An implementation of Eclat on spark

Article *in* International Journal of Computer Science and Information Security, · June 2017

CITATIONS
2

READS
1,298

3 authors:

Wael Mohamed
Helwan University
3 PUBLICATIONS **2** CITATIONS
SEE PROFILE

Manal A. Abdel-Fattah
Helwan University
37 PUBLICATIONS **113** CITATIONS
SEE PROFILE

Sayed Abdelgaber
Helwan University
29 PUBLICATIONS **47** CITATIONS
SEE PROFILE

An implementation of Eclat on spark

Wael Mohamed
Information Systems Department
Faculty of Computers and Information
Helwan University
Waelyousef148@gmail.com

Manal A. Abdel-fattah
Information Systems Department
Faculty of Computers and Information
Helwan University
MANAL_8@hotmail.com

Sayed Abd El-Gaber
Information Systems Department
Faculty of Computers and Information
Helwan University
Sgaber14@gmail.com

Abstract— **frequent pattern mining is an essential data mining technique. It aims to discover frequently co-occurring items, correlations and interesting information from data. Because we are in the era of big data and using of traditional frequent itemset mining for very large database will not scale well and generate high computational problems. Many algorithms have been proposed and implemented such as Apriori, Fp-growth and Eclat on Map Reduce framework. Map reduce has high disk I/O at each Map reduce job. Spark is another big data processing engine for processing large scale dataset. Spark is an in-memory framework.**

Previous studies have presented implementations for Apriori and Fp-growth on spark, however there is no implementation for Eclat algorithm on spark. In this paper, an implementation of Eclat algorithm on spark framework is proposed. Intensive experiments were conducted to know the efficiency and scalability of the proposed algorithm. Our studies show that the proposed algorithm can mine large dataset speedily. However the proposed algorithm may be unable to mine truly big data if tidset cannot fit into memory.

***Index Terms*—Big data, Eclat, frequent pattern mining, spark.**

I. INTRODUCTION

Data mining is the process of automatically discovering useful information in large data repositories. It is an integral part of knowledge discovery in databases (KDD)[1].It has attracted a great deal of attention from both research and commercial communities as a whole in recent years. Association rule mining is a very popular and effective method to extract meaningful information from large datasets. It tries to find associations between items in large datasets. It can be decomposed into subtasks finding frequent itemsets, and generating association rule. Finding frequent itemsets considered to be the base of association rule process. Several algorithms along with some improvement were proposed in order to extract frequent itemsets from data. This paper mainly focuses on Apriori, FP-Growth and Eclat.

Apriori uses a breadth-first strategy. It uses iterative approach where results of the previous iteration are used to find frequent pattern in the current iteration. It employs horizontal layout of data set. Apriori starts with finding singleton frequent items where item occurring more than minimum support. K-frequent itemsets are found using k-1 frequent itemsets. It uses level-wise manner.at each level of k ,the algorithm generate a huge number of candidates itemsets by joining the frequent itemsets produced at level k. each level k, Apriori scan database to computes supports of k-itemsets. For each level in Apriori, it needs to scan the database to calculate supports of itemsets[2].

Fp-Growth uses a depth-first strategy.it does not use paradigm of generate-and-test as in Apriori. It uses tree to represent the data set in a compact form, which is called Fp-tree .it needs only two passes over dataset. First scan of dataset is for counting support of each item in data set, generating frequent items and sorting frequent items in a descending order. Second scan of dataset is for constructing Fp-tree. It mines frequent itemsets by recursively divide-and-conquer approach[3].There exist prefix-tree-based algorithms namely cantree[4],CP-Tree[5],SPO-Tree[6] and ISPO-Tree[7]used for sequential mining. All these methods for construction tree share the same problem which is tree cannot fit to memory.

Eclat is a memory optimization of breadth-first approach. The Eclat algorithm partitions the lattice based on prefixes, which is called equivalence classes. There are different strategies for Eclat. In the earliest versions of Eclat, a breadth-first strategy was used and presented experimental results for only breadth-first strategy[8].Another approach for Eclat algorithm is depth-first as in dEclat and also presents experimental results for depth-first strategy[9].The Eclat version in[10]also mentioned that the equivalence class lattice can be traversed in either breadth-first or depth-first.it uses level-wise strategy in which all(k+1) candidates within a lattice partition are generated from frequent k-itemsets in level-wise as candidates generation step on Apriori. The tidlists are used for support counting. Support of itemset is calculated by intersecting tidlists of the subsets of the itemsets.

All the aforementioned algorithms work on sequential manner on small datasets. Due to big data era and datasets size are increased, therefore the efficiency of these algorithms drops. For handling the increasing in size of datasets, parallel implementations of algorithms were introduced. MapReduce and spark are processing engines of large-scale data processing. MapReduce and apache spark are used for implementing parallel version of frequent pattern mining algorithms.

Map reduce is a distributed parallel programming model developed by Google for processing large data sets in a parallel fashion[11].MapReduce is a fault-tolerant framework for distributed application. MapReduce works by breaking the processing into two phases .the map phase and the reduce phase .Each phase has key-value pairs as input and output, the types of which may be chosen by the programmer. The programmer also specifies two functions: the map function and the reduce function. The output from the map function is processed by the MapReduce framework before being sent to

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

the reduce function this phase is called shuffling and sorting stage. It is scalable platform, which allow parallel applications to run on large clusters. It is a fault tolerance platform. It is simplified programming model for distributed data processing on large clusters. One of disadvantages of map reduce is working on (key, value) pairs and all input datasets have to converted to (key, value).it is not suitable for iterative processing because of heavy disk I/O.

Spark is in-memory parallel computing model platform. Spark was created at AMPLabs in UC Berkeley as part of Berkeley Data Analytics Stack (BDAS) .Spark is a general engine for large-scale data processing[12]. It is tested to be up to 100× faster than Map Reduce. Spark is designed to overcome the disk I/O limitations and improve the performance of earlier systems such Map Reduce. It can run on Hadoop, standalone and cloud. It is suitable for iterative algorithms. It depends on a flexible DAG (directed acyclic graph).it has three types of data abstraction which are RDD, Dataframe and Dataset. RDD abstraction is supported from apache spark (1.0) to all news versions of apache spark. Dataframe abstraction is supported from apache spark (1.3). Dataset abstraction is supported from apache spark (1.6).RDD abstraction uses unstructured computations. Dataframe considered being a step in the way to transformation to structure computations. Dataset is an extension of Dataframe. Dataset tries to provide the best from RDD and Dataframe. Dataset provide Encoders. Encoders acts as interface between java virtual machine (JVM) objects and off-heap custom memory binary format data.

Previous studies have presented implementations for Apriori and Fp-growth on apache spark, however there is no implementation for Eclat algorithm on spark. Eclat-spark is an implementation of Eclat algorithm on spark. Intensive experiments were conducted to know the efficiency and scalability of Eclat-spark.

The paper is organized as follows. Section II presents a quick overview about the related work. The subsequent section highlights the proposed Eclat-spark algorithm. Experiments and results are outlined in Section IV. The conclusion of this research is offered together with suggestions for future research directions in the final section.

II. RELATED WORK

The existing frequent pattern mining algorithms can be classified into two categories:

- Frequent pattern mining algorithms on Map Reduce
- Frequent pattern mining algorithms on Spark

1. Frequent pattern mining algorithms on Map Reduce

1.1. Apriori on Map Reduce

The existing parallel version of Apriori on Map Reduce can be classified:

A. One phase ,two phase and k-phase:

One phase is an algorithm need only MapReduce job to generate all frequent k-itemsets. Two phase is algorithm need two MapReduce job to generate all frequent k-itemsets. K phase need k MapReduce job to generate all frequent k-itemsets.

B. Using combiner function

Combiner function used between mapper and reducer phases. Some algorithms used combiner function to reduce communication cost of transferring intermediate results between mapper and reducer.

C. Complete parallel algorithm (CPA) and incomplete parallel algorithm (ICPA)

Iteration of Apriori algorithm consists of two steps: generating candidates step, counting and pruning step. If algorithm parallelizes the two steps, this algorithm considered to be (CPA).if algorithm parallelize counting and pruning step only, this algorithm considered to be (ICPA).in CPA candidate generation realized under MapReduce framework.

D. Candidate generation step

Most algorithms make generation of candidates itemsets inside Mapper class in MapReduce.

Table I demonstrates the common characteristics between almost parallel versions of Apriori algorithm on MapReduce.

Apriori algorithm is iterative in nature. After each level, the results are stored in Hadoop Distributed File System (HDFS) to be used for the next level, which decrease its performance of Apriori on MapReduce due to high I/O time.

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

TABLE I COMPARSION OF PARALLEL VERSIONS OF APRIORI ALGORITHM ON MAP REDUCE

Algorithm proposed by	MapReduce	Using combiner	CPA or ICPA	Candidate generation step
[13]	One phase	Yes	ICPA	In Mapper
[14]	Two phase	No	ICPA	In Mapper
[15], [18], [21]	K phase	Yes	ICPA	In Mapper
[16]	K phase	No	ICPA	In Reducer
[17], [19], [22], [24], [25], [26]	K phase	No	ICPA	In Mapper
[18]	K phase	No	CPA	In Mapper

1.2. Fp-growth on Map Reduce

PFP (parallel fp-growth) is an implementation of Fp-growth algorithm on MapReduce[19].It consists of five steps. It needs three MapReduce job to find all frequent pattern. BFPF[20] is an implementation of Fp-growth on MapReduce. It consists of four steps. It needs two MapReduce job. PISPO[20] consist of five steps. It uses novel tree structure called improved single pass ordered tree (ISPO- tree).It needs two MapReduce job to find all frequent patterns.

Parallel versions of Fp-growth on MapReduce have the same problem which is data redundancy because of subtree overlapping.

1.3. Eclat on Map Reduce

There exist implementations of Eclat algorithm on MapReduce. MREclat[21] is an implementation of Eclat algorithm on MapReduce. It is two phase algorithm.it does not use combiner function. PEclat[22] is another implementation of Eclat on MapReduce. It use breadth-first and depth-first. Dist-Eclat[23] is another implementation of MapReduce .It needs three jobs of MapReduce. Parallel versions of Eclat have problem which is tidset may not fill well on memory.

2. Frequent pattern mining algorithms on spark

2.1. Apriori on spark

YAFIM[24] is an implementation of Apriori algorithm on spark RDD. It is consists of 2 phases. The first phase, for generating frequent items .the second phase, they use k-frequent itemsets to generate K+1-frequent Itemsets. It uses in-memory advantage for storing data set as RDDs on all workers of the cluster-Apriori[25] is another implementation of Apriori algorithm on spark RDD. It consists of three phases. It is similar to YAFIM .first phase is used for generating frequent items. Second phase is used for generating frequent 2- itemset. Third phase for generating frequent k-itemset.it differs from YAFIM in generating frequent 2-itemset. In second iteration, they remove candidate generation step and replacing hash tree by bloom filter. DFIMA [26] is another implementation of Apriori on spark. It uses matrix-based pruning strategy.it consists of two steps. In first step, they generate Boolean vectors for frequent item and produce all the frequent 2-itemset based on these Boolean vectors .in second step, they generate frequent k+1 itemsets from frequent k itemsets.

2.2. Fp-growth on spark

There exist an implementation of PFP[19] on apache spark ML.

2.3. Eclat on spark

There is no implementation of Eclat on spark. This research introduces an implementation of Eclat algorithm on spark

III. ECLAT ON SPARK

For implementing Eclat algorithm on apache spark. A breadth-first search and bottom-up approach are used to propose this algorithm. It scans dataset only one time.

Eclat-spark consists of two phases as in algorithm 1:

A. Phase one: finding singleton frequent itemset
B. Phase two : finding frequent k-itemsets

Algorithm 1 (Eclat - spark)
Input: Dataset D, minimum support
Output: All frequent itemsets
1: $L_1 \leftarrow$ finding-singleton-frequent-itemset (D, minimum support);
// for phase 1 pseudo code is presented in algorithm 1.1
2: for (k=2; $l_{k-1} \neq \{\}$; k++) {
3: $L_k \leftarrow$ finding-frequent k-itemsets (L_{k-1}, minimum support);
// for phase 2 and pseudo code is presented in algorithm 1.2
4 :}

1. Phase one: finding singleton frequent items

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

As illustrated on figure 1, transaction dataset is loaded into apache spark using spark context and store dataset as RDD using textfile() method to make good use of cluster memory and achieve the following sub phases as in algorithm 1.1 :

A. Transformation dataset from horizontal to vertical
B. Generate singleton frequent items

Table III depicts functions operations for phase 1

Algorithm 1.1 for phase 1 (finding singleton frequent items)

Input: Dataset D, minimum support

1: For each transaction in RDD

2: **FlatMapToPair** (key =tid, value =Transaction T)

3: **For each** item I in transaction

4: Output (I, T.tid);

5: **end for each**

6: **end FlatMapToPair**

7: **end for each**

8: **aggregateByKey** (key = I, value = tid)

9: **output** (I, tidset of I);

10: **end aggregateByKey**

11: **filter** (I, tidset of I)

12: if support of I > minimum support

13: **output** ((I, tidset of I));

14: **end filter**

TABLE II: TRANSACTION DATASET

Tid	Items in transaction
1	a, b, c, d
2	a ,c
3	b , d

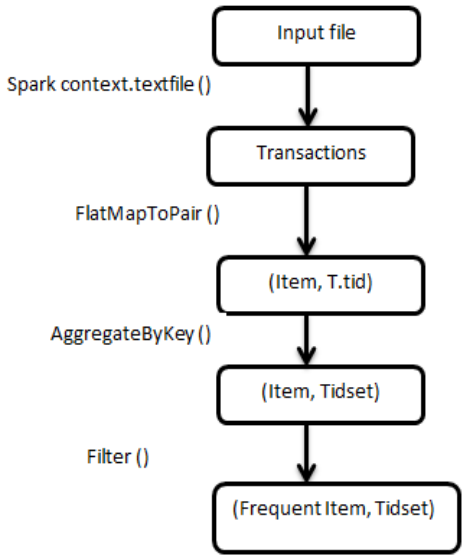

Figure 1 linage graph for first phase

TABLE III FUNCTIONS IN PHASE 1

Function name	Purpose
FlatMapToPair	Read each transaction and generates pairs for each item in transaction in the form (item ,tid)
aggregateByKey	Aggregate values for each key (item ,tidset)
Filter	Compare each key with minimum support for finding singleton frequent items if tidset greater than or equal to minimum support

For example, if we have dataset as in Table II. An invocation of FlatMapToPair function will take the first transaction and output the following pairs {(a, 1), (b, 1), (c, 1), (d, 1)}. Another invocation of FlatMapToPair function will read the second transaction and output the following pairs {(a, 2), (c, 2)} .another invocation of FlatMapToPair read the third transaction and output the following pairs {(b, 3), (d, 3)}. An invocation of aggregateByKey will get tidset for each key like tidset of a will be {1,2} and tidset of b will be {1,3}. An invocation of filter function will count support each key and yields only keys with support greater than or equal to minimum support.

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

2. Phase two: finding frequent k-itemsets

As illustrated on figure 2, in this phase: firstly, we use frequent (k-1)-itemsets, which are produced in previous iteration, for generating candidates for k-itemsets c_k by using candidate generation as in Apriori (joining only). Table IV depicts functions operations for phase 2.

Algorithm 1.2 for phase 2 (finding frequent k-itemsets)

Input: $L_{k-1,}$ minimum support

Output: L_k

1: load frequent items L_{k-1};

2: $c_k \leftarrow$candidate-generation (L_{k-1});

// by using joining step only

1: **FlatMapToPair** (Itemset I, tidset)

2: load c_k;

3: **for each** candidate C in C_k

4: **if** I is a subset of C and I share prefix (k-2) with c

5: **output** (c, tidset of I));

6**: end for each**

7: end FlatMapToPair

8: reduceByKey (key= candidate c, value= list of tidset)

/* in fact, **reduceByKey** will receive only the covers of two of its subsets that together give the candidate itself ***/**

9: tidset of y1 ← the first item is list of tidset

10: tidset of y2 ← the second item is list of tidset

11: tidset of c ← tidset of y1 ∩ tidset of y2

12: **output** (c, tidset of c);

13: **end reduceByKey**

14: **filter** (c, tidset of c)

15: if support of c > minimum support

16: **output** (c, tidset of c);

17: **end filter**

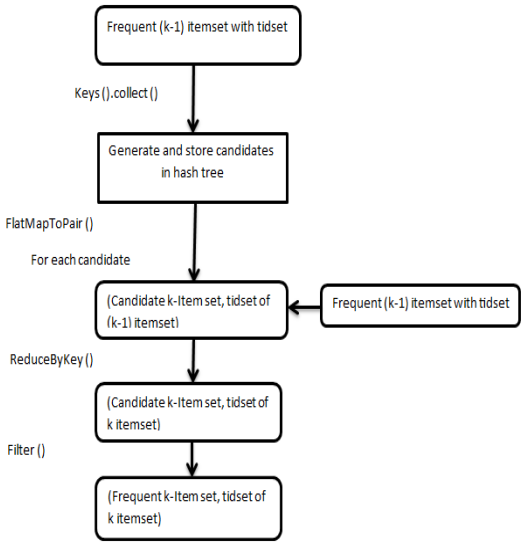

Figure 2 linage graph for phase two

TABLE IV FUNCTIONS IN PHASE 2

Function name	Purpose
FlatMapToPair	For each frequent itemset y in the previous iteration (k-1), compare it with each candidate c in c_k and yields (c , tidset(y)) if y is a subset of c and share the same prefix (k-2) with c .
reduceByKey	Receive candidate c as key , tidsets as value (ex, y_1,y_2) and make intersection between (tidset y_1, tidset y_2) and yields (c ,tidset of c)
Filter	Compute support of c and compare it with minimum support for finding frequent k-itemset if tidset greater than or equal to minimum support

For example, if we have frequent itemsets {ab, ac, ad, bc, bd}, then candidates will be {abc ,abd , acd , bcd}. FlatMapToPair function will check if itemset and candidate have the same (k-2) prefix and itemset is a subset of candidate. For example , FlatMapToPair function take ab as input and outputs{ (abc , tidset of ab) , (abd , tidset of ab) }. FlatMapToPair function take ac as input and output { (abc , tidset of ac) , (acd , tidset of ac) }. **ReduceByKey** takes abc as key, tidset of ab and tidset of ac as value. **ReduceByKey** also make intersection between them, store result as tidset of abc and output (abc, tidset of abc). **Filter** function count support of abc and output (abc, tidset of abc) if support > minimum support.

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

IV. EXPERIMENTS AND RESULTS

In this section, we evaluate the performance of Eclat-spark. All experiments were conducted on a single node cluster using Cloudera VM with 8GB RAM and processor Intel core i7. All experiments were executed three times and average results were taken.

A. Datasets

Four benchmarks datasets were used to test performances of the algorithm. Characteristics Of these datasets are illustrated in Table V.

TABLE V DATASETS

Dataset	Number of Items	Number of Transactions
Chess	75	3196
Mushroom	119	8124
T10I14D100K	870	100000
BMSWebView2	3340	77512

B. Performance analysis

All results from applying the Eclat-spark algorithm on each dataset is illustrated in figure 3.1 figure 3.2, figure 3.3 and figure 3.4

For Chess as in figure 3.2 with minimum support 85 % Eclat-spark takes 28 seconds. With minimum support 75 % Eclat-spark takes 280 seconds.

For Mushroom as in figure 3.4 with minimum support 80% Eclat-spark takes 8 seconds. With minimum support 60 % Eclat-spark takes 10 seconds. With minimum support 35 % Eclat-spark takes 25 seconds.

For T10I14D100K as in figure 3.1 with minimum support 1%, Eclat-spark takes 110 seconds. With minimum support 0.50 % Eclat-spark takes 210 seconds. With minimum support 0.25 % Eclat-spark takes 280 seconds. With minimum support 0.15 % Eclat-spark takes 370 seconds.

For Bmswebview-2 as in figure 3.3, when minimum support is 1%, Eclat-spark takes 10 seconds. With minimum support 0.1% Eclat-spark takes 320 seconds.

From previous results, when minimum support is decreased and the tidset cannot fit on memory, Eclat-spark spill data from memory to disk which leads to increasing in time.

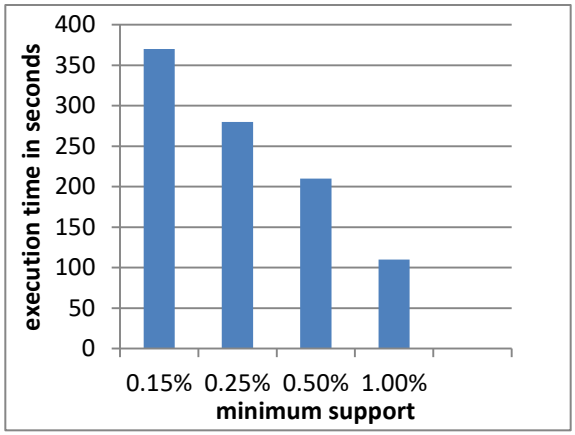

Figure 3.1 T10I14D100K

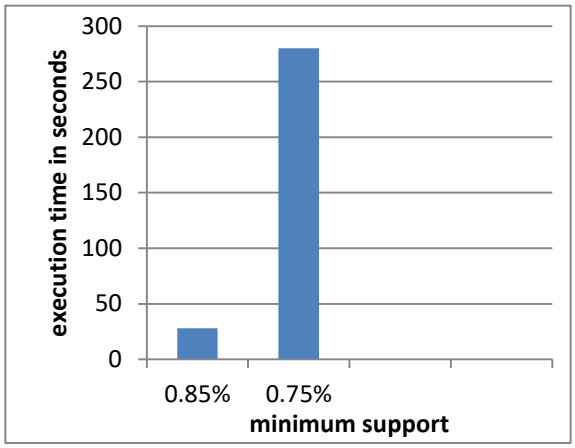

Figure 3.2 chess

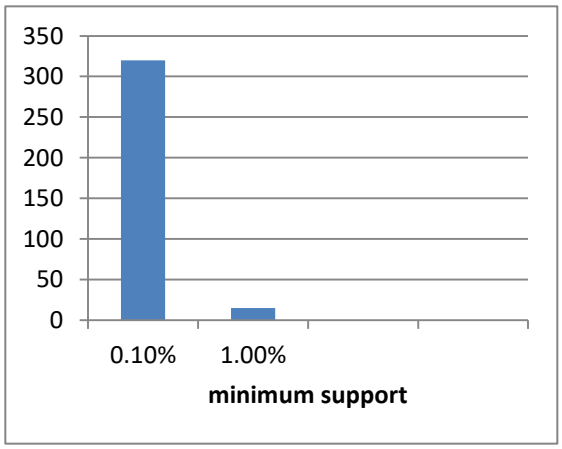

Figure 3.3 Bmswebview- 2

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

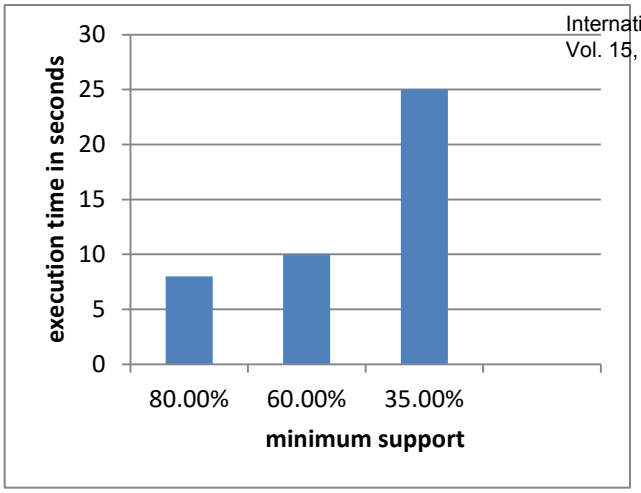

Figure 3.4 mushroom

v. CONCLUSION AND FUTURE WORK

In this paper, an implementation for Eclat algorithm on spark is proposed. The proposed algorithm scans dataset only one time. The proposed algorithm used breadth-first strategy .The proposed algorithm consists of two phases. In phase 1, transaction dataset is loaded into RDD and generating singleton frequent items.in phase 2, generating frequent k-itemsets. It can be used for mining large dataset. The proposed algorithm may be unable to mine truly big data if tidset cannot fit into memory. We plan to use hybrid method by using Apriori for first levels and then switch to use Eclat for the rest of levels to solve the problem of tidset cannot fit into memory. We plan to compare performance of hybrid , Eclat-spark and conducting experiments on multimode cluster with real big datasets to know scalability of each algorithm. We also plan to use apache flink because it is native iterative processing engine.

REFERENCES

[1] K.-P. Pang-Ning, Steinbach, *Introduction to Data Mining*. 2005.

[2] R. Agrawal, "Fast Algorithms for Mining Association Rules 1 Introduction," in *20th int. conf. very large data bases, VLDB*, 1994.

[3] J. Han, J. Pei, and Y. Yin, "Frequent Pattern Tree : Design and Construction," *ACM Sigmod Rec.*, vol. 29, pp. 1–12, 2000.

[4] C. K. S. Leung, Q. I. Khan, Z. Li, and T. Hoque, "CanTree: A canonical-order tree for incremental frequent-pattern mining," *Knowl. Inf. Syst.*, vol. 11, no. 3, pp. 287–311, 2007.

[5] S. K. Tanbeer, C. F. Ahmed, B. S. Jeong, and Y. K. Lee, "CP-tree: A tree structure for single-pass frequent pattern mining," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5012 LNAI, pp. 1022–1027, 2008.

[6] Y. S. Koh and G. Dobbie, "SPO-Tree : Efficient Single Pass Ordered," in *International Conference on Data Warehousing and Knowledge Discovery*, 2011, pp. 265–276.

[7] X. Jiang and G. Sun, "MapReduce-based frequent itemset mining for analysis of electronic evidence," *2013 8th Int. Work. Syst. Approaches to Digit. Forensics Eng.*, pp. 1–6, 2013.

[8] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li, "New Algorithms for Fast Discovery of Association Rules," *3rd Intl Conf Knowl. Discov. Data Min.*, vol. 20, no. 651, pp. 283–286, 1997.

[9] M. J. Zaki and K. Gouda, "Fast vertical mining using diffsets," *Proc. ninth ACM SIGKDD Int. Conf. Knowl. Discov. data Min. - KDD '03*, p. 326, 2003.

[10] M. J. Zaki, "Scalable algorithms for association mining," *IEEE Trans. Knowl. Data Eng.*, vol. 12, no. 3, pp. 372–390, 2000.

[11] J. Dean and S. Ghemawat, "MapReduce: Simplied Data Processing on Large Clusters," *Proc. 6th Symp. Oper. Syst. Des. Implement.*, pp. 137–149, 2004.

[12] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark : Cluster Computing with Working Sets," *HotCloud'10 Proc. 2nd USENIX Conf. Hot Top. cloud Comput.*, p. 10, 2010.

[13] M. Zhang, "Cloud Computing," in *international conference on Business Computing and Global Informalization*, 2011, pp. 475–478.

[14] O. Yahya, O. Hegazy, and E. Ezat, "An efficient implementation of Apriori algorithm based on Hadoop-Mapreduce model," *Proc. of the*, vol. 12, no. December, pp. 59–67, 2012.

[15] M.-Y. Lin, P.-Y. Lee, and S.-C. Hsueh, "Apriori-based frequent itemset mining algorithms on MapReduce," *Proc. 6th Int. Conf. Ubiquitous Inf. Manag. Commun.*, no. ACM, p. 76, 2012.

[16] F. Kovacs and J. Illes, "Frequent itemset mining on hadoop," *...), 2013 IEEE 9th Int. Conf.*, pp. 241–245, 2013.

[17] X. Y. Yang, Z. Liu, and Y. Fu, "MapReduce as a programming model for association rules algorithm on Hadoop," *Proc. - 3rd Int. Conf. Inf. Sci. Interact. Sci. ICIS 2010*, pp. 99–102, 2010.

[18] X. Zhou and Y. Huang, "An improved parallel association rules algorithm based on MapReduce framework for big data," *2014 11th Int. Conf. Fuzzy Syst. Knowl. Discov. FSKD 2014*, pp. 284–288, 2014.

[19] H. Li, Y. Wang, D. Zhang, M. Zhang, and E. Chang, "PFP : Parallel FP-Growth for Query Recommendation," *Group*, pp. 107–114, 2008.

[20] L. Zhou, Z. Zhong, J. Chang, J. Li, J. Zhexue, and S. Feng, "Balanced parallel FP-growth with mapreduce," *Proc. - 2010 IEEE Youth Conf. Information, Comput. Telecommun. YC-ICT 2010*, pp. 243–246, 2010.

[21] Z. Zhang, G. Ji, and M. Tang, "MREclat: An Algorithm for Parallel Mining Frequent Itemsets," *Adv. Cloud Big Data (CBD), ...*, pp. 177–180, 2013.

[22] J. Liu, Y. Wu, Q. Zhou, B. C. M. Fung, F. Chen, and B. Yu, "Parallel eclat for opportunistic mining of frequent itemsets," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2015.

[23] S. Moens, E. Aksehirli, and B. Goethals, "Frequent itemset mining for big data," *Big Data, 2013 IEEE ...*, no. 1, pp. 111–118, 2013.

[24] H. Qiu, R. Gu, C. Yuan, and Y. Huang, "YAFIM: A parallel frequent itemset mining algorithm with spark," *Proc. Int. Parallel Distrib. Process. Symp. IPDPS*, pp. 1664–1671, 2014.

[25] S. Rathee, M. Kaul, and A. Kashyap, "R-Apriori: An Efficient Apriori based Algorithm on Spark," *Acm*, pp. 27–34, 2015.

[26] F. Zhang, M. Liu, F. Gui, W. Shen, A. Shami, and Y. Ma, "A distributed frequent itemset mining algorithm using spark for big data analytics," *Cluster Comput.*, vol. 18, no. 4, pp. 1493–1501, 2015.

https://sites.google.com/site/ijcsis/
ISSN 1947-5500