Current Journal of Applied Science and Technology

26(1): 1-14, 2018; Article no.CJAST.39546
Previously known as British Journal of Applied Science & Technology
ISSN: 2231-0843, NLM ID: 101664541

Efficient Association Rule Mining for Retrieving Frequent Itemsets in Big Data Sets

Chandaka Babi[1], Mandapati Venkateswara Rao[1] and Vedula Venkateswara Rao[2*]

[1]Gandhi Institute of Technology and Management (GITAM University), Visakhapatnam, India.
[2]Sri Vasavi Engineering College, Tadepalligudem, India.

Authors' contributions

This work was carried out in collaboration between all authors. Author CB designed the study, performed the problem analysis, wrote the algorithm, and wrote the first draft of the manuscript. Author MVR managed the literature searches, algorithm design and author VVR managed the analyses of the study, implementation and analysis of results. All authors read and approved the final manuscript.

Article Information

DOI: 10.9734/CJAST/2018/39546
Editor(s):
(1) Hui Li, Associate Professor, School of Economics and Management, Zhejiang Normal University, China.
Reviewers:
(1) Parijit Kedia, Switzerland.
(2) Manish Mahajan, CGC College of Engineering, India.
Complete Peer review History: http://www.sciencedomain.org/review-history/23209

Original Research Article

Received 22nd November 2017
Accepted 2nd February 2018
Published 15th February 2018

ABSTRACT

Information retrieval and decision-making demand a scalable and efficient methodto process and extract relevant information from Big Data. Data mining is a refined analysis of a large quantity of data to determine new information in the outline of patterns, trends, and relations. With the spread of the World Wide Web, the quantity of data stored and made available electronically has increased enormously, and methods to retrieve information from such big data have gained immense significance for both business and scientific research communities. Frequent Itemset Mining is one of the most extensively applied procedures to retrieve useful information from data. However, when this method is applied to Big Data, the combinatorial outburst of candidate itemsets has become a challenge. Recent developments in the area of parallel programming have offered outstanding tools to overcome this problem. Nevertheless, these tools have their own technical drawbacks, e.g. unbiased data sharing and inter-communication costs. In our study, we examine the applicability of Frequent Itemset Mining in the MapReduce framework. We introduce a new method for

**Corresponding author: E-mail: venkatvedula2017@gmail.com, venkatvedula2012@gmail.com;*

extracting large datasets: Big-Frequent-ItemsetMining. This method is optimized to run on extremely large datasets. Our approach is similar to FP-growth but uses a different data structure that is based on an algebraic topology. In this study, we demonstrate the scalability of our techniques.

Keywords: Data mining; frequent itemsets; association rules; big data sets; frequent pattern mining; map reduce.

1. INTRODUCTION

With the current progress in technology, science, user habits, and businesses, a massive quantity of data is being produced and stored. Therefore, effective analysis of big data has become more significant for both businesses and academics. Frequent Itemset Mining is an important data analysis and mining method [1]. This method retrieves information from databases on the basis of the frequency of occurrence of items in the data, i.e., an event or a set of events, with respect to a user-specified minimum frequency threshold. Many methods have been developed to extract databases based on the frequency of events [2-4]. Although these methods work well in practice for archetypal datasets, they are not suitable for extremely Big Data. Therefore, applying Frequent Itemset Mining to large databases is difficult. Because very large databases are not generally stored in main memory, algorithms based on level-wise breadth-first search is appropriate for data mining. The Apriori algorithm [2] is one such scheme, where frequency accumulation is achieved by reading the dataset repeatedly for each size of the candidate itemsets [5]. Unfortunately, the extremely large memory required for managing the large number of candidate itemsets makes the Apriori-based schemes incompetent to be applied on single machines. Recent approaches tend to keep the output and runtime under control by incrementing the minimum frequency threshold, thereby reducing the number of candidate and frequent itemsets. However, research on recommendation systems has revealed that itemsets with lower frequencies are more desirable [6]. Hence, there is still a need for methods that can retrieve data on the basis of low frequency thresholds in Big Data. Parallel programming is gaining importance to deal with massive amounts of data [7]. Parallel programming architectures and algorithms can be classified into two major subcategories: shared memory and distributed (shared nothing). In shared memory systems, all processing units use a shared memory area in tandem. In contrast, the distributed systems consist of processors that have their individual internal memories and communicate with each other by transmitting data through a network [8]. In general, it is easier to adapt algorithms to the shared-memory architecture; however, it is typically not sufficiently scalable [8]. Through Google's MapReduce framework [9], which simplifies the program for distributed data dispensation, and the Apache Hadoop [10], which renders the framework accessible to everyone, distributed programming has been gaining increasing acceptance [11]. In addition, current viable and non-viable systems and services advance the usability and accessibility for anyone. In this study, we establish a new algorithm that exploits the MapReduce framework to process Big Data. Frequent Itemset Mining for Big Data is optimized by using a hybrid algorithm developed on the MapReduce framework [12,13].

2. RELATED WORK

Since the beginning of research on data mining, parallel processing methods have been in focus [14]. There exist many parallel mining methods as well as techniques to parallelize existing sequential mining techniques. However, only a few algorithms have been adapted for use in the MapReduce framework. In this section, we provide an overview of the data mining algorithms.

2.1 Overview of Association Analysis

Among the various data mining techniques, association analysis is a popular and well-researched method and it extracts interesting associations and relationships among variables in large databases. For a pattern to be interesting, it should be logical and actionable. Association rule learning basically involves determining relationships among attribute values that occur frequently together in a dataset and representing them in the form of association rules. Association rules do not indicate causality, but suggest strong co-occurrence relationships that can be further investigated as associated factors. Two important metrics in association analysis are support and confidence [15,16].

Support indicates how often a rule is applicable to a specific dataset and can be used to eliminate uninteresting rules, such as those that occur simply by chance [9,17]. Confidence measures the reliability of the inference made by a rule; for instance, X -> Y measures how frequently items or attributes in Y appear in transactions that contain X. Minimum support and confidence thresholds are selected for assessing the association rules extracted from the data. An itemset is frequent if its support is greater than or equal to the minimum support value. One important issue with mining association rules in large datasets is the fact that it can be computationally expensive depending on the algorithm used. A brute-force approach for determining patterns from data involves computing the support and confidence for every possible rule. As the number of rules that can be obtained from a dataset increases exponentially with the number of items in that set, this brute-force approach becomes prohibitively expensive. This approach also results in wasted transactions because many of the rules that fall below the selected minimum support and confidence levels would be discarded.

2.2 Apriori and FP-Growth Algorithms

Many algorithms for generating association rules have been presented over time, such as the Apriori and FP-Growth algorithms [3,18,19]. A common strategy that these algorithms implement, in terms of performance improvement, is to decompose the problem into two subtasks [20-22]:

a) Frequent itemset generation, accomplished by reducing either the number of (i) candidate itemsets on the basis of the support measure, as in the Apriori algorithm, or (ii) comparisons, as in the FP-Growth algorithm;
b) Rule generation, which first excludes rules that have empty antecedents or consequents and then checks whether, after splitting itemset Y into two non-empty subsets (X and Y – X), rule X -> Y – X satisfies the confidence threshold. Y – X in this case is known as the rule consequent. Rule generation does not require any additional passes over the dataset.

2.3 Data Mining Algorithms on MapReduce

Lin et al. proposed three algorithms that are adaptations of Apriori on MapReduce [23]. These algorithms distribute the dataset to mappers and perform the frequency counting step in parallel. Single-Pass Counting (SPC) utilizes a MapReduce phase for each candidate generation and frequency counting step. Fixed Passes Combined-Counting generates candidates with n different lengths after p phases, where n and p are given as parameters, and counts their frequencies in a single database scan. Dynamic Passes Counting is similar to FPC, but n and p are determined dynamically at each phase by the number of generated candidates [24,25]. The PApriori algorithm proposed by Li et al. [26] works similar to SPC, although they differ in minor implementation details [27-29]. MRApriori [30] iteratively switches between vertical and horizontal database layouts for mining all frequent itemsets. At each iteration, the database is partitioned and distributed across mappers for frequency counting. Parallel FP-Growth [31] is a parallel version of the well-known FP-Growth [32]. PFP groups the items and distributes their conditional databases to the mappers [33]. Each mapper builds its corresponding FP-tree and mines it independently. The PARMA algorithm by Riondato et al. [34] determines approximate collections of frequent itemsets.

2.4 Hadoop Framework

Hadoop is a software framework for distributed processing of large datasets across large clusters of computers. The HadoopDistributed File System (HDFS) was designed in the project NUTCH to serve as the storage mechanism. Input data are split into various chunks of default size 64 MB and stored in the HDFS [35,36].

Hadoop consists of two main layers:

- Distributed file system (HDFS)
- Execution engine (MapReduce)

Fig. 1 shows the Hadoop architecture.

The HDFS has a block-oriented architecture. Each block, called the data node, contains actual data, has a fixed size, and is stored in the Hadoop cluster. The data nodes are stored in different machines at different clusters and a dataset is processed in the same cluster where it is stored. The HDFS follows the master–slave architecture. Each Hadoop cluster contains a single name node, which is the master node, and data nodes, which are the slave nodes. The primary communication mechanism between the

name node and data node is called heartbeat. In every three seconds, a heartbeat containing the block report and list of blocks in the data node is sent to the name node from the data node. If a heartbeat is not received, the name node will create a replica of the data node. Fig. 2 describes the HDFS architecture [37].

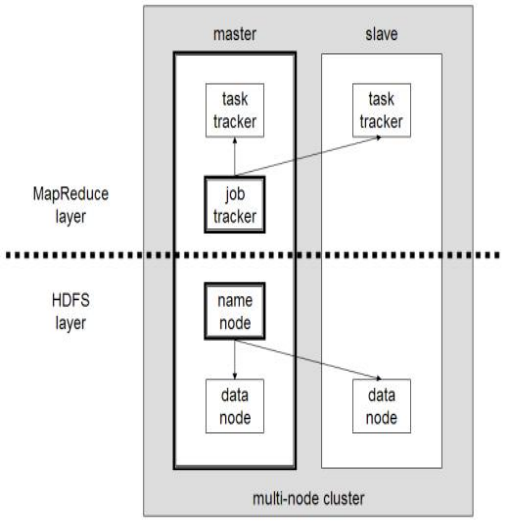

Fig. 1. Hadoop architecture

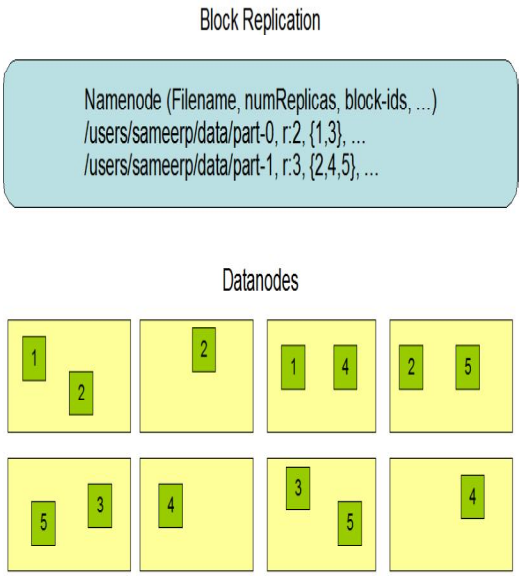

Fig. 2. HDFS architecture

2.5 MapReduce

The MapReduce framework was introduced by Google in 2004 to retrieve data efficiently from large spectrum of data. MapReduce processes the data as a key-value pair. The MapReduce operation is performed in four phases. The first phase is the mapper phase, where the data are collected from the HDFS stored in different clusters [38,39]. The output from the mapper phase is the intermediate results, which are sent to the next phase for passing them on to the reducer. The second phase is the shuffle phase; here, the intermediate results are shuffled so that the results from different mappers are mixed. The third phase is the sort phase. Here, the shuffled intermediate results are sorted on the basis of the key value such that the contents with the same key value are brought together. The sorted contents can be easily passed to the reducer for processing. The last phase is the reducer phase, where the sorted contents are processed to yield the significant data. The jobs in the Hadoop cluster are performed by the task tracker. When a job is scheduled, the job tracker will assign the job to the task tracker. It will then proceed to the job execution and the output is produced. Thus, the large amount of data is processed into useful contents. Fig. 3 explains the layers involved in MapReduce.

The functionality of MapReduce can be explained by Fig. 4.

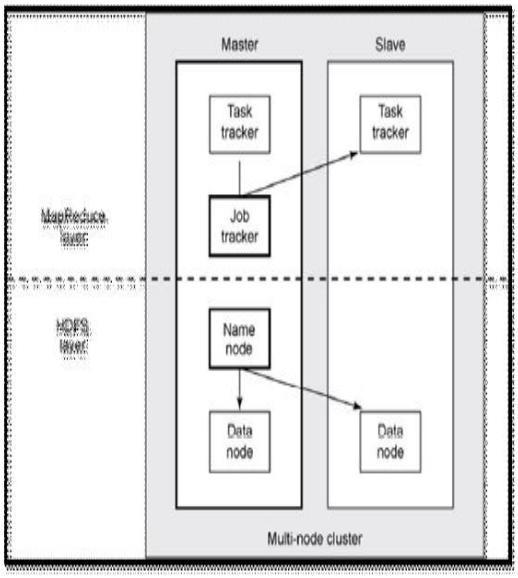

Fig. 3. Layers in the MapReduce framework

3. FREQUENT ITEMSET MINING ON MAPREDUCE

We propose two new methods for mining frequent itemsets in parallel on the MapReduce

framework, where frequency thresholds can be set low. We introduce a second method, which is a hybrid method that first uses an Apriori-based method to extract frequent itemsets of length k and subsequently switches to Eclat when the projected databases fit into memory. First, mining for k-FIs can already be infeasible [40]. Indeed, in the worst case, one mapper needs the complete dataset to construct all 2-FI pairs. Considering Big Data, the tid-list of even a single item may not fit into memory. Second, most of the mappers require the whole dataset in memory to mine the subtrees [41-43].

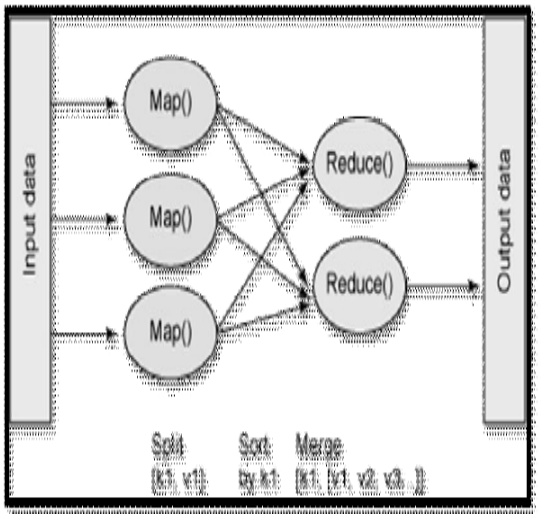

Fig. 4. Functionality of MapReduce

The following are the steps involved in the proposed algorithm

1. **Generating k-FIs**: Frequency Itemset Mining for Big Data covers the problem of large tid-lists by generating k-FIs using the breadth-first search method. This can be achieved by adapting the Word Counting problem for documents, i.e. each mapper receives part of the database (a document) and reports the items/itemsets (the words) for which we want to know the support (the count). A reducer combines all local frequencies and reports only the globally frequent items/itemsets. These frequent itemsets can be redistributed to all mappers to act as candidates for the next step of breadth-first search. These steps are repeated k times to obtain the set of k-FIs.
2. **Finding Potential Extensions:** After computing the prefixes, the next step is computing the possible extensions, i.e. obtaining tid-lists for (k+1)-FIs. This can be performed similar to how Word Counting is performed; however, in computing possible extensions, instead of local support counts, the local tid-lists are reported. A reducer combines the local tid-lists from all mappers to a single global tid-list and assigns complete prefix groups to all the mappers.
3. **Subtree Mining:** Finally, the mappers work on individual prefix groups. A prefix group defines a conditional database that completely fits into memory. The mining part then utilizes diffsets to mine the conditional database for frequent itemsets using depth-first search. The iterative process is continued until a set of k-FIs that are small enough is reached.

In our methods, frequent itemsets are mined in step 3 by the mappers and then communicated to the reducer. To reduce network traffic, we encoded the mined itemsets using a compressed trie string representation for each batch of patterns.

The new algorithm for mining frequent itemsets in Big Data sets is a modification of the original FPTree algorithm and is described as follows.

```
Algorithm BigFPTree ():
Input: Big Data Set of Transactions
          Support
Output: Frequent Patterns
Begin
ConstructHeaderTable ()
FindOneFrequentItemSets ()
ProcessTransactionSets ()
End

Algorithm ConstructHeaderTable ()
Begin
Scan Dataset
Count support of each item
Construct Header Table (TH) by using Items and
their support
(Header Table Consists 3 fields name, support
and link)
Link refers to all nodes of an item on Tree
End

Algorithm FindOneFrequentItemSets ()
Begin
While (TH Not empty)
Do
```

Remove Items with support less than min support from TH
Sort the TH based on support in descending order
Done
End

Algorithm ProcessTransactionSets ()
Begin
Scan the Data set to create Transaction Item Sets (TIS)
Remove the non-frequent Items from TIS
Sort TIS by order of Items in TH
Construct Tree (T) with Ordered Item Sets
Add new nodes to link field of TH
ProcessItem ()
End

Algorithm ProcessItem ()
Begin
Add Item Q into Base Item (BI)
In TH, Q.link contains all Nodes in Tree T whose Item is Q
Read all Items from Node N_i i= 1 to k to root of T
Create SubHeaderTable (SHT) with items and support
While (SHT Not empty)
Do
Remove items from SHT with support less than min support
Sort SHT on support in descending order
Done
Read all Items from N_i to root
Remove non-local frequent Item Sets
Sort Item Sets by SHT
Construct new Sub Tree with sorted Item Sets
Add s to support keep all new nodes in link of SHT
End

The flowchart presented in Fig.5 explains the process of finding frequent itemsets in Big Data sets

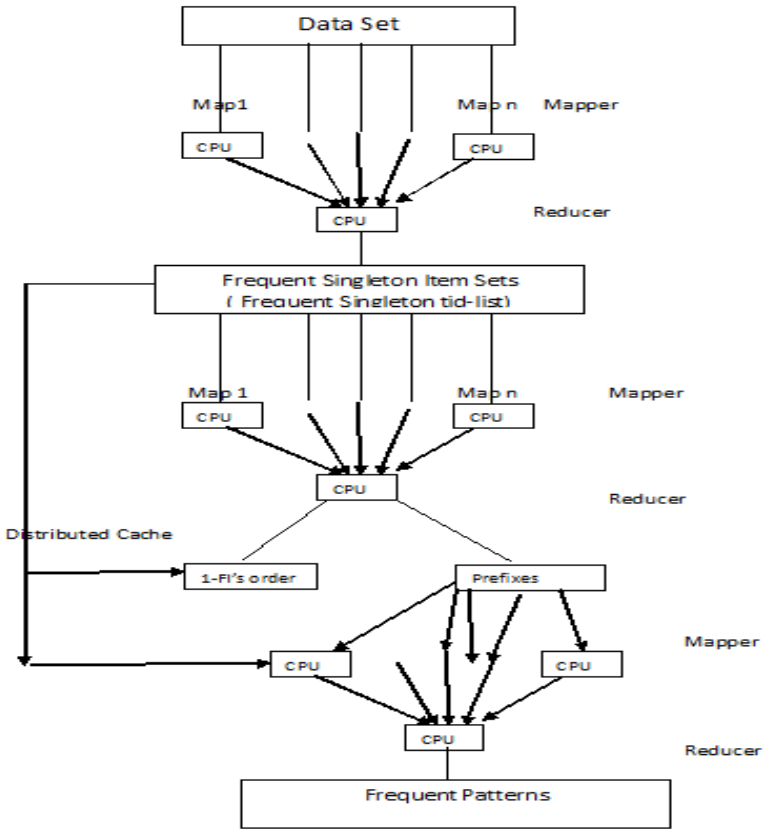

Fig. 5. Process for finding frequent itemsets in Big Data sets

4. EXPERIMENT RESULTS AND ANALYSIS

We analyse the running time of algorithms on both synthetic and real datasets. Our experiments suggest independence of the assignment method: using long prefixes results in a better balancing of the load. Although generating long prefixes requires additional initial computation, for large databases, this computation is negligible compared to the entire mining process. Here, we consider two datasets of different sizes and evaluate the generation of association rules based on different parameters.

Fig. 6. shows the relation between support and confidence in generating association rules.

Fig. 7 shows the mapper and reducer execution.

Figs. 8 and 9 show variation in the execution time for different numbers of association rules for dataset1 and dataset2, respectively.

Figs. 10 and 11 show the number of association rules generated for dataset1 and dataset2, respectively.

Figs.12 and 13 show the efficiency of BigFM and Apriori algorithms for dataset1 and dataset2, respectively, with different minimum support and minimum confidence values.

Table 1 shows the results of BigFM and Apriori on Big Data sets

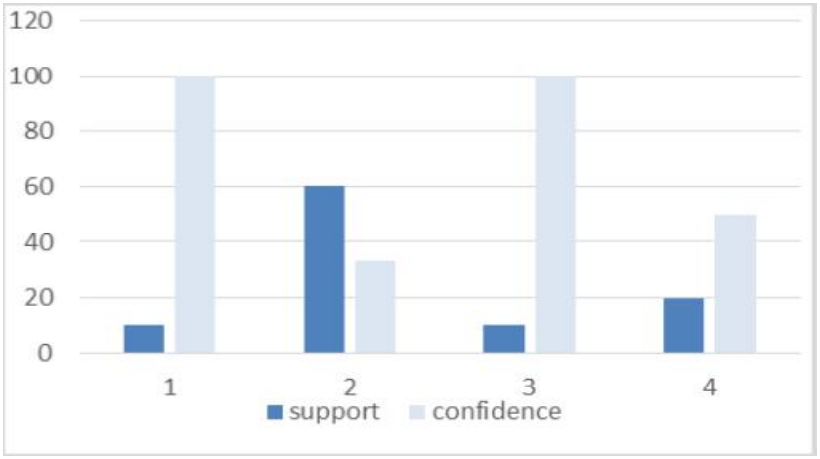

Fig. 6. Support vs. confidence

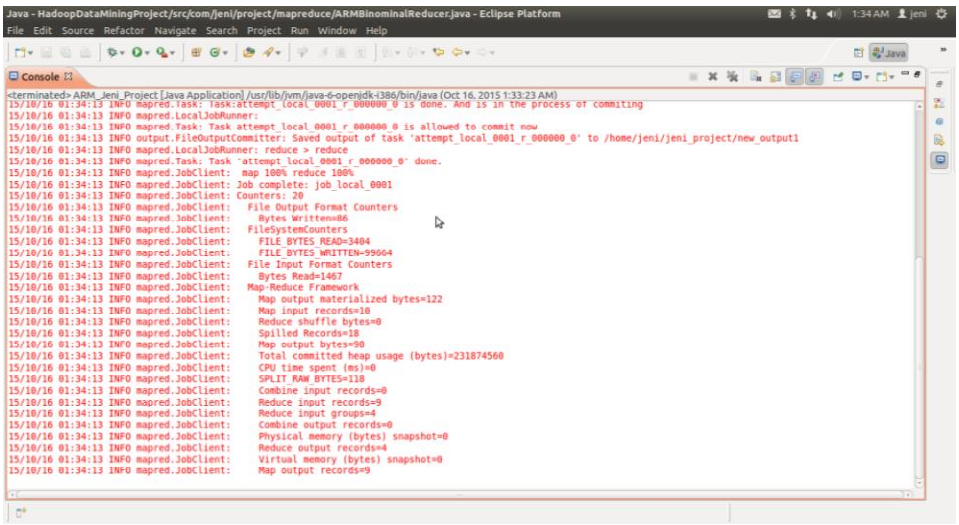

Fig. 7. Mapper and reducer execution

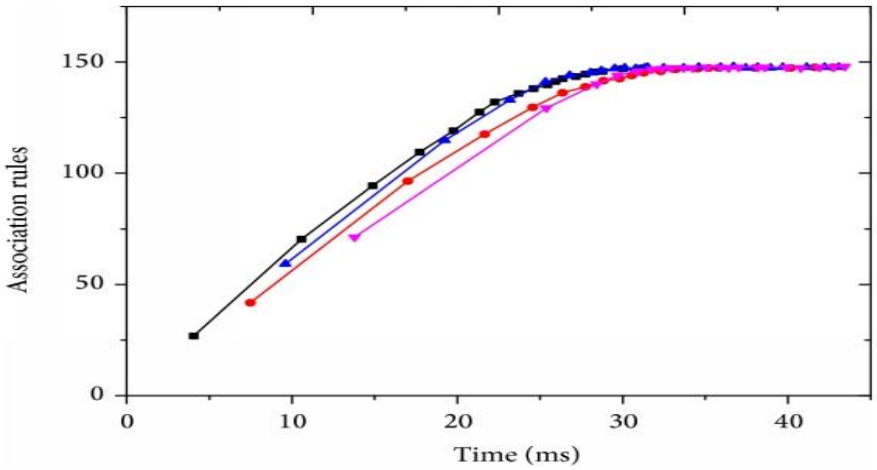

Fig. 8. Number of association rules and time taken for execution for dataset1

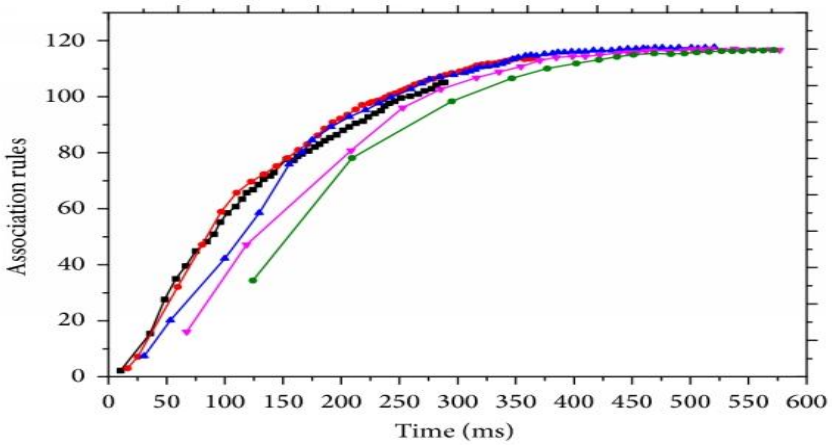

Fig. 9. Number of association rules and time taken for execution for dataset2

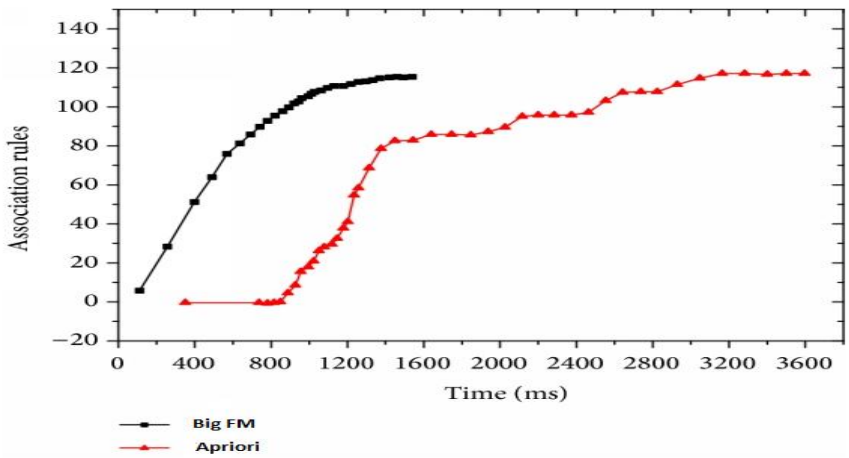

Fig. 10. Number of association rules mined from dataset1

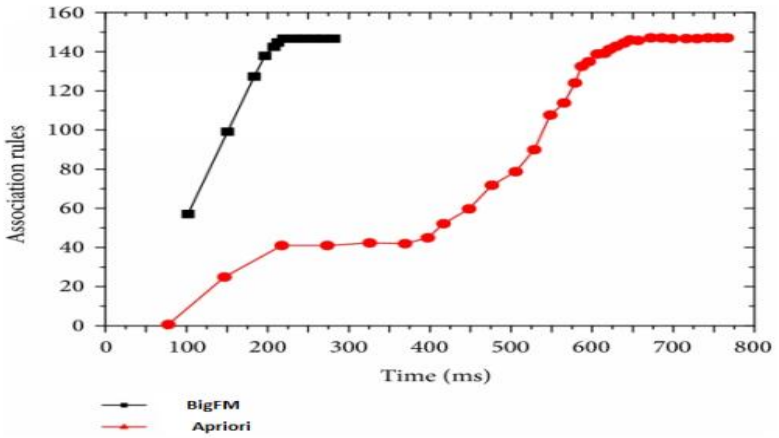

Fig. 11. Number of association rules mined from dataset2

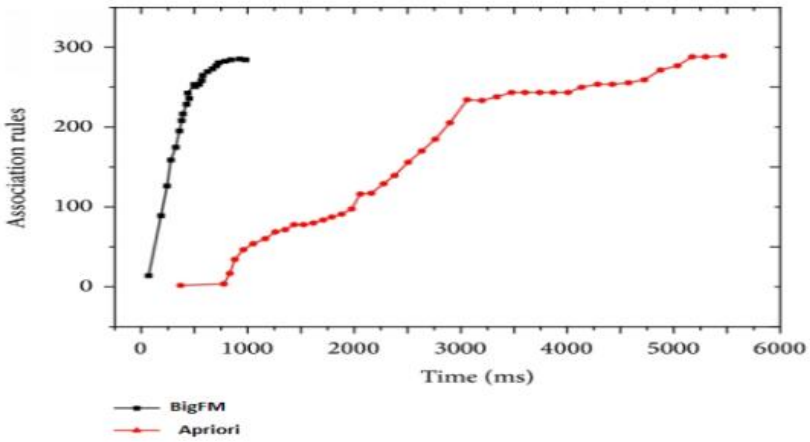

Fig. 12. Mining efficiencies of BigFM and Apriori algorithm with differentmin_sup and min_conf for dataset1

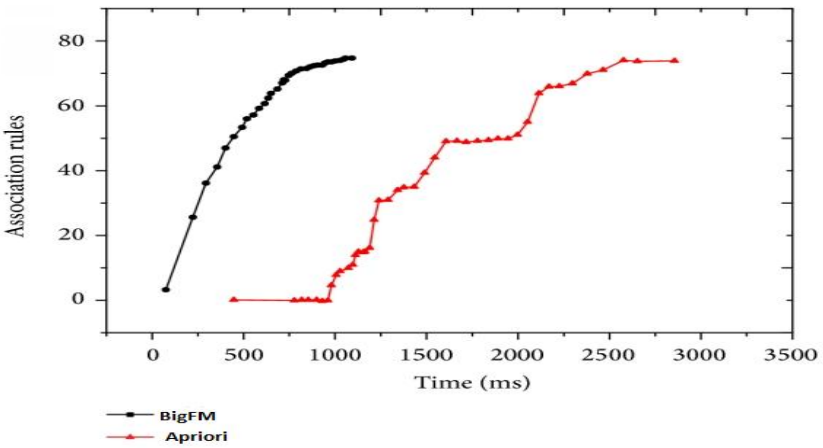

Fig. 13. Mining efficiencies of BigFM and Apriori algorithm with different min_sup and min_conf for dataset 2

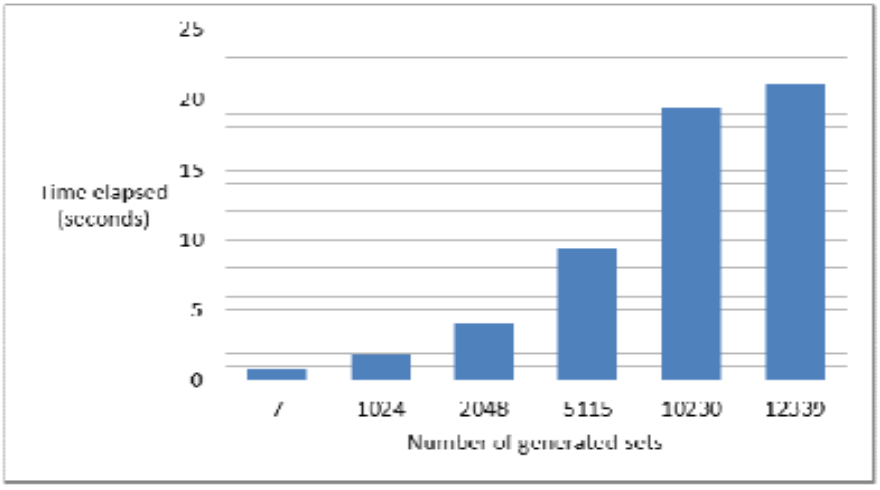

Fig. 14. Relationship between the number of generated itemsets and time

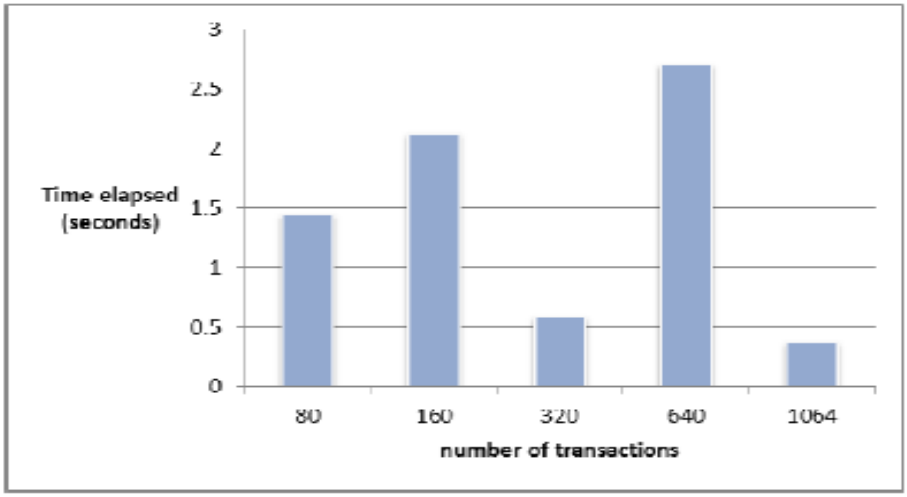

Fig. 15. Relationship between the number of transactions and time using BigFM.

Table 1. Results of BigFM Vs Apriori on Big Data Sets

Status	#New Transa-ctions	# Transa-ctions	BigFM	Apriori
First run	10	10	0.51	0.17
Frequent run	10	20	0.53	0.42
Frequent run	50	70	0.91	1.88
Frequent run	200	270	1.13	3.61
Frequent run	400	670	1.30	5.00
Frequent run	600	1270	2.67	6.98
Frequent run	1200	2470	5.28	14.98
Average frequent run times			1.97	5.482
Decrease percentage				63.95%

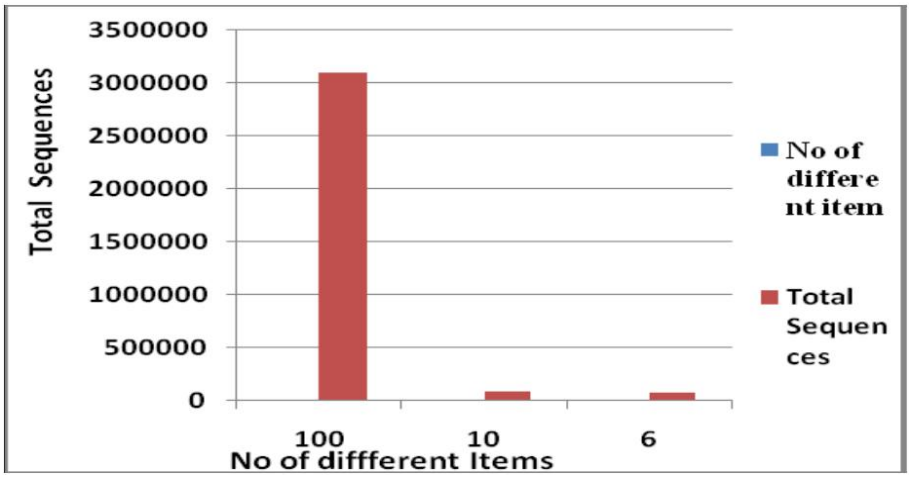

Fig. 16. Number of different items versus total frequent itemsets

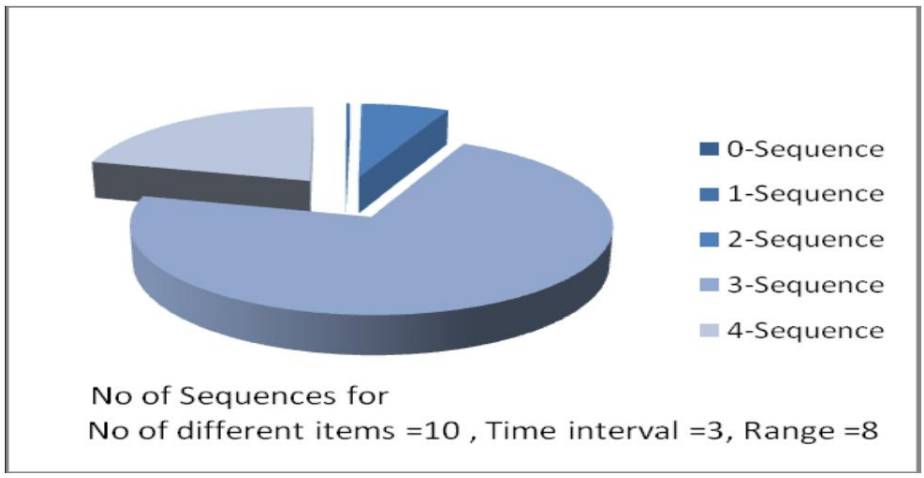

Fig. 17. Number of different itemsets for number of different items

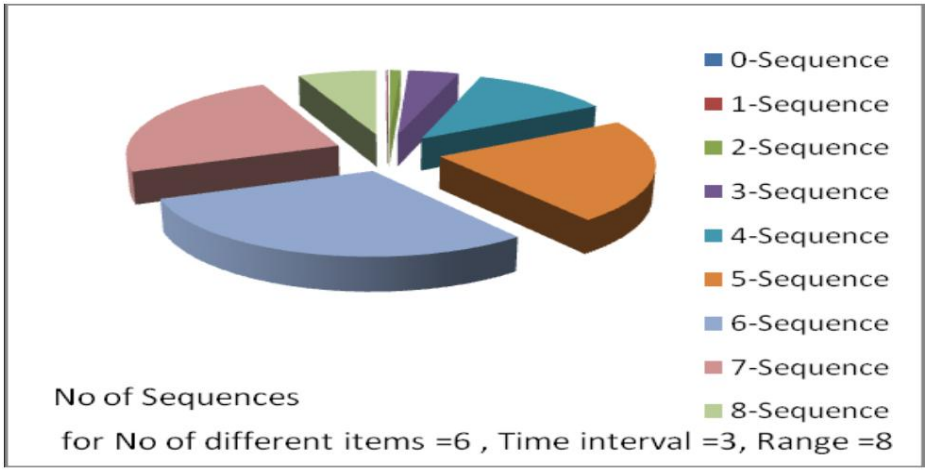

Fig. 18. Number of different itemsets for number of different items

Fig. 16 describes the efficiency in terms of the number of different items against the total frequent itemsets.

5. CONCLUSION AND FUTURE WORK

The association rule generation or mining can be performed effectively in distributed systems that use parallel programing such as the Hadoop framework. This is because this system can be scaled up for large volumes of data and can achieve high accuracy with less computation time and cost. The proposed algorithm considers the type of input data and can be applied to any data formats. By dividing the input data into many chunks of datasets and processing them using different nodes, the execution is made easy. Issues such as data transfer between nodes, data storage, failure of a node and other issues within the cluster are all handled by Hadoop automatically. Thus, the proposed system is highly efficient in terms of scalability and robustness. The proposed algorithm for association rule mining also has the same features and is shown to be efficient. In addition, because the key-value pair approach is used for the processing, it is easier compared to existing binomial approaches. However, the proposed algorithm may not perform at its best in case of extremely large datasets. Therefore, as a topic for future research, use of Fuzzy-based association rule mining in Hadoop can be considered to handle extremely large data. Furthermore, the input data are classified on the basis of the support and confidence values calculated using a suitable classification algorithm. In future work, the algorithms can be extended to implement feature selection using information gain or mutual information before implementing the association rule mining.

COMPETING INTERESTS

Authors have declared that no competing interests exist.

REFERENCES

1. CheikhTidianeDieng, Tao-Yuan Jen, Dominique Laurent, Nicolas Spyratos. Mining frequent conjunctive queries using functional and inclusion dependencies. VLDB J. 2013;22(2):125 150.
2. Ayad Ibrahim, Hai Jin, Ali A. Yassin, Deqing Zou. Towards privacy preserving mining over distributed cloud databases. In Proceedings of the 2nd International Conference on Cloud and Green Computing (CGC 2012), Xiangtan, Hunan, China, IEEE Computer Society. 2012; 130-136.
3. Fan Jiang, Carson Kai-Sang Leung. Stream mining of frequent patterns from delayed batches of uncertain data. In Proceedings of the 15th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2013), Prague, Czech Republic, pages 209{221. Springer-Verlag New York, Inc.; 2013.
4. Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters. In Proc. OSDI. USENIX Association; 2004.
5. CheikhTidianeDieng, Tao-Yuan Jen, Dominique Laurent. An effcient computation of frequent queries in a star schema. In Database and Expert Systems Applications, 21th International Conference, DEXA 2010, Bilbao, Spain, August 30 - September 3, 2010, Proceedings. 2010;Part II:225 239.
6. Markus Hegland. The apriori algorithm – a Tutorial, CMA, Australian National University, WSPC/Lecture Notes Series, 22-27, March 30; 2005.
7. Alfredo Cuzzocrea, Ladjel Bellatreche, Il-Yeol Song. Data warehousing and olap over big data: Current challenges and future research directions. In Proceedings of the 16th International Workshop on Data Ware-housing and OLAP (DOLAP 2013), San Francisco, California, USA, ACM, 2013;67-70.
8. Leila Ismail, Liren Zhang. Modeling and performance analysis to predict the behavior of a divisible load application in a cloud computing environment. Algorithms. 5(2):289-303.
9. Alfredo Cuzzocrea, Carson Kai-Sang Leung, Richard Kyle MacKinnon. Mining constrained frequent itemsets from distributed uncertain data. Future Generation Computer Systems. 2014;37:117-126.
10. Bart Goethals, Dominique Laurent, Wim Le Page, CheikhTidianeDieng. Mining frequent conjunctive queries in relational databases through dependency discovery. Knowl. Inf. Syst. 2012;33(3):655 684.
11. dsonDela Cruz, Carson Kai-Sang Leung, Fan Jiang. Mining `following' patterns from big sparse social networks. In Proceedings of the International Symposium on

Foundations and Applications of Big Data Analytics (FAB 2016), San Francisco, CA, USA. ACM. 2016;923-930.

12. Malu Castellanos, Chetan Gupta, Song Wang, Umeshwar Dayal. Leveraging web streams for contractual situational awareness in operational BI. In Proceedings of the 2010 International Conference on Extending Database Technology/International Conference on Database Theory (EDBT/ICDT 2010) Workshops, Lausanne, Switzerland, ACM. 2010;7:18.
13. Alfredo Cuzzocrea, Domenico Saccia, Jefrerey D. Ullman. Big data: A research agenda. In Proceedings of the 17th International Database Engineering & Applications Symposium (IDEAS 2013), Barcelona, Spain, ACM. 2013;198-203.
14. Kun He, Yiwei Sun, David Bindel, John E. Hopcroft, Yixuan Li. Detecting overlapping communities from local spectral subspaces. In 2015 IEEE International Conference on Data Mining (ICDM 2015), Atlantic City, NJ, USA. 2015;769-774.
15. Bart Goethals, Dominique Laurent, Wim Le Page, Cheikh Tidiane Dieng. Mining frequent conjunctive queries in relational databases through dependency discovery. Knowl. Inf. Syst. 2012;33(3):655 684.
16. Agrawal A, Choudhary A. Identifying hotspots in lung cancer data using association rule mining. 11th IEEE International Conference on Data Mining Workshops. 2011;995–1002.
17. Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation. SIGMOD Rec. 2000;1–12.
18. Li J, Liu Y, Liao Wk, Choudhary A. Parallel data mining algorithms for association rules and clustering. In Intl. Conf. on Management of Data; 2008.
19. Li N, Zeng L, He Q, Shi Z. Parallel implementation of Apriori algorithm based on MapReduce. In Proc. SNPD. 2012; 236–241.
20. Alfredo Cuzzocrea. CAMS: OLAPing multidimensional data streams efficiently. In Proceedings of the 11th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2009), Linz, Austria, Springer Verlag. 2009; 48-62.
21. Yifan Chen, Xiang Zhao, Xuemin Lin, Yang Wang. Towards frequent subgraph mining on single large uncertain graphs. In 2015 IEEE International Conference on Data Mining (ICDM 2015), Atlantic City,NJ, USA. 2015;41-50.
22. Jeferey Dean, Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. Commun. ACM. 2008;51(1):107-113.
23. Anandhavalli M, Suraj Kumar Sudhanshu, Ayush Kumar, Ghose MK. Optimized association rule mining using genetic algorithm. Advances in Information Mining. 2009;1(2):01-04.
ISSN:0975-3265,
24. Dongme Sun, Shaohua Teng, Wei Zhang, Haibin Zhu. An algorithm to improve the effectiveness of apriori. In Proc. Int"l Conf. on 6th IEEE Int"l Conf. on Cognitive Informatics (ICCI'07); 2007.
25. Zhou L, Zhong Z, Chang J, Li J, Huang J, Feng S. Balanced parallel FP-Growth with MapReduce. In Proc. YC-ICT. 2010;243–246.
26. Mannila H, Toivonen H. Discovering generalized episodes using minimal occurrences. In Proc. of ACM Conference on Knowledge Discovery and Data Mining (SIGKDD). 1996;146–151.
27. Li H, Wang Y, Zhang D, Zhang M, Chang EY. Pfp: Parallel fp-growth for query recommendation. In Proc. RecSys. 2008; 107–114.
28. Lin MY, Lee PY, Hsueh SC. Apriori-based frequent Itemset mining algorithms on MapReduce. In Proc. ICUIMC, ACM. 2012; 26–30.
29. Malek M, Kadima H. Searching frequent itemsets by clustering data: Towards a parallel approach using mapreduce. In Proc. WISE 2011 and 2012 Workshops, Springer Berlin Heidelberg. 2013;251–258.
30. Mohammad El-Hajj, Osmar R. Zaiane. Parallel bifold: Largescale parallel pattern mining with constraints. Distributed and Parallel Databases. 2006;20(3):225{243,
31. Fan Jiang, Carson Kai-Sang Leung, Dacheng Liu, Aaron M. Peddle. Discovery of really popular friends from social networks. In Proceedings of the 4th IEEE International Conference on Big Data and Cloud Computing (BDCloud 2014), Sydney, Australia. 2014;342-349.
32. Mohammad El-Hajj, Osmar R. Parallel leap: Large-scale maximal pattern mining in a distributed environment. In Proceedings of the 12th International Conference on Parallel and Distributed Systems (ICPADS 2006), Minneapolis, USA, IEEE. 2006;135-142.

33. Boley M, Grosskreutz H. Approximating the number of frequent sets in dense data. Knowl. Inf. Syst. 2009;65–89.
34. Zeng L, Li L, Duan L, Lu K, Shi Z, Wang M, Wu W, Luo P. Distributed data mining: a survey. Information Technology and Management. 2012;403–409.
35. Apache hadoop; 2013. Available:http://hadoop.apache.org/
36. Apache mahout; 2013. Available:http://mahout.apache.org/
37. De Bie T. An information theoretic framework for data mining. In Proc. ACM SIGKDD. 2011;564–572.
38. Ekanayake J, Li H, Zhang B, Gunarathne T, Bae SH, Qiu J, Fox G. Twister: A runtime for iterative MapReduce. In Proc. HPDC, ACM. 2010;810–818.
39. Finley K. 5 ways to tell which programming languages are most popular Read Write; 2012. Available:http://readwrite.com/2012/06/05/5-ways-to-tell-whichprogramming-lanugages-are-most-popular
40. Hammoud S. MapReduce network enabled algorithms for classification based on association rules. Thesis; 2011.
41. Geerts F, Goethals B, Bussche JVD. Tight upper bounds on the number of candidate patterns. ACM Trans. Database Syst. 2005;333–363.
42. Ghoting A, Kambadur P, Pednault E, Kannan R. NIMBLE:a toolkit for the implementation of parallel data mining and machine learning algorithms on mapreduce. In Proc. ACM SIGKDD, ACM. 2011;334–342.
43. Goethals B. Survey on frequent pattern mining. Univ. of Helsinki; 2003.

Peer-review history:
The peer review history for this paper can be accessed here:
http://www.sciencedomain.org/review-history/23209