International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:14, No:12, 2020

Association Rules Mining and NOSQL Oriented Document in Big Data

Sarra Senhadji, Imene Benzeguimi, Zohra Yagoub

***Abstract*—**Big Data represents the recent technology of manipulating voluminous and unstructured data sets over multiple sources. Therefore, NOSQL appears to handle the problem of unstructured data. Association rules mining is one of the popular techniques of data mining to extract hidden relationship from transactional databases. The algorithm for finding association dependencies is well-solved with Map Reduce. The goal of our work is to reduce the time of generating of frequent itemsets by using Map Reduce and NOSQL database oriented document. A comparative study is given to evaluate the performances of our algorithm with the classical algorithm Apriori.

***Keywords*—**Apriori, Association rules mining, Big Data, data mining, Hadoop, Map Reduce, MongoDB, NoSQL.

I. Introduction

TODAY there is a huge amount of digital data that can be used to achieve several goals. This evolution of data is called "Big data". This kind of data is available in different formats and is propagated at a high streaming speed. These characteristics has opened up new opportunities in different fields such as industry, commerce, health and much more. However, some problems must be handled in terms of storage and how to process these enormous sets of data in order to use them effectively and obtain hidden information. Association rule mining is one of the most used data mining techniques to extract knowledge from large databases. Extracting frequent item set is the most important step in the process of finding association rule mining, which allows finding relationships between the data to be presented in simple and understandable rules that help in decision-making. However, the existing techniques of data mining must be adapted to the different characteristics of big data. In the literature, we find many algorithms of extracting of frequent itemsets: Apriori, FP-Growth. But, these algorithms are used mostly on structured data. That is why we propose to use the Apriori algorithm to extract frequent itemsets on NOSQL databases oriented document.

The rest of the paper is organized as follows: Section II some definitions are dressed to define the challenges we are facing, the related works are cited in Section III, and in Section IV we describe our work. Experimental results of our algorithm are shown in Section V. Finally, a conclusion is presented to summarize the main outcomes of this work.

Sarra Senhadji is with the University of Science and Technology of Oran, Algeria (e-mail: sarra.senhadji@gmail.com).

II. Definitions

In this section we explain briefly the most important challenges we are facing in a big data context.

A. Big Data [6]

Many definitions were proposed by different authors to try to describe the big data. For example in [8] Big Data is defined as "datasets whose size is beyond the ability of typical database software tools to capture, store, manage, and analyze". Likewise, Davis and Patterson [26] say that "Big data is data too big to be handled and analyzed by traditional database protocols such as SQL".

Many authors [8], [7], [3], [15] use the term 3V (Volume, Variety and Velocity) to characterize the Big Data.

- Volume (Data in rest). Volume is the principal characteristic of big data that refers to the large amount of data generated every minute from several sources.
- Variety (Data in many forms). These data do not have a fixed structure and are rarely presented in a perfectly ordered form to be ready for processing [13].
- Velocity (Data in motion). Velocity means the speed with which data are being generated, produced, created, or refreshed. Big Data helps the company to hold the incoming flow of data and at the same time process it fast.

More Vs are defined in [6], [13], [9] to describe the other characteristics of big data.

B. NOSQL

The relational database systems deal with structured data, where data are manipulated with the standard language SQL (Structured Query Language). However, due to the growth of the different varieties of format, new database systems must be developed to handle the problem of unstructured and scalable data. NoSQL (Not Only SQL) [18] was proposed to deals with non-relational databases systems, that are schema-free, scalable and BASE property (does not support full ACID property). NoSQL [12] has four principal categories: key-value databases, column-based, document-based and graph-oriented databases. In this work we focus on NOSQL oriented document.

C. Association Rule Mining

The association rule is one of the interesting techniques of data mining that has captured the attention of many researchers in different domains. This technique permits to find intelligible rules trough a large set of data.

Considering a transactional database D, an association rule has the form $X \rightarrow Y$, where X and Y are two itemsets and $X \cap Y = \emptyset$. An item set is composed of database's attributes; this

item is frequent if its support is higher than a defined minimum support. A rule is strong if its confidence is greater than a minimum confidence specified by a user.

There are two steps for extraction of the association rule mining:

1. The first step consists of finding frequent itemsets: APRIORI [1], FP-growth [16], Close [14] etc.
2. In the second step strong rules are defined from the generated frequent itemsets.

Even if the algorithm for extracting association rules seems simple and easy to implement, the first step is computationally intensive due to the phenomenal number of generated itemsets.

While the itemsets are generated the entire database is scanned every time. This means that the execution time is increased according to the number of transactions [5]. The purpose of the paper is to improve the Apriori algorithm by reducing the number of scan of the transactions.

Parallelism of item sets generating algorithms can play an important role in improving the execution time. This is well solved with map reduce [10, 11].

D. MapReduce

MapReduce [4] is a computational model that allows us to divide a big problem to smaller problems in parallel in order to speed up the time process.

A MapReduce algorithm is a programming model that will implement 2 phases, the mapping phase and the reducing phase. The map phase consists of a treatment that takes input data and maps it to <key, value> pairs. The reduce phase takes all the <key,value> with the same key and made a process on it .

The example in Fig. 1 illustrates how MapReduce works [24]. First, the input text file is split row by row. Each row will be "mapped" by a host. The mapping step is going to produce many associations of <key, value> pairs, in the above example the key is the word and the value is "1".

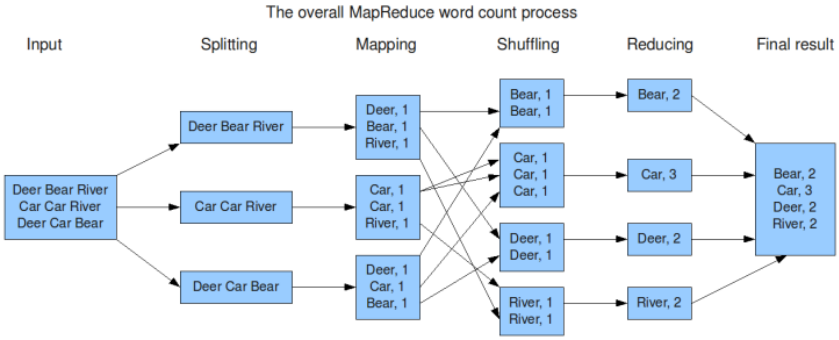

Fig. 1 MapReduce Example [24]

After, the produced <key, value> pairs are sorted according to the key. This is the most complicated part, but fortunately it is made by the MapReduce library.

Finally, a "reduce" task is performed by each node by counting all the <key, value> pairs which have the same key. The final result represents the number of instances of each word in the file text.

III. Related Works

In the literature the association rule minings are widely used in distributed systems and they have to be reviewed to deal with unstructured and scalable data [17], [25]. However, we found an interesting study of [2] which is close to our work. APRIORI is one of the implemented algorithms used by the authors. They use a cluster of 5 nodes of Hadoop/MapReduce. They also explore NoSQL databases oriented document by using MongoDB. Thus, Bson documents are used to store item sets with their support. The main steps are presented as:

1. The first step is to find L_1, which basically counts the occurrence of each item in the transactional database.

a. In Map Task, each document will input to the map (Key, Value) function. The key is items, and the value is one.

b. The Reduce Task, after shuffling and sorting the data, is based on keys In the Reduce (Key, Value []) function, the key is the item, and the value is a list of counts of that item. The Reduce task will iterate over the value and output the final sum.

2. In the second step, an auto join of $L_1 * L_1$ is required to build L_2. This can be done in Map task. Reduce task will find the final sum of the items, and filter them based on minimum support.
3. For a step K, an auto join of $L_{k-1} * L_{k-1}$ is required to build L_k
4. Step K is repeated until L_k is empty.

Our main idea is to review the process of generating frequent item sets in order to improve the execution time. In the next section, we will present in details our proposition.

IV. Methodology

In this section we present our algorithm and how we can ameliorate the classical Apriori algorithm.

A. Apriori and MapReduce

Our main idea is to ameliorate the algorithm APRIORI by reducing the time consumed for the item sets generating process. For this we propose to reduce the number of

transactions to be analyzed.

Proposed algorithm:

// for a transnational database T, a transaction is identified by an ID
input: ID_transaction, minsup
output: K-frequent itemsets

- First step consists of generating L1 (1-itemsets, support, ID_transaction). An itemset is defined with all the id of transactions where it appears. The support of itemset is calculated by a complete scan of the transnational database T. The Itemset that has a support greater than MinSup is retained for the next step.
- The second step consists of generating the itemset candidate L_2 by an auto join of L_1 * L_1 to build 2-itemset candidates (Ix,Iy) where Ix and Iy are itemset candidates of L_2. Before scanning all the transactions for calculating the L_2 candidate's support, we use the set L_1 to get the ID_transaction with the minimal support between Ix and Iy. This permits to limit the size of L_2 by scanning only a subset of transactions.
- For step k, we use the L_{k-1} list to generate the K-itemset candidates and the corresponding transactions to be scanned.
- We repeat this step until no more frequent itemsets can be found.

B. Proposed Algorithm

Pseudo code

• **Notation:**
TDB: transactional database
TID: ID of a transaction
t: transaction
C_k: Candidate itemset of size k
L_k: frequent itemset of size k
L_1= {frequent items};
Join Step: C_k is generated by joining $L_{k-1}$with itself
Prune Step: Any (k-1)-itemset that is not frequent cannot be a subset of a frequent k-itemset

//map reduce operation consists on generating itemsets, support and the ID of transaction to be scanned (TID)

First step: map reduce operation on itemsets of size 1

Map operation:
L_1 = find 1-itemsets;
return (1-itemsets, {TID});
Reduce operation:
count = 0;
For each transaction t in TDB do
increment the count of all 1-itemsets that are contained in t
if count>= MinSup then return (1-itemsets, (count,{TID})) ;
end;

Second step: map reduce operation on itemsets of size k

Map operation:
For (k= 1; $L_k \neq \emptyset$; k++) do
begin
tab={}; //list of transactions to be scanned
C_{k+1}= candidates generated from L_k;
x = find_ item_min_sup(C_{k+1}, L_K);
trans = find_TID (x);
add trans to tab;
For each transaction t in tab do
if k+1-itemsets in C_{k+1} then return (k+1-itemset, TID) ;
end;
Reduce operation:
count = 0;
For each transaction t in tab do
increment the count of all k-itemsets that are contained in t
if count>= MinSup then return (k-itemsets, (count,{TID})) ;
end;

V. Implementation and Results

In this section we present the tools and the datasets used to implement our algorithm. We realized experiments to compare our approach to the common frequent itemset search algorithm. The obtained results of different experimentation are presented to show the effectiveness of our work.

A. Environment

We implement our work on a virtual machine with VMware Workstation 12 Player [19] having the following configurations:

- **Operating system:** Ubuntu 14.04 LTS is used with 2 Go RAM and 30 Go disk space.
- **Hadoop**: Hadoop 2.6.5 [22] was installed over Ubuntu and set up with a Single Node Cluster.
- **Eclipse:** Algorithms are developed in Java by using the IDE Eclipse version 3.8.1[20].
- **MongoDB**: MongoDB [23] is an open source, document-oriented, schema free database that stores data as a BSON (Binary Simple Object Notion) document which is a binary encoded format of JSON. Each database may have multiple collections. Each collection may have many documents. MongoDB architecture has three core components: Mongod process handles data request and manages the underlying data format and data store.

B. Datasets

To experiment our work, we used two datasets of frequent mining itemsets [21]:

- **Retail**: contains the (anonymized) retail market basket data from an anonymous Belgian retail store. There are 88 161 transactions and each item represent the Id of the products.
- **Kosarak**: contains (anonymized) click-stream data of a Hungarian on-line news portal. It has 900 000 transactions and each item represent the ID user.

TABLE I
Datasets

Data set	#items	#transactions
Retail	16 470	88 161
Kosarak	41 270	900 000

The datasets used for our experiments are prepared to be ready for applying algorithms of generating frequent itemsets. The file of the transactional database is shown in Fig. 2 where each line contains items separated by a space.

C. MongoDB

In order to analyze the datasets, transactional databases must be imported to MongoDB. For this, the « mongoimport » command is executed in MongoDB to import JSON, CSV or TSV files.

Fig. 3 shows the import of the file retail. The collection is named retail where each line represents a unique id named ObjectId with the items of the transaction.

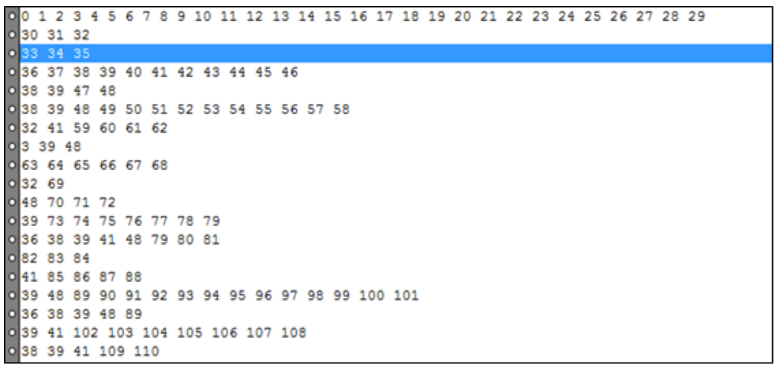

Fig. 2 Transactional database

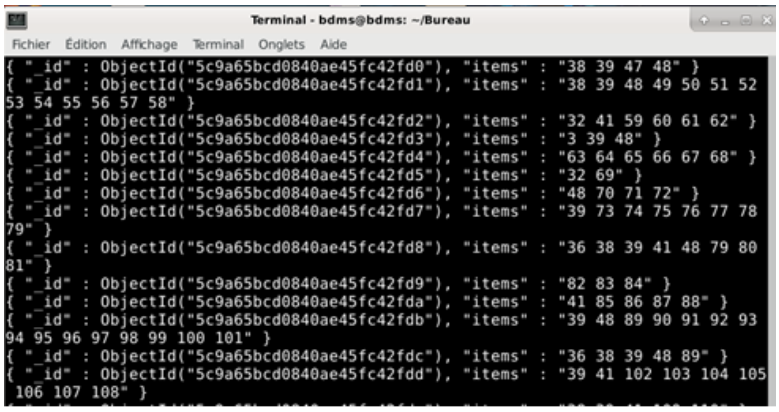

Fig. 3 Transactional database

Fig. 3 shows the import of the file retail. The collection is named retail where each line represents a unique id named ObjectId with the items of the transaction.

D. Comparative Study

In this section, we present the result of different experiments.

Experiment 1

In this we use the transactional database Retail, we vary the minimal support and we fix the minimal confidence to 0.1.

- *Execution Time vs Minsup*

TABLE II
EXECUTION TIME COMPARISON

Min_sup	execution time (min)		gain %
	Classical Apriori	Proposed Apriori	
1000	565	55	90.26%
1300	275	27	90.01%
1500	33	12	63.63%
2000	18	5	72.22%
3000	11	3	72.72%

The results of implementing of classical and ameliorated Apriori are presented in Table II and Fig. 4.

Table II shows that the improved Apriori reduces the execution time required for the original Apriori by 90.26% when the minimum support is equal to 1000 and by 72.72% when the Minsup is equal to 3000. So we notice that when the value of the minimum support decreases, the rate of reduction of time increases, which shows that our approach is effective.

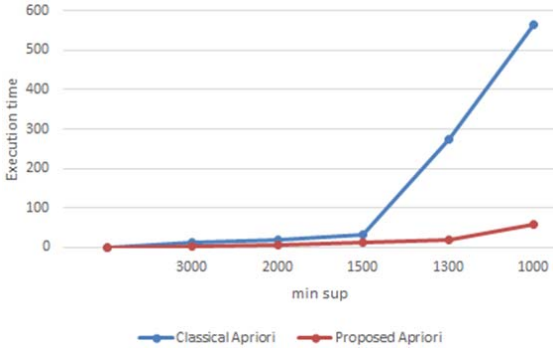

Fig. 4 Execution Time

As we see in Fig. 4, the time required for the proposed Apriori is less than the classical Apriori. Moreover, the execution time is reduced when the minimum support value decreases.

- *Number of Scanned Transactions*

In the classical Apriori, for each step of generating item sets candidates, all transactions are scanned to determine the frequent ones. However, our proposed algorithm will scan only the required transactions for generating frequent item sets as shown in Table III.

TABLE III
NUMBER OF SCANNED TRANSACTIONS

Min_sup	Classical Apriori	Proposed Apriori
1000	352644	212750
1300	352644	204459
1500	264483	188402
2000	264483	183783
3000	264483	175357

Fig. 5 Number of scan

As shown in Fig. 5 the number of scanned transactions is significantly reduced in the proposed algorithm, because all transactions are scanned in the classical Apriori while only the required transactions are used to determinate the frequent item sets.

Experiment 2

The main objective of this experiment is to compare the performances of our algorithm and the Apriori algorithm when the number of transactions becomes huge. So when the number of transactions grows the extraction of item sets become computational. The classical Apriori scan all transactions to generate the itemsets, but in the proposed algorithm only a set of transactions is scanned.

For this we use the transactional database Kosarak and we fix the minimum support to 20000 and the minimum confidence to 0.1.

TABLE IV
COMPARATIVE RESULT

number of transactions	Classical Apriori	Proposed Apriori
700 000	5418.614s = 90m	8304.6s = 138m
600 000	4449.973s = 74m	6564.38s = 109m
300 000	1948.29s = 32m	3525.34s = 58m
200 000	1093.67s =18m	3040.478s = 40m
100 000	171.028s = 2m	1413.346s = 24m

In Table IV, the proposed algorithm takes much less time than the classical Apriori algorithm. For example, a database that contains 700,000 transactions, the Apriori algorithm takes 138 minutes while the proposed algorithm takes only 90 minutes.

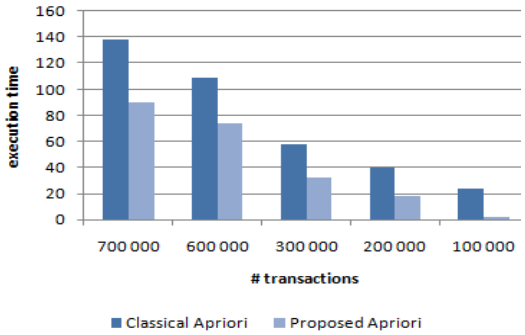

Fig. 6 Comparative result

In Fig. 6, we can see that the consumed time of the proposed algorithm is less than the time taken by the classical algorithm of Apriori, and the difference increases more and more when the size of the database increases.

VI. CONCLUSION

In this paper we studied the problem of association rule mining generation with NOSQL databases under a big data environment. The main goal of this paper is to improve the phase of discovery of frequent itemsets; we studied the Apriori algorithm and discussed its limits in order to propose a solution that reduces the time consumed for execution.

We used Hadoop Framework to experiment our work, and we choose NoSQL document-oriented databases. After the implementation of the two algorithms, many experiments were presented to compare the execution time of each algorithm, the obtained results show a significant gain in the proposed algorithm compared to the classical Apriori algorithm.

As future works, we can study the performances of our algorithm with other algorithms of extracting frequent itemsets.

REFERENCES

[1] Rakesh Agrawal, Tomasz Imielinski, and ArunSwami. Mining association rules between sets of items in large databases. In Proceedings of the 1993 ACM SIGMOD international conference on Management of data - SIGMOD '93. ACM Press, 1993.
[2] Idrees Al-Hashemi. Applying data mining technique over big data. PhD thesis, August 2013.
[3] AprilReeve. Managing data in motion: data integration bestpractice techniques and technologies. Morgan Kaufmann, 2013.
[4] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: a fexible data processing tool. Communications of the ACM, 53(1):72_77, 2010.
[5] Y. Djenouri, D. Djenouri, J. C. Lin, and A. Belhadi. Frequent itemset mining in big data with effective single scan algorithms. IEEE Access, 6:68013_68026, 2018.
[6] Cheikh Kacfah Emani, Nadine Cullot, and Christophe Nicolle. Understandable big data: a survey. Computer science review, 17:70_81, 2015.
[7] Pascal Hitzler and Krzysztof Janowicz. Linked Data, Big Data, and the 4th Paradigm. Semantic Web, 4(3):233_235, 2013.

[8] Brad Brown Jacques Bughin Richard Dobbs Charles Roxburgh James Manyika, Michael Chui and Angela Hung Byers. Big Data: The Next Frontier Forinnovation, Competition, And Productivity. Technical report, McKinsey Global Institute, 2011.
[9] Nawsher Khan, Mohammed Alsaqer, Habib Shah, Gran Badsha, Aftab Ahmad Abbasi, and Soulmaz Salehian. The 10 vs, issues and challenges of big data. In Proceedings of the 2018 International Conference on Big Data and Education - ICBDE '18. ACM Press, 2018.
[10] Ming-Yen Lin, Pei-Yu Lee, and Sue-Chen Hsueh. Apriori-based frequent itemset mining algorithms on MapReduce. In Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication - ICUIMC '12. ACM Press, 2012.
[11] Xueyan Lin. MR-apriori: association rules algorithm based on MapReduce. In 2014 IEEE 5th International Conference on Software Engineering and Service Science. IEEE, June 2014.
[12] Andreas Meier and Michael Kaufmann. NoSQL databases. In SQL & NoSQL Databases, pages 201_218. Springer Fachmedien Wiesbaden, 2019.
[13] https://nikinfotech.wordpress.com/category/bigdata/.
[14] Pasquier, Nicolas, et al. "Discovering frequent closed itemsets for association rules. International Conference on Database Theory. Springer, Berlin, Heidelberg, 1999.
[15] Rafael Peixoto, Hassan Thomas, Christophe Cruz, Aurélie Bertaux, and Nuno Silva. Semantic HMC for Business Intelligence using Cross-Referencing. In 14th International Conference on Informatics in Economy, Bucharest, Romania, April 2015.
[16] Agrawal Rakesh and Ramakrishman Srikant. Fast algorithms for mining association rules in large databases. In VLDB, 1994.
[17] Nataliya Shakhovska, Roman Kaminskyy, Eugen Zasoba, and Mykola Tsiutsiura. Association rules mining in big data. International Journal of Computing, 17(1):25_32, 2018.
[18] Michael Stonebraker. SQL databases v. NoSQL databases. Communications of the ACM, 53(4):10, apr 2010.
[19] VMWare. https://my.vmware.com/en/web/vmware/downloads.
[20] Eclipse. https://www.eclipse.org/.
[21] Frequent Itemset Mining Dataset Repository. http://fimi.uantwerpen.be/data/.
[22] https://hadoop.apache.org/.
[23] MongoBD. Available: http://www.mongodb.org/.
[24] https://whatsbigdata.be/mapreduce/
[25] Sudhakar Singh, Rakhi Garg, and PK Mishra. Review of apriori based algorithms on mapreduce framework. arXiv preprint arXiv:1702.06284, 2017.
[26] Davis, D. Patterson, Ethics of Big Data: Balancing Risk andInnovation, O'Reilly Media, 2012.

Sarra Senhadji, Associate professor since 2015, computer science department, faculty of mathematics and informatics, University of science and Technology Mohamed Boudiaf, Oran, Algeria. Interest in data mining, data replication and big data.

Imene Benzeguimi, Master in distributed information system in 2019, University of Science and Technology Mohamed Boudiaf of Oran, Algeria.

Zohra Yagoub, Master in distributed information system in 2019, University of Science and Technology Mohamed Boudiaf of Oran, Algeria.