellow defining the correspondance between each object Obj represented in L2 as \{K2, V2\} and the target tables Table CF.

- **Rule 1**: the index of each L2 list indicates the column family label (CF).
• **Rule 2**: for each L2 list, K2 is used to build the corresponding column-oriented table columns and V2 represents their values.

**Algorithm-4: Reducing Function**

**Input:** List L2<K2, list V2>  
**Output:** Table,<CF>  
**Begin**  
1. CF_label ← L2.getIndex()  
2. Table,.setCFLabel(CF_label)  
3. Foreach elt in L2 do  
4. If (Table,.isEmpty() = True) then  
5. Table,.addColumns(K2)  
6. Table,.addValuesOf(V2, K2)  
7. Else  
8. DiffCol ← ExtractDifferentColumns(K2, Table,.getColumns())  
9. Table,.addColumns(DiffCol)  
10. Table,.addValuesOf(V2, K2)  
11. End If  
12. End Foreach  
13. Return Table,  
**End Algorithm**

5. **Illustrative Example**

We aim to integrate the graph presented in Figure 2 into a target data warehouse starting by the portion presented in JSON format as a basic example. According to our approach, we start with splitting horizontally the input data to construct sub-json files presented as key-value pairs. In our case, we have two nodes labelled as ‘Person’ in addition to relationship ‘Knows’ that will integrates the start node with id=0 as described in Figure 4 below and according to Algorithm-2. The resulted sub-json file will be presented as a list of {K1,V1} with K1=‘Person’ and V1= the content presented in Figure 5.

```
{"type": "node",
 "id": "0",
 "labels": ["Person"],
 "properties": {
 "name": "Sara",
 "age": 35,
 "KNOWS": {
 "id": "1",
 "labels": ["Person"],
 "since": 2015
 }
 }

{"type": "node",
 "id": "1",
 "labels": ["Person"],
 "properties": {
 "born": "1991-07-21",
 "name": "Ahlam",
 "age": 28,
 "female": true
 }
 }
```

**Figure 5.** Example of the extracted sub-json file ‘Person’.

The previous sub-json file will be handled by Map function defined in Algorithm-3 in order to decompose their objects properties as key/value format stored in the list L2<K2, V2> as presented in
Table 1 where K2 includes the different combinations of attributes label for the sub-json file ‘Person’ and V2 contains the corresponding K2 values.

| K2                             | V2                              |
|--------------------------------|---------------------------------|
| { id, name, age, knows{id, Label, since}} | {0, Sara, 35, {1, Person, 2015}} |
| { id, born, name, age, female}  | {1, 1991, Ahlam, 28, true}      |

In the last step of our proposed approach, the Reduce function defined in Algorithm-4 operates on the previous list L2 in order to generate its equivalent column oriented table (CF-Person presented in Table 2) which corresponds to a dimension in the target data warehouse.

| Person_CF | id | name | born | age | female | knows          |
|-----------|----|------|------|-----|--------|----------------|
|           | 0  | Sara | 35   |     |        | id:1, label:Person, since:2015 |
|           | 1  | Ahlam| 1991 | 28  | true   |                      |

6. Conclusion
In order to adapt decision-making systems with Big Data management, we have proposed in the current paper a new solution named Big-Parallel-ETL that integrates NoSQL Graph oriented data in a target data warehouse by ameliorating ETL process with MapReduce paradigm characterized by its efficient parallel processing feature used to accelerate big data handling. In fact, we have developed a set of complex Algorithms based on several rules able to conceive rapidly and efficiently the target multidimensional structure (Dimensions and facts) from NoSQL Graph oriented data. As future work, we will focus on implementing the proposed Algorithms using Neo4j as one of the famous and more used NoSQL Graph oriented database.

References
[1] S. Boukettaya, A. Nabli, AND F. Gargouri, Towards the Evolution of Graph Oriented Databases. In International Conference on Intelligent Systems Design and Applications, Springer, Cham (2018) : pp. 392-399.
[2] Neo4j Graph Database. Available online: https://neo4j.com/ (accessed on 01 June 2020).
[3] Neo4j APOC Library. Available online: https://neo4j.com/developer/neoj4j-apoc/ (accessed on 01 June 2020).
[4] Cypher Query Language. Available online: https://neo4j.com/developer/cypher-query-language/ (accessed on 01 June 2020).
[5] H. Hallek, F. Ghozzi, O. Teste AND F. Gargouri, BigDimETL: ETL for Multidimensional, In International Conference on Intelligent Systems Design and Applications Springer, Cham (2016) : pp. 935-944.
[6] H. Mallek, F. Ghozzi, O. Teste AND F. Gargouri, BigDimETL with NoSQL Database, Procedia Computer Science, 126 (2018) : pp. 798-807.
[7] H. Mallek, F. Ghozzi AND F. Gargouri, Towards Extract-Transform-Load Operations in a Big Data context, International Journal of Sociotechnology and Knowledge Development (IJSKD), 12(2) (2020) : pp. 77-95.
[8] M. Bala, O. Boussaid AND Z. Alimazighi, Extracting-Transforming-Loading Modeling Approach for Big Data Analytics, International Journal of Decision Support System Technology (IJDSSST), 8(4) (2016) : pp. 50-69.
[9] S. M. F. Ali, Next-generation ETL Framework to Address the Challenges Posed by Big Data, In DOLAP, (2018).

[10] J. Jo AND K.-W. Lee, MapReduce-Based D_ELT Framework to Address the Challenges of Geospatial Big Data, ISPRS International Journal of Geo-Information, 8(11) (2019) : pp. 475.

[11] E. Zdravevski, P. Lameski, A. Dimitrievski, M. Grzegorowski AND C. Apanowicz, Cluster-size optimization within a cloud-based ETL framework for Big Data, In 2019 IEEE International Conference on Big Data (Big Data), IEEE (2019) : pp. 3754-3763.

[12] Y. Liu AND T. M. Vitolo, Graph data warehouse: Steps to integrating graph databases into the traditional conceptual structure of a data warehouse, In 2013 IEEE International Congress on Big Data, IEEE (2013) : pp. 433-434.

[13] A. Sellami, A. Nabli AND F. Gargouri, Transformation of Data Warehouse Schema to NoSQL Graph Data Base, In International Conference on Intelligent Systems Design and Applications, Springer, Cham (2018) : pp. 410-420.

[14] K. Thulasiraman AND M. N. Swamy, Graphs: theory and algorithms. John Wiley & Sons, (2011).

[15] H. El Bousty, S. D. krit, M. Elasikri, H. Dani, K. Karimi, K. Bendaoud AND M. Kabrane, Investigating business intelligence in the era of big data: Concepts, benefits and challenges, In Proceedings of the Fourth International Conference on Engineering & MIS (2018) : pp. 1-9.

[16] A. Oussous, F. Z. Benjelloun, A. A. Lahcen AND S. Belfkih, Big Data technologies: A survey, Journal of King Saud University-Computer and Information Sciences, 30(4) (2018) : pp. 431-448.

[17] T. White, Hadoop: The Definitive Guide, O’Reilly Media, Inc., Sebastopol (2012).

[18] J. Dean AND S. Ghemawat, MapReduce: simplified data processing on large clusters. Commun. ACM 51(1) (2008) : pp. 107–113.

[19] C. Strauch, U. L. S. Sites AND W. Kriha, NoSQL databases. Lecture Notes, Stuttgart Media University, 20 (2011) : 24.

[20] S. Salloum, R. Dautov, X. Chen, P. X. Peng AND J. Z. Huang, Big data analytics on Apache Spark. International Journal of Data Science and Analytics, 1(3-4) (2016) : pp. 145-164.

[21] M. Bala, O. Boussaid AND Z. Alimazighi, P-ETL: Parallel-ETL based on the MapReduce paradigm. In 2014 IEEE/ACS 11th International Conference on Computer Systems and Applications (AICCSA), IEEE (2014) : pp. 42-49.