I.J. Intelligent Systems and Applications, 2015, 07, 44-49
Published Online June 2015 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijisa.2015.07.06

Mining Interesting Infrequent Itemsets from Very Large Data based on MapReduce Framework

T Ramakrishnudu
Dept. of CSE, National Institute of Technology, Warangal, 506004, India
Email: trk@nitw.ac.in

R B V Subramanyam
Dept. of CSE, National Institute of Technology, Warangal, 506004, India
Email: rbvs66@nitw.ac.in

***Abstract*—**Mining frequent and infrequent itemsets from a given dataset is the most important field of data mining. When we mine frequent and infrequent itemsets simultaneously, infrequent itemsets become very important because there are many valued negative association rules in them. Mining frequent Itemset is highly expensive, if the minimum threshold is low, whereas mining infrequent itemsets is highly expensive, if the minimum threshold is high. When the dataset size is very large, both memory usage and computational cost of mining infrequent items is very expensive. In addition, single processor's memory and CPU resources are not enough to handle very large datasets. Parallel and distributed computing are effective approaches to handle large datasets. In this paper we proposed a method based on Hadoop-MapReduce model, which can handle massive datasets in mining infrequent itemsets. Experiments are performed on 8 node cluster with a synthetic dataset. The performance study shows that the proposed method is efficient in handling very large datasets.

***Index Terms*—** Data Mining, Association Rule, Frequent Itemset, Infrequent Itemset, Hadoop, Mapreduce.

I. INTRODUCTION

Data mining is the process of extracting interesting, previously unknown and potentially useful patterns from the large repositories and it is the core process of Knowledge Discovery in Database (KDD) [13]. Frequent Itemset Mining (FIM) or Association Rule Mining (ARM) is a data mining task [13]. Frequent Itemset is actionable if its support count is greater than or equal to a user-specified threshold, called a *minimum support (ms),* whereas Infrequent Itemset support count is below the *minimum support (ms)*. Association Rule Mining discovers associations among items in a transactional database [1].

Frequent Itemset Mining has been extensively studied in the literature since Agrawal et al. first introduced it in [1, 2]. A typical example of Frequent Itemset Mining application is the market basket analysis. Much effort has been devoted and algorithms proposed for efficiently discovering association rules [2, 3, 4, 5, 6, 25 26]. Association rules provide a convenient and effective way to identify and represent certain dependencies between attributes in a database.

In recent years, there has been an increasing demand for mining the infrequent Itemset. For instance, in [7, 8, 9, 12] algorithms for discovering infrequent itemsets have been proposed. However, traditional infrequent Itemset mining algorithms still suffering from the scalability, especially if the data size is very large.

Mining frequent itemsets is highly expensive, if the minimum threshold is low, whereas mining infrequent itemsets is very expensive, if the minimum threshold is high. When the dataset size is very large, both memory usage and computational cost can still be very expensive in mining frequent as well as infrequent itemsets. In addition, single processor's memory and CPU resources are not enough to handle very large datasets. Additionally, because of exponential growth of the data, the organizations have to deal with continually growing amount of data. As these data grow past hundreds of gigabytes towards terabytes or more, it becomes nearly unimaginable to mine them on a single machine. The solution for the above problem is the distributed computing.

The distributed and parallel computing provides an excellent solution for the above problems. Distributed data mining algorithms attempts to divide the mining problem into sub- problems and solves the sub-problems using homogeneous machines such that each node works independently and simultaneously. Although the distributed data mining improve the performance, but raises quite a few issues like partitioning the input data, load balancing, communication cost between the working nodes and identifying the failure of nodes. To overcome the above problems the MapReduce framework [10, 11] has been introduced. MapReduce, as a simplified distributed framework developed by Google [10, 11], is more appropriate for data processing. It has been widely used in the tasks of search engines, data mining and machine learning etc.

In the MapReduce framework [10, 11], a distributed file system initially partitions the input file and data represented as <key, value> pairs. All computations are carried out by two functions called *Map* and *Reduce*. Both the functions *Map* and *Reduce* take <*key, value*> pair as an input and produce the same pair as an output. The *Map* function takes an input pair and produces intermediate <*key, value*> pair. The *Reduce* function takes an *intermediate key* and the set of *values* associated with that *key*. It merges these values to form a possible small set of values. The output of *reduce* function is written to a distributed file in the Distributed File System (DFS).

Mining very large datasets using MapReduce is not new, some of the researchers made an effort to the Frequent Itemset Mining (FIM) and the association rule mining (ARM) [15-22] on transactional data. And few methods [23-24] deal with different kind of data. All the existing work talks about the methods which are used for frequent Itemsets mining. In this paper, we focused on mining interesting infrequent Itemset from very large data using MapReduce programming model.

The rest of this paper is organized as follows. Section 2 briefly presents the relevant concepts and definitions. In Section 3, the existing strategies are reviewed. The proposed algorithm is presented in Section 4. Experimental results are given in Section 5. The concluding remarks are finally made in Section 6.

II Concepts and Definitions

Let $I = \{i_1, i_2, i_3 ... i_n\}$ be a finite set of items and *DB* be a set of database transactions where each transaction T I is a set of items.

Let X, be a set of items called Itemset. Support of the itemset $X \subseteq I$ is:

Supp(X) = No.of transactions contains X/Total No.of Transactions in DB.

If the support of an itemset *X* is greater than or equal to user defined *minimum support (ms)* threshold, then *X* is called frequent Itemset otherwise infrequent Itemset [13].

Definition1: Partial minimum support count [17] = count (S_i) * *ms*. Where S_i is the $split_i$ and *count* (S_i) is the number of transactions in S_i.

Definition 2: I is an infrequent Itemset of potentially interest [8] if: $\exists X,Y$: $X \cap Y = \phi$, $X \cup Y = I$, *for* $\forall i_k \in X$, $j_k \in Y$, $sup(i_k) \leq ms$, $sup(j_k) \geq ms$, $interest(X,Y) \geq$ minimum interest(*mi*).

Definition 3: Partial minimum interest (pmi) = count (S_i) *m_i , Where m_i is the minimum interesting value.

A. Association Rule:

A (positive) association rule is of the form: $X \Rightarrow Y$, with $X \subseteq I$, $Y \subseteq I$ *and* $X \cap Y = \emptyset$ [1]. Support and confidence of $X \Rightarrow Y$ are defined as [2]:

$$Supp(X \Rightarrow Y) = Supp(X \cup Y) \quad (1)$$

$$Conf(X \Rightarrow Y) = \frac{Supp(X \cup Y)}{Supp(X)} \quad (2)$$

An interesting association rule has support and confidence greater than user given thresholds *minimum support (ms)* and *minimum confidence (mc)* respectively.

B. Hadoop

Hadoop is a framework that allows for the distributed processing of large datasets across cluster of computers using simple programming models [14]. Hadoop is the parallel programming platform built on Hadoop Distributed File Systems (HDFS) for MapReduce computation. The HDFS is the distributed file system designed to run on commodity hardware. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large datasets. HDFS was originally built as infrastructure for the Apache web search engine project. HDFS is a part of Apache Hadoop main project [14].

C. MapReduce:

MapReduce is a programming model and an associated implementation for processing and generating large datasets. Users specify a *map* and *reduce* functions, they takes <*key, value*> pair as an input and generates intermediate <*key, value*> pairs and merges all intermediate values associated with the same intermediate key respectively. Programs written in this function style are automatically parallelized and executed on a large cluster of commodity machines [10][11].

III. Related Work

Several algorithms have been proposed for mining frequent Itemsets using MapReduce framework. But no algorithms have been proposed for mining interesting infrequent Itemsets using MapReduce framework.

Xin Yue Yang et al [21], proposed a one pass algorithm based on Hadoop-MapReduce. The algorithm needs only one scan (MapReduce job) to find all frequent k-itemsets. Firstly, splitting will take place and after that each mapper will apply apriori on that split and it will generate all length Itemsets. It produce output as Itemsets as key and value as one. The reduce will take output of mapper and sum all values for particular keys, then prune infrequent Itemsets and finally generate all frequent Itemsets.

In [19] the authors proposed a k-phase parallel apriori algorithm based on MapReduce. It needs k scans (MapReduce jobs) to find k-frequent items. The algorithm uses two different map functions: one for the first phase and one for rest of the phases. Though the algorithm was successful in finding k-frequent Itemsets using the parallel approach, it has a huge overhead of reading frequent Itemsets of previous phase every time from HDFS. The fundamentals of parallelizing the Apriori algorithm in the MapReduce framework is to design the map and the reduce functions for candidate generations and support counting.

Each mapper calculates counts of each candidate from its own partition, and then each candidate and the corresponding count are output. After map phase, candidates and its counts are collected and summed in reduce phase to obtain partial frequent Itemsets. By using count distribution between map phase and reduce phase, the communication cost can be decreased as much as possible. Since frequent 1-itemsets are found in pass-1 by simple counting of items. Phase-1 of the algorithms is strait forward. The mapper outputs <item, 1> pair's for each item contained in the transaction. The reducer collects all the support counts of an item and outputs the <item, count> pairs as a frequent 1-itemset to the L1, when the count is greater than the minimum support count. The k-itemsets are passed as an input to the mapper function and the mapper outputs <item, 1>, then the reducer collects all the support counts of an item and outputs the <item, count> pairs as a frequent k-itemset to the Lk.

Othman Yahya et al [17] proposed a two-phase algorithm on Hadoop MapReduce, which is more efficient than previous one-phase and k-phase algorithms. It needs only two MapReduce phases to find all frequent k-itemsets. In phase1, each input split is assigned a map task (executed by map worker) that calls a map function to process this split. The mapper function uses traditional Apriori with the partial minimum support count; which is equal to the number of transactions in the split multiply by the minimum support threshold.

The mappers output is a list of intermediate <key, value> pairs grouped by the key via combiner, and stored in the map worker where the key is an element of partial frequent k-itemsets and the value is its partial count. When all map tasks are finished, the reduce task is started. The mappers output are shuffled to the reduce worker that calls a reduce function. The output of reduce function is a list (Lp) of <key, value> pairs, where the key is an element of partial frequent k-itemsets and the value equal one, stored in HDFS.

In phase two, one extra input is added to the data flow of the previous phase, which is a file that contains all partial frequent k-itemsets. The map function of this phase counts occurrence of each element of partial frequent k-itemset in the split and outputs a list of <key, value> pairs, where the key is an element of partial frequent k-itemset and the value is the total occurrence of this key in the split. The reduce function outputs a list (Lg) of <key,value> pairs, where the key is an element of global frequent k-itemsets and the value is its occurrence in the whole dataset. The main drawback of this method is the large number of partial frequent itemsets may overload the map functions of the phase-II.

Mohammadhossein B et al [23] proposed a scalable and distributable binomial method, which deals with different kind of data. It converts the input data into binomial format to take benefit of MapReduce method structures, and then mine association rules from that data. It uses the layered approach to mine frequent itemsets from the binomial data.

Zahara Farzanaryar et al [24] proposed a method based on insignificant Itemset property, and it deals with social network data. It improves the method proposed in [17].

IV PROBLEM DESCRIPTION AND PROPOSED METHOD

Most of the methods proposed for mining frequent Itemsets using Hadoop MapReduce, but no method was designed for mining infrequent Itemsets using Hadoop MapReduce. It is necessary to design a method to mine infrequent Itemsets from very large data using Hadoop MapReduce framework.

Problem Statement: Given a large transactional database *LDB* and user-defined *minimum support (ms)* value, *minimum interesting (mi)* values, the problem is to find interesting infrequent Itemsets using Hadoop MapReduce framework.

We propose a two phase method to find interesing infrequent Itemsets and a pruning technique to decrease the number of intermediate infrequent Itemsets during phase one.

In first phase the input is divided into number of chunks and each chunk is assigned to one node. The mapper function at each node accepts two more inputs are partial minimum support (pms) and partial minimum interest (pmi) in addition to the chunk, and it generates the candidate k Itemsets of that chunk. If the partial support count of an itemset is less than the partial minimum support and the interest is greater than the partial minimum interest then the itemset is assigned to reducer. The reducer function outputs the itemsets in the form <key, 1>. The algorithms for mapper and reducer are shown in "Fig.2" and "Fig.3".

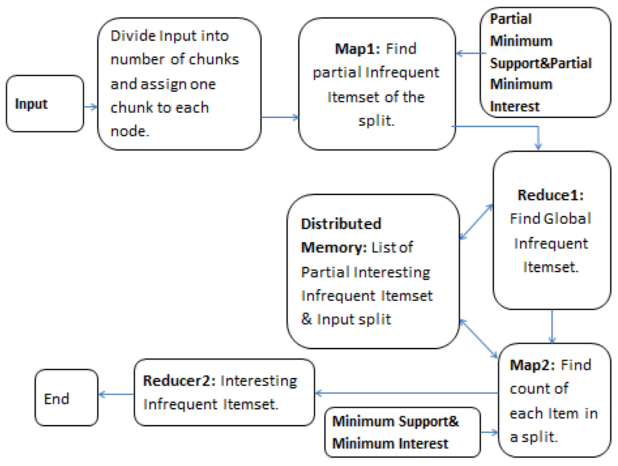
Input
Divide Input into number of chunks and assign one chunk to each node.
Map1: Find partial Infrequent Itemset of the split.
Partial Minimum Support&Partial Minimum Interest
Reduce1: Find Global Infrequent Itemset.
Distributed Memory: List of Partial Interesting Infrequent Itemset & Input split
Map2: Find count of each Item in a split.
Minimum Support& Minimum Interest
Reducer2: Interesting Infrequent Itemset.
End

Fig. 1. Work flow the method

In second phase the mapper function of this phase takes an infrequent itemset list from the distributed file system and the input chunk as an input. It calculates the frequency of an itemset in that chunk, and return <key, count> as an output. The reducer summaries the count of each itemset and for each itemset if the count is less than the minimum support and the interest is greater than the minimum interest then the itemset is considered as an interesting infrequent itemset.

The algorithms for mapper and reducer are shown in "Fig.4" and "Fig.5". The detailed flow diagram of the process is shown in "Fig.1".

In each phase different Mapper and Reducer functions are used. In the first phase the mapper function accepts three different inputs called input split, minimum partial support and partial minimum interest where as in the second phase of the mapper accepts only two different inputs called infrequent Itemset list generated in phase one and the input split.

Map1()

Input: Split-S_i; partial minimum support(pms); partial minimum interest(pmi).

Output: <key, value>; key: interesting infrequent k-Itemset of the split Si; value: partial count of k-Itemset.

```
Begin
1.  Ck =Generate_Candidate_Itemsets($S_i$)/* generates candidate itemsets of the split$S_i$ */
2.  For each $I_k \in C_k$ do
3.    If k=1 then $L_1$= $L_1$U $I_1$
4.    If k>1
5.      If(partial_count($I_k$)<pms&& Interest($I_k$)>pmi)then
6.        $L_k$= $L_k$U $I_k$;
7.      End if;
8.    End if
9.    $L_k$= $L_k$U $L_1$ ;
10. Foreach Item I in $L_k$ do
11.   Output(I, pc) /* pc: partial count */
12. End for
End;
```

Fig. 2. Algorithm for Map1

Reduce1()

Input: <key, value>; key: interesting infrequent k-Itemset of the split Si; value: partial count of k-Itemset.

Output: <key, 1>; key is global candidate Infrequent Itemsets.

```
Begin
1. Foreach key do
2. Output(key,1)
3. End for
End
```

Fig. 3. Algorithm for Reduce

Map2()

Input: S_i: split, L: infrequent Itemset list read from distributed memory.

Output: <key, value> key: Itemset from list L, value: Item count in S_i .

```
Begin
1.  Foreach Item I in L do
2.    count=count+ find(I,$S_i$ ) /*The find() function finds the occurrence of I in $S_i$ */
3.  End for
4.  Out(I, count)
End
```

Fig. 4. Algorithm for Map2

Reduce2()

Input: <key, value> key: candidate itemset, value: Item count in each split, minimum support (ms) and minimum interest(mi).

Output: <key, value> key: Interesting Infrequent Itemset; value: Its count in the whole dataset

```
Begin
1.  Foreach key $I_k$ do
2.    Foreach value in $I_k$'s list
3.      count($I_k$)=count($I_k$)+$I_k$.value;
4.    End for
5.  End for
6.  if(count($I_k$)<ms&&Interest($I_k$)>mi)
7.    Out($I_k$, count)
8.  End if
End
```

Fig. 5. Algorithm for Reduce

V. Experimental Results

In this section we measure the performance of the proposed algorithm running on cluster of nodes. To evaluate the performance of our method we formed few clusters with different size. All the experiments were conducted in a Hadoop 2.2.0 cluster where each node contains 2.20 GHz processors with 4 GB RAM, and a 500 GB hard disk and 2 a gigabyte Ethernet link.

Synthetic dataset is used in experiments. It is a transactional dataset. It consists 1,000 distinct items and the average size of the transaction is 120.

We test our approach to find the infrequent itemsets. A set of experiments conducted to show the behaviour of our approach at different minimum support and dataset size in one cluster and different cluster size for fixed minimum support. For better results each case is executed two times and the average values are taken.

"Fig.5" depicts performance of the algorithm; the execution time of the algorithm is observed for different dataset size with a fixed minimum support on 8 nodes cluster. The results show that the algorithm takes less time even for larger datasets.

"Fig.6" delineates the performance of the algorithm; in this the execution time of the method is observed for different minimum support values for two dataset sizes of 1GB and 10GB. The results show that there is no much difference in execution time when the minimum support is high, but there is a difference in case of smaller minimum support.

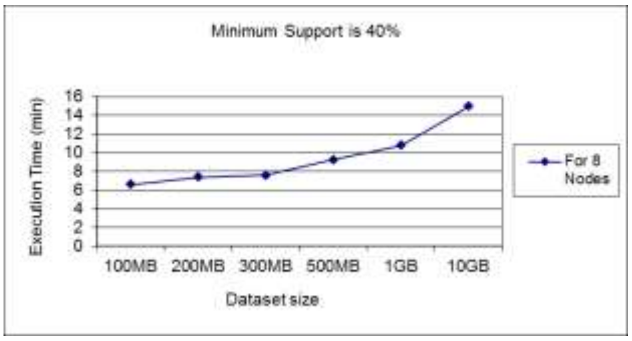

Fig. 6. Execution time for different dataset size

Then we fix the minimum support at 40% and test the behaviour of the proposed method at different cluster size for two different data sizes of 500MB and 1GB. "Fig.7" shows results of these experiments. The results show that there is a much difference in the execution time if the cluster size is less, but there is no much difference in case of number of nodes are increased in the cluster. Also we observed that the impact of MapReduce framework is very less when the cluster size small.

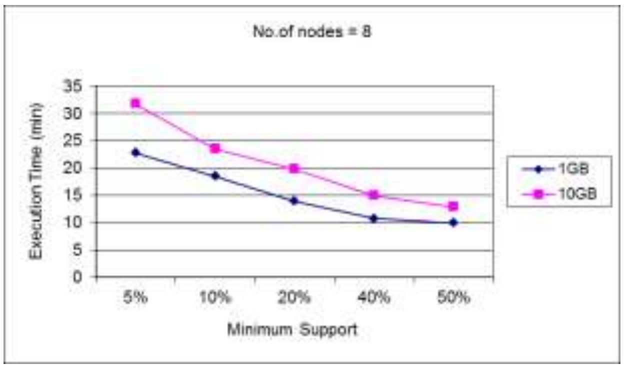

Fig. 7. Execution time for different minimum support

At the end we evaluate the impact of the prunning technique, which is used during the first phase of the method. First we execute the algorithms without prunning technique, it produce large number of intermediate items. Intermediate items effects on the performance of the entire process. And next we execute the algorithm with the prunning technique, it produce less number of items.

“Fig.8” shows the impact of the prunning technique. The results show that the impact of prunning techniques is very high when the the data size is large.

“Fig.9” shows ths effect of prunning technique. We fixed the minimum support and the dataset size, and observed the execution time by changing the cluster size. The rusults show that there is no much difference beteen with and without prunning technique.

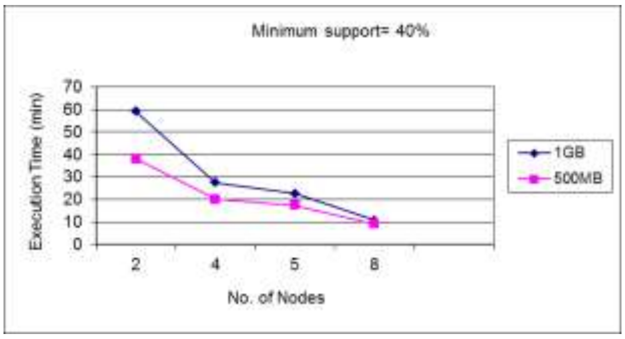

Fig. 8. Execution time for different Cluster size

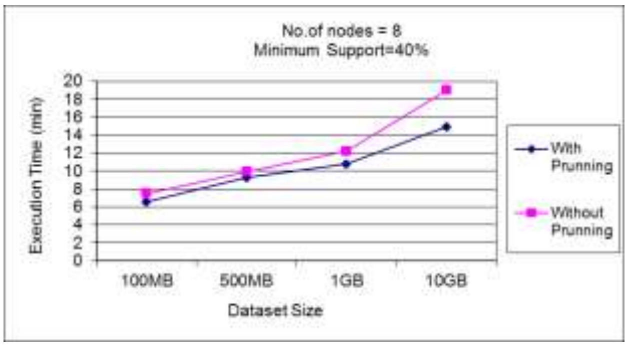

Fig. 9. Execution time for different minimum support

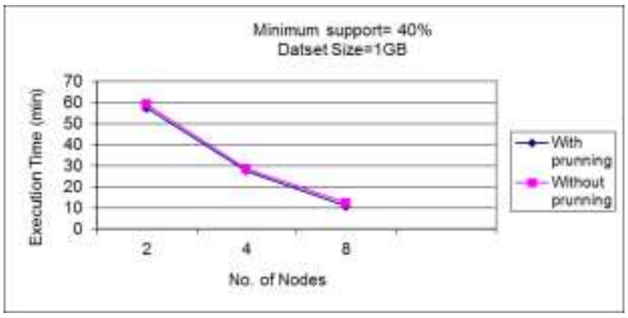

Fig. 10. Execution time for different minimum support

VI. Conclusion and Future Work

Finding infrequent Itemset is one of the most important data mining problems. The task of finding interesting infrequent items from very large data requires a lot of computational and memory power.

In this paper we have proposed new method to find interesting infrequent Itemsets from very large data based on MapReduce framework. The results show that the proposed method in this paper is very efficient in finding infrequent items from very large datasets. Also the experimental results show that the proposed method is more efficient as the data size is increased.

Our future research works include design of better interesting functions, which produce less number of intermediate items during its process. Performance of the reducer is depending on the intermediate nodes produced by the mapper.

References

[1] R. Agrawal, T. Imielinski, A. Swami, “Mining association rules between sets of items in large databases”, In Proceedings of ACM SIGOM International Conference on Management of Data, New York, May 1993, pp. 207–216.

[2] R.Agrawal and R.Srikant, “Fast algorithms for mining association rules”, In Proceedings of 20th International Conference on VLDB, Chile, May 1994, pp. 207–216.

[3] J.Han and Y.Fu, “Mining multiple-level association rules in large databases”, IEEE Trans. on Knowledge and Data Engineering, Vol. 11, No 5, September 1999, pp. 798-805.

[4] X. Wu, C. Zhang and S. Zhang,”Efficient mining of both positive and negative association rules”, ACM Trans. on Information Systems, vol.22 (3), 2004, pp 381–405.

[5] Chris Cornelis, Peng Yan, Xing Zhang, Guoqing Chen: “mining positive and negative association rules from large databases”, in IEEE conference on Cybernetics and Intelligent Systems, Bangkok, June 2006, pp.1-6.

[6] X. Yuan, B.P. Buckles, Z. Yuan and J. Zhang, ”Mining negative association rules”, Proceedings of the Seventh International Symposium on Computers and Communication, Italy, July 2002, pp. 623–629.

[7] Junfeng Ding, Stephen S.T. Yau, “TCOM, an innovative data structure for mining association rules among infrequent items”, Computers and Mathematics with Applications, Vol. 57, No. 2, January 2009, pp. 290-301.

[8] Ling Zhou, Stephen Yau, “Efficient association rule mining among both frequent and infrequent items”,

Computers and Mathematics with Applications, Vol. 54, No.6, September 2007, pp. 737–749.
[9] Luca Cagliero and Paolo Garza “Infrequent weighted itemset mining using frequent pattern growth ” IEEE Transactions on Knowledge and Data Engineering, Vol. 26, No. 4, April 2013, pp. 903-915.
[10] Jeffery Dean and Sanjay Ghemawat “MapReduce: simplified data processing on large clusters”, 6th Symposium on Operating Systems Design and Implementation, October 2004, pp.107-113.
[11] Jeffery Dean and Sanjay Ghemawat “MapReduce: simplified data processing on large clusters”, Communications of the ACM, Vol. 51, No.1, 2008, pp. 107-113.
[12] Dong, Z Zheng, Z Niu and Q Jiam ”Mining infrequent itemset based on multiple level minimum supports”, 2nd Int. Conf. on Innovative Computing, Information Control, 2007.
[13] Jiawei Han and Micheline Kamber, “Data mining: concepts and techniques”, Morgan Kaufman, 2001.
[14] Apache Hadoop Project, http://hadoop:apache.org/. accessed at 201408251930.
[15] Jongwook Woo, “Market basket analysis algorithm on Map/Reduce in AWS EC2”, International Journal of Advanced Science and Technology, Vol.46, September 2012, pp. 25-38.
[16] Su-Qi W, Yu-Bin Y, Guang-Peng C, Yang G and Yao Z, “MapReduce-based closed frequent Itemset mining with efficient redundancy filtering”, 12th International Conference on Data Mining Workshops, December 2012, pp. 449-453.
[17] Othman Y, Osman H and Ehab E, “An efficient implementation of apriori algorithm based on hadoop-MapReduce model”, International Journal of Reviews in Computing, Vol. 12, December 2012, pp.57-67.
[18] Ming-Yen Li, Pei-Yu L and Sue-Chen H, “Apriori-based frequent Itemset mining algorithms on MapReduce”, The 6th International Conference on Ubiquitous Information Management and Communication, Malaysia, February 2012, pp.257-264.
[19] Ning Li, Li Z, Qing H and Zhongzhi S, “Parallel implementation of apriori algorithm based on MapReduce”, 13th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, Japan, August 2012, pp. 236-241.
[20] Le Z, Zhiyong Z and Jin C, “Balanced parallel fp-growth with mapreduce”, in 2010 IEEE Youth Congress on Information Counting and Telecommunications, Chaina, November 2010, pp. 243-246.
[21] Xin Yue Y, Zhen L and YanFu, ”Mapreduce as a programming model for association rules algorithm on hadoop”, in 3rd International Conference on Information Sciences and Interaction Sciences, Chaina, June 2010, pp. 99-102.
[22] Matteo Riondato, Justin A. DeBratant, Rodrigo Fonseca, and Eli Upfal, “PARAM: A parallel randomized algorithm for approximate association rules mining in MapReduce” in 21st ACM International Conference on Information and Knowledge Management, USA, October 2012, pp.85-94.
[23] Mohammadhossein B and Madhi Niamanesh, “ScaniBino: An effective MapReduce-based association rule mining method”, in proceedings of the the sixteenth International Conference on Electronic commerce, USA, August 2014, pp.1-8.
[24] Zahara Farzanaryar and Nick Cercone, “Efficient mining of frequent itemsets in social network data based on MapReduce framework”, in proceedings of the IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, Canada, August 2013, pp.1183-1188.
[25] M M Rahman “Mining social data to extract intellectual knowledge”, International Journal of Intelligent Systems and Applications, Vol. 4, No. 10, 2012, pp. 15-24.
[26] Thabet slimani and Amor Lazzez, “Efficient analysis of pattern and association rule mining approaches”, International Journal of Information Technology and Computer Science, Vol. 6, No. 3, 2014, pp. 70-81.

Authors' Profiles

T Ramakrishnudu was born on June 01, 1980. He received B.Tech (Computer Science and Engineering) degree from Jawaharlal Nehru Technological University, Hyderabad, India in 2001 and M.Tech (Computer Science and Technology) from Andhra University, Visakhapatnam, India in 2005. Currently he is working as an Assistant Professor in the department of Computer Science and Engineering in National Institute of Technology Warangal, India. His research interests include Data Mining, Distributed Data Mining and Big Data Analytics. He is a member in IEEE, ACM and Computer Society of India.

R B V Subramanyam received M.Tech and Ph.D from Indian Institute of Technology Kharagpur, India. Currently He is working in National Institute of Technology Warangal. He has published many journal and conference papers in the areas of Data Mining. Some of his research interests include Data Mining, Distributed Data Mining, Fuzzy Data Mining, Distributed Data Mining and Big Data Analytics. He is one of the reviewers for IEEE Transactions on Fuzzy Systems and also for Journal of Information and Knowledge Management. He is member in IEEE and The Institution of Engineers (India).

How to cite this paper: T Ramakrishnudu, R B V Subramanyam,"Mining Interesting Infrequent Itemsets from Very Large Data based on MapReduce Framework", International Journal of Intelligent Systems and Applications (IJISA), vol.7, no.7, pp.44-49, 2015. DOI: 10.5815/ijisa.2015.07.06