**Journal of Advances in Mathematics and Computer Science**

**27(1): 1-11, 2018; Article no.JAMCS.39635**
***ISSN: 2456-9968***
*(Past name: British Journal of Mathematics & Computer Science, Past ISSN: 2231-0851)*

# Improved FTWeightedHashT Apriori Algorithm for Big Data using Hadoop-MapReduce Model

**Sarem M. Ammar[1*] and Fadl M. Ba-Alwi[2]**

*[1]Departement of IT, Yemen Academic for Graduate Studies, Yemen.*
*[2]Faculty of Computer & IT, Sana'a University, Yemen.*

***Authors' contributions***

*This work was carried out in collaboration between both authors. Author SMA designed the study, designed the framework of the study, wrote the pseudo code of the proposed algorithm, implemented the proposed algorithm on eclipse (java-based) and wrote the first draft of the manuscript. Author FMBA managed the analyses of the study, reviewed the results and the all proposed work in general (as a supervisor of author SMA on master's degree). The final manuscript read and approved by both authors.*

***Article Information***

DOI: 10.9734/JAMCS/2018/39635
Editor(s):
(1) Francisco Welington de Sousa Lima, Professor, Dietrich Stauffer Laboratory for Computational Physics, Departamento de Física, Universidade Federal do Piaui, Teresina, Brazil.
(2) Dariusz Jacek Jakóbczak, Assistant Professor, Chair of Computer Science and Management in this department, Technical University of Koszalin, Poland.
Reviewers:
(1) Carson Leung, University of Manitoba, Canada.
(2) Sudhakar Singh, Institute of Science, Banaras Hindu University, India.
(3) Ishwar Kalbandi, JSPM'S Jayawantrao Sawant College of Engineering, India.
(4) Enrique Lazcorreta Puigmartí, Institute Center of Operations Research, Miguel Hernández University of Elche, Spain.
Complete Peer review History: http://www.sciencedomain.org/review-history/24053

***Original Research Article***

***Received: 25th January 2018***
***Accepted: 30th March 2018***
***Published: 9th April 2018***

## Abstract

The most significant problem of data mining is the frequent itemset mining on big datasets. The best-known basic algorithm for frequent mining itemset is Apriori. Due to the drawbacks of Apriori algorithm, many improvements have been done to make Apriori better, efficient and faster. We have reviewed over 100 papers related to this work that include enhancements be done to improve Apriori algorithm. Weighted based Apriori and Hash Tree based Apriori are the most significant improvements. One of the recent papers integrated the weight concept of weighted Apriori and Hash tree construction concept of Hash Tree Apriori to produce a hybrid Apriori algorithm named WeightedHashT. In this paper, we aim to propose a new approach to improve WeightedHashT Apriori algorithm on big data using Hadoop-MapReduce model by employing the transaction filtering technique. The experiment of this work using different datasets manifests that the proposed algorithm is efficient and effective regarding execution time.

**Corresponding author: E-mail: saremammar@gmail.com;*

*Keywords: Big data; hadoop; mapreduce; apriori; frequent itemset mining.*

# 1 Introduction

We swim in an exceedingly ocean of data, and therefore the level of the ocean is increasing rapidly. So-seeming, we tend to live in the data age. Terabytes or petabytes of data pour into our laptop networks, the planet Wide internet (WWW), and varied data storage devices on a daily basis from business, society, science and engineering, medicine, and virtually every alternative facet of lifestyle. This explosive growth of accessible data volume could be a result of the computerization of our society and therefore the quick development of powerful knowledge assortment and storage tools. Big data is defined as any data source that manages a vast amount of data and has three dimensions known as large volume, wide variety, and high velocity [1]. The computerization of our society has well increased our capabilities for each generating and grouping knowledge from various sources. A tremendous quantity of data has flooded virtually every facet of our lives. This explosive growth of a preserved or temporary data has generated an urgent need for new techniques and automated tools that can intelligently assist us in transforming the vast amounts of data into helpful information and knowledge. This has a diode to the generation of a promising, and sensible frontier in applied science called data mining. Data mining is the method to achieve new models for data analysis to avail of business [2]. It is additionally a method of discovering unknown and helpful information on giant databases. Data mining methods involve seven common categories of data mining techniques consistent with operating and application purpose: outlier detection, association rule mining (ARM), classification, clustering, regression, summarization, and mining time-series data. The necessity of data mining algorithms is to handle and process the huge data. Big Data algorithms analyze the massive quantity of information, reveal the association rules, hidden patterns, trends and the alternative purposeful information. Among them, ARM is the foremost well-known one. Several business enterprises accumulate giant quantities of data onto their regular operations. As an example, immense amounts of client purchase data are collected daily at the checkout counters of grocery stores. Retailers have an interest in analyzing the data to be told concerning the getting behavior of their customers. Such valuable information is accustomed support a range of business-related applications like marketing promotions, inventory management, and client relationship management. To address aforesaid problems, association analysis comes in, that is helpful for discovering interesting relationship hidden in giant datasets. The uncovered relationships are delineated within the variety of association rules or sets of frequent items.

Association Rule Mining (ARM) is the process of finding the co-occurrences between items in a given dataset and describing the association relationship among different items. There are many studies have been proposed to association rule mining on big data. The most used algorithm is Apriori. To address the drawbacks of Apriori, some methods have been proposed such as [3-15]. The common drawback of these algorithms is the delay in execution time. We have noted that the latest study [4] is efficient to improve the performance of Apriori on MapReduce implementation in terms of time execution as well as memory utilization; it was designed by combining the benefits of weighted Apriori algorithm and Hash tree Apriori algorithm to get a hybrid Apriori algorithm called "WeightedHashT" whereas, the result that they have introduced is about 50 MB of memory utilization and about 9900 ms for execution time. Concerning their study, we consider that there is a delay in execution time especially when the data increasing exponentially. Increasing the volume of data onto various types; structured, unstructured, and semi-structured need to be processed in a fast way by using a different architecture than the existed ones. A distributed architecture can serve as an umbrella for many different systems. Hadoop is just one example of a framework that can bring together a broad array of tools and services such HDFS and MapReduce for splitting a given big task into small-tasks and running them in a parallel way. Thus, the proposed work aims to improve the algorithm presented in [4] by reducing the time complexity to get the best result in finding the frequent itemset on big data using Hadoop-MapReduce model.

The proposed work is conducted with some boundaries. We only focus on the time execution aspect while the other aspects are not considered. Also, we focused on generating frequent itemset on big datasets, so generating association rules is not considered. Therefore, will not consider the measure confidence that used in Apriori algorithm.

## 2 Related Works

The most famous algorithm in the ARM is Apriori, which has been proposed in 1993 by Agrawal [16]. Several Apriori-based algorithms have been introduced to minimize the time execution, reduce the memory consumption, and enhance the scalability [3-15,17-19].

Y. Xun et al. [17] designed a parallel algorithm for frequent mining itemset named FiDoop using the model of MapReduce. To obtain squeezed storage and avert constructing conditional pattern bases, FiDoop integrates frequent items ultra-metric tree, instead of classic FP trees. To complete the mining task in FiDoop, only three MapReduce jobs are executed. In the pivotal third MapReduce job, the itemsets are decomposed independently by the mappers, small ultra-metric trees are built by the reducers, and separate the real mining of these trees. They implemented FiDoop at their in-house cluster of Hadoop. They also developed a metric for workload balance to scale load-balancing with the cluster of computing nodes. Their developed metric named FiDoop-HD, which is an expansion of FiDoop, to fast the performance of mining for elevated-dimensional data analysis. The reason for their proposed paper is to resolve automatic parallelization and load-balance challenges to the present parallel algorithms for frequent itemset mining.

D. Apiletti et al. [18] worked a comparative analysis that reviewed the scalable algorithms that based on spark and Hadoop technologies. They concentrated their work on parallel algorithms of frequent itemset mining in the field of big data. Their experimental comparison considered distinct data distributions, and the cases used. The evaluation of their survey intended to infer the relationship between the performance of the algorithms and its strategies of parallelization.

S. Singh et al. [19] introduced six factors which affect the performance of Apriori-based MapReduce on Hadoop cluster. They have investigated the factors that affect the time execution of Apriori-based MapReduce running on the Hadoop clusters of homogeneous and heterogeneous. The factors are certain of jointly algorithmic and non-algorithmic refinements. Treated factors certain algorithmic refinements are data structures and transaction filtering. While the non-algorithmic factors involve the nodes of bad performance, locality of data and distribution of data blocks, reflective execution, and the control of parallelism with spilled input size. They claimed that these factors are important to influence the performance of the algorithm.

Singh et al. [3] had executed the Apriori algorithm on Hadoop for three central structures which are the trie, hash tree, and hash table trie. They evaluated the time execution for each one of these data structures. Their study considers these structures of Java to be used in MapReduce code. They stated that ‘In a sequential computing environment, trie outperformed hash tree experimentally as well as theoretically. HashTableTrie theoretically accelerates the Apriori algorithm but experimentally could not succeed. In their experiment on Hadoop cluster, trie outperforms hash tree as usual, but hash table trie is outstanding among the three data structures for both real-life and synthetic datasets.'

However, Sumithra et al. [4] had proposed a hybrid Apriori algorithm which combined the interests in jointly hash tree apriori algorithm and weighted based-apriori ‘tree-based hybrid approach.' in their proposed algorithm, the selected datasets have large numbers of transactions. In these transactions, it is necessary to presage and expect the items in the given dataset. For expecting the frequent itemsets and associations with the dataset, it mostly consumes lots of time and memory. To address the problem, their proposed algorithm extracts the frequent itemsets without the candidate generation of itemsets. Tree construction approach is used to minimize the memory consumed in the expectation of association rules during the reducing task. The algorithm uses the approaches that based on correlation to enhance the process. Their work conclude with an implementation of new improved WeightedHashT Apriori algorithm that called a hybrid algorithm in the environment of Hadoop. In their proposed algorithm, they selected retail and mushroom datasets.

# 3 Proposed Work

Although WeightedHashT succeeded in extracting frequent itemsets, it consumed more time to execute because of multi scans of the dataset where the scanning process includes frequent and infrequent itemsets. To address this problem, we propose a novel approach named "FTWeightedHashT" (Filtered Transaction Weighted Hash Tree). The framework (Figs. 1-2) of the proposed algorithm works as follows. A given data set will be processed in a parallel way in Hadoop cluster of n nodes. Fig. 1 shows the different phases of the proposed algorithm for the ¡th node. At the first step, the entire dataset will be stored in HDFS. JobTracker will partition the dataset into n small parts ($P1$, $P2$, …, $Pn$). TaskTracker will assign each part $Pi$ (i=1,2,..n) to ¡th node. At the second step, each partition $Pi$ (i=1,2,..n), will be scanned (i.e., first scan) to compute the frequency of itemsets (denoted by $FPi$).The obtained $FPi$ is used to compute the weight of itemsets (denoted by $WPi$) for transaction filtering as the third step. Transaction filtering was first used in a sequential implementation of Apriori by Christian Borgelt [20]. 'If T is a transaction of a dataset. Then, a filtered transaction of T is the itemset obtained by removing infrequent items from transaction T. Transaction filtering is sufficient to determine all the frequent itemsets' [21]. We apply (at the present work) transaction filtering on the efficient parallel implementation of Apriori using Hadoop-MapReduce model. The transaction filtering phase is used to scan the partition $Pi$ at the ¡th node to determine all frequent itemsets. In the fourth step, the obtained frequent itemsets will be stored in a file called "FT file" to be scanned instead of scanning the whole partition of the dataset. This will address the main drawback of the consumed time calculated by the hybrid WeightedHashT Apriori algorithm [4] (i.e., wasting time for scanning both frequent and infrequent items). After that, the tree is constructed by getting the frequent itemsets from the FT file and mapping it to a path of a fixed order in the tree (i.e., Second scan) as the fifth step. These steps will be repeated until all transactions in the ¡th node are covered to get the frequent itemsets for the partition $Pi$ and return the result to the JobTracker. Certainly, Hadoop will process all the above steps for all nodes in a parallel way and combine the discovered frequent itemsets from each node and return the final result to the application; as shown in Fig. 2.

## 3.1 Design of the proposed FTWeightedHashT apriori algorithm

FTWeightedHashT Apriori algorithm contains three main phases: Item Prediction phase, Filtration phase, and Tree Construction phase (see Fig. 3). In the first phase, the algorithm scans the dataset for counting the frequencies of the itemsets. At the second phase, the algorithm calculates the weight of itemsets by giving a transaction a particular importance/weight where the weight is the relation of the item with other items while treating the total unique transactions; then filters the transactions by removing infrequent items (i.e. items with support less than the minimum support value). The last phase contains of two sub-phases namely relevancy computation and reachability verification. These sub-phases depict for constructing the frequent tree with both types of relation, direct and indirect. The relevancy computation (direct relationship) is the process of expecting the relevance among nodes in the tree that means the itemsets in the tree of the frequent itemset. The reachability verification (indirect relationship) refers to the ability to reach the end point of the transaction for each itemset in the tree. More explanation of tree generation can be found in the next subsection.

The proposed algorithm mines the transactional dataset that is deemed to be the data that can be distributed. The algorithm uses weighted support (support) measure instead of using support count measure. The diagram above describes the way of explaining the logic of the algorithm formulation. It iterates for a specific number of times till no new frequent itemset is found. The algorithm is defined as map tasks to be executed on the distributed data which is available on various nodes of Hadoop cluster. Then the results are combined using the reduce task; as shown in Fig. 2.

## 3.2 Pseudo code of the proposed FTWeightedHashT apriori algorithm

The phases presented in the flow diagram of Fig. 3 are implemented in Java programming language as given in Fig. 4 that is the Pseudo code of the proposed algorithm. The algorithm has a transactional

dataset as input, and it produces frequent itemsets as output. We illustrate the phases of the algorithm as follows.

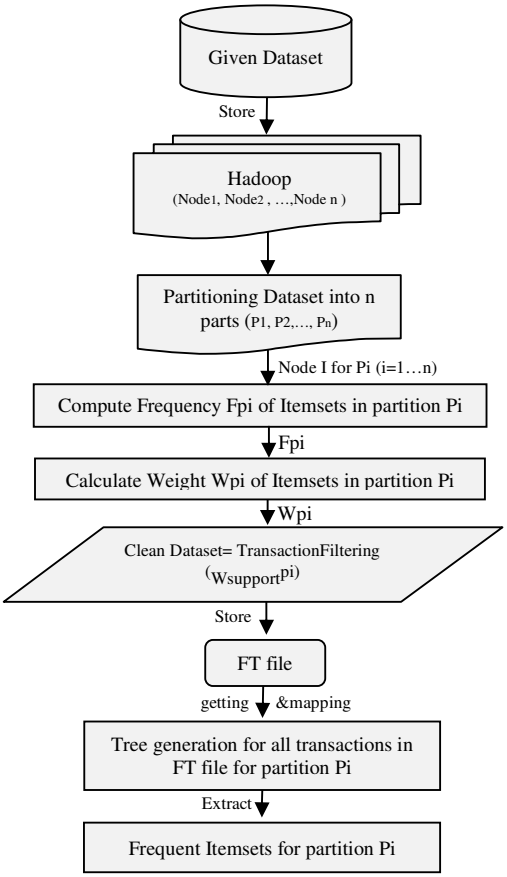


**Fig. 1. Framework of the Proposed FTWeightedHashT Algorithm**

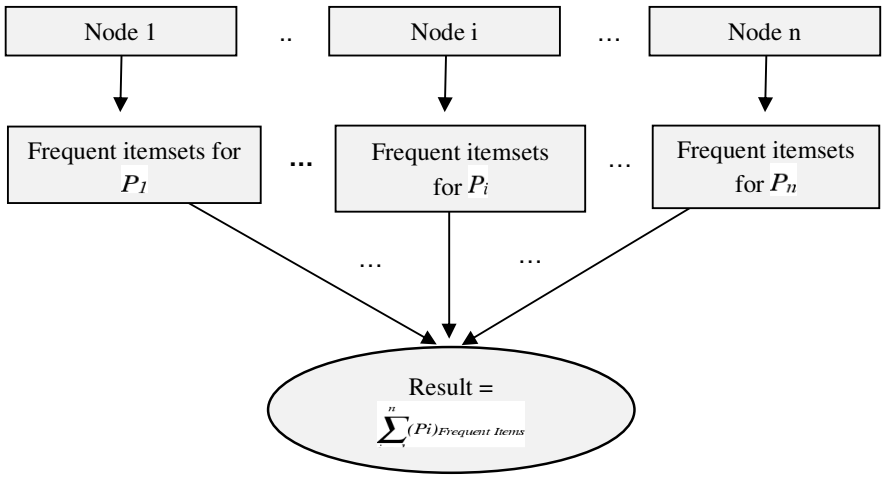


**Fig. 2. Framework of the Proposed FTWeightedHashT Algorithm for final result**

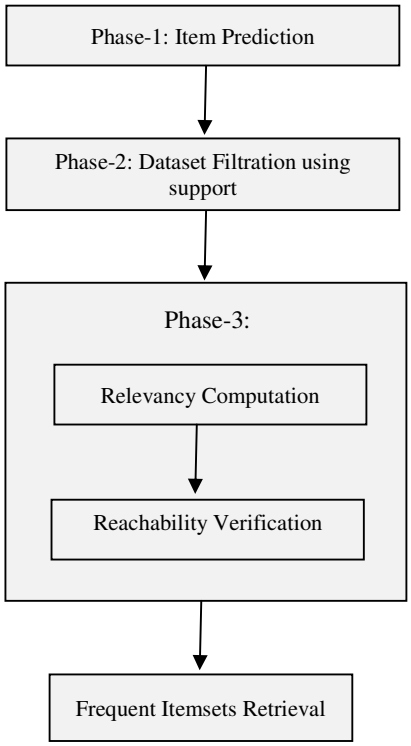


**Fig. 3. Flow diagram of the proposed algorithm**

As aforementioned, the introduced algorithm comprises of three phases. In the first phase, the algorithm scans the dataset for counting the frequency of itemsets. Here, we used a function named index() to obtain the frequency of each itemset in a particular partition of the dataset. Whereas, the second phase is to calculate the weight of itemsets by cal_weight() function and then filter the transactions by removing the infrequent items. Here all items that satisfy the support measure are decided as frequent ones. The last phase is the tree construction. The tree is constructed by generate tree() function. We followed [4] for building the tree. The tree is built by getting the filtered items. The nodes are corresponding to itemsets and having a counter to each node. It reads one transaction only at a time and mapping it to a path of fixed order. Paths can overlap when transactions are sharing items with the same prefix. If so, then counters are incremented.

# 4 Implementation and Experimental Details

To take advantage of the parallel processing that Hadoop provides, we need to submit our query as a Job of MapReduce running on the cluster of Hadoop. By using driver class, a MapReduce job is configured. Mapper class and Reducer class of the framework of MapReduce are defined beside the driver class. Also, the driver class specifies the directories of input and output in HDFS and any other problem parameters. Apriori algorithm can be implemented on MapReduce framework by splitting it into two sub-problems corresponding to map and reduce tasks independently. Each mapper processes a split in input dataset and generates local candidates with locally weighted support count. To generate frequent itemsets, each reducer receives local candidates and sums up the local count. All the mappers and reducers executed in a parallel way across different nodes of the cluster. Apriori is an iterative algorithm. So, we have to submit a job each time a new iteration starts.

The proposed FTWeightedHashT Apriori algorithm is implemented over Hadoop distributed data processing platform as MapReduce model that reflect the phases of the approach: Item Prediction phase, Transaction Filtering phase, and Tree Construction phase. The overall results of the implementation are to increase the

performance of Apriori regarding execution time. The implementation of the proposed algorithm using Hadoop-MapReduce involves the following steps:

Step1: All data preprocessing is performed on a given dataset using MapReduce paradigm. It is used as text files directories into NameNode then uploaded to HDFS which divides the input text files into data blocks of size 64 MB. The HDFS stores the metadata of each block of the NameNode, and all the data blocks in the DataNodes.

Step2: Running the proposed FTWeightedHashT Apriori algorithm over Hadoop to extract the frequent itemsets. The result is stored in a directory called "output" on HDFS.

```
Input: D = {P1, P2, …, Pi} where Pi is the partition of transactional dataset
Output: Frequent Itemsets
FTWeightedHashT Apriori Algorithm
BEGIN
(1)  TD = Read_TransactionalDataset (D);
(2)  Fi = ∅ ;   // Fi = key
(3)  FI = ∅ ;   // FI = value
     //Phase-1: Compute Frequency for FPi
(4)  For t = 1 … n Then // n = Number of Transactions
(5)  For i=1…m then   //m= Number of items in a particular transaction
(6)  FI = FI. Indexof (Ii);
(7)  FIi = FIi + 1 ;
(8)  End For i
(9)  End For t
     // Phase-2: Transaction Filtering for
(10) WVi = Cal_Weight (Fi. FI) ;  // wv= weighted value
(11) If (WVi < wsupp) then
(12) Fi . Remove (Ii);
(13) End If
     // Phase-3: Tree Construction
(14) Tree Tr= generateTree (Fi);  // Tr = the name of the tree
(15) For t = 1 … n Then // n = Number of Transactions
(16) If Tr.contains(Tt) then
(17) For each combination (denoted as CT) of Tt  // CT = combination tree
(18) Generate pattern CT U RT with wsuppt >= min (wsupptt) ;
(19) Else for each Ni∈ Tt
(20) The header of Tree {
(21) Generate pattern CT=Ni U RT with  wsuppi >= Ni . wsuppi
(22) Construct CT's conditional Frequent TreeCT;
(23) If TreeCT!= 0 then
(24) RT = CT;  // RT = result tree
(25) Repeat;
(26) }
(27) RT = CT;
(28) End IF
(29) End For t
(30) End algorithm
```

**Fig. 4. Pseudo Code of the Proposed Algorithm**

## 4.1 Dataset Description

The improved algorithm is tested for retail, and mushroom datasets which have been taken from UCI data repository. The retail itemset is having 176324 transactions, 16470 items and with a size of 4153 KB. Mushroom data is with 8124 instances, 22 attributes and with a size of 558 KB.

### 4.2 Environment Setup

The experiment is carried out in a machine with Intel core i5-G570 with 2.40GHz and RAM of 12GB.VirtualBox workstation is installed. Most of the literature works have used 2 VMs. Two virtual machines are used because one virtual machine is needed for Hadoop configuration and the other virtual machine for Eclipse that is java programming language. However, the present work has used only one virtual machine for both Hadoop and eclipse. The one VM is used for Hadoop 2.6.0 – cdh 5.10.0 setup and Eclipse 3.3.2 with the configuration of 167GB SSD hard disk and 8192 MB base memory. RedHat 64-bit is installed in it. A virtual machine is created using the Virtualbox workstation 7.1.3. Cloudera-Quickstart-VM-5.10.0-0 is the platform that the Hadoop and Eclipse worked in.

### 4.3 Hadoop Setup

This section describes the experimental environment for testing our proposed approach. We implemented both algorithms using Java programming language. The experimental environment is built on a Hadoop cluster with four nodes. One of these nodes is configured as Hadoop Master or as the NameNode which controls the data distribution over the Hadoop cluster and the other nodes acting as DataNodes.

HDFS splits dataset into 64 MB chunks each presented as a map task and then, distributes them among workers with three replications by default, the input split includes location information about the next block. HDFS stores replicas of each data block to ensure both reliability, availability, and performance [1].

A split is a logical division of the input data. However, the block is said to be a physical division of data. The default block size of HDFS is default split size if input split is not given in the code. The user defines the split, and its size can be held in the MapReduce program. Various splits can be mapping to one block and one split of various blocks. The number of map tasks (number of mappers) can specify the number of splits. Hence the number of mappers depend on file size and the block size. Block size is fixed to 64 MB here, but file size varies as per the input dataset [22].

## 5 Results and Discussion

Here, we summarize and discuss the results of the experiments that are conducted. We use Retail and Mushroom datasets. We evaluate the performance of the proposed FTWeightedHashT Apriori algorithm with the WeightedHashT Apriori algorithm of the former work [4]. We notice that, only two scans through the dataset to obtain the frequent itemsets are required (one scan for frequency of itemsets and the other scan to build the tree of frequent itemsets) rather than the WeightedHashT Apriori which takes three passes (one scan for frequency of itemsets, one scan to build the tree, and one scan to obtain the frequent itemsets). Coding has been achieved, and results are obtained.

### 5.1 Investigation of execution time for FTWeightedHashT (Retail dataset)

We measure the execution time by executing both, the improved FTWeightedHashT Apriori algorithm, and WeightedHashT Apriori algorithm on the retail dataset for different values of support, and run on four nodes. We get improved results of 60% with the comparison to WeightedHashT because of employing transaction filtering technique. Fig. 5 depicts the execution time of the retail dataset.

### 5.2 Investigation of execution time for FTWeightedHashT (Mushroom dataset)

As described in the retail dataset, we also measure the execution time of the mushroom dataset for different values of support, and get improved results of 65% compared to WeightedHashT. Fig. 6 explains the execution time of the mushroom dataset.

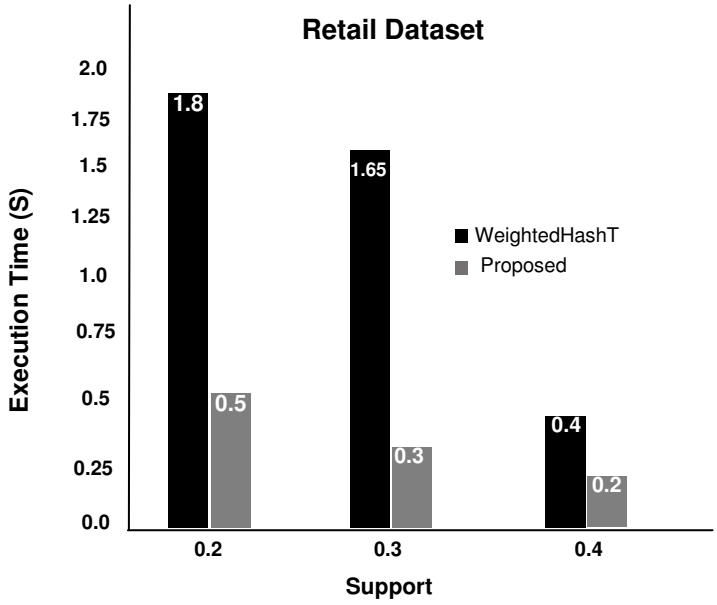


**Fig. 5. Execution time comparison with retail dataset**

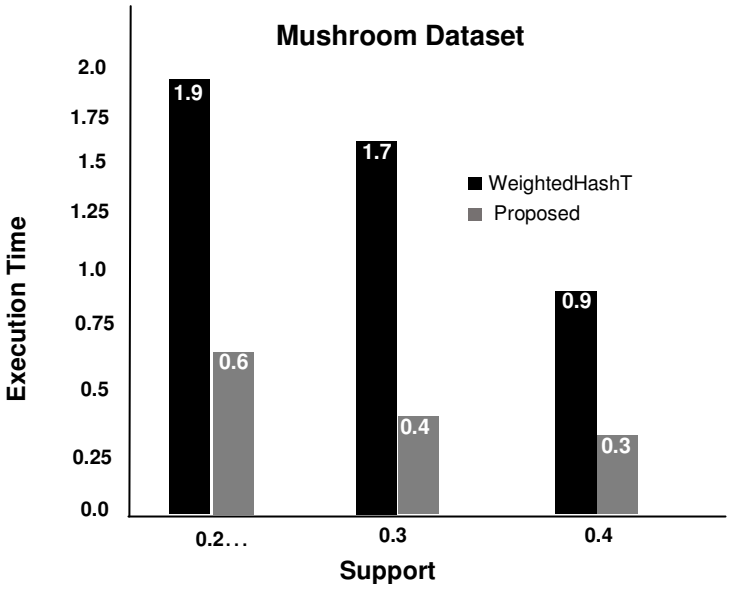


**Fig. 6. Execution time comparison with mushroom dataset**

# 6 Conclusion

Re-designing the algorithms of data mining on Hadoop environment to analyze and mine big data are the new movement towards the field of research. So, we proposed a new algorithm named "FTWeightedHashT" Apriori algorithm for extracting frequent patterns. In our experiment on Hadoop cluster, two comparison figures are given to analyze the performance of the proposed algorithm. It is made sure that the presented

work is efficient regarding time execution where the proposed algorithm obligates the process to read only the frequent itemsets from FT file instead of reading the dataset which contains both frequent and infrequent ones. A further possible studies could be done to enhance the Hadoop-based Apriori. One can apply the existing idea about auto-multi minimum supports because for numerous datasets it is difficult to pick up the value of minimum support. Very small values may give rise to rule explosion, and too large values may cause scarce item deadlock. Thus, implementation of auto-multi minimum support expected to give better results. Another thing is that we can apply the proposed algorithm to be run on Spark instead of MapReduce. Because of main memory caching mechanism that Spark has, it will give better results absolutely.

## Acknowledgements

We owe a special debt to Eng. Abdullah Al-jerafi for his patience and support as we put long hours of this work. Special recognition goes to Mr. Abdullah Al-matary whose thoughtful perspectives and attention to detail helped me to mold the work into text, and for his hours of emergency babysitting throughout the writing process. Many thanks also to Prof. Biljana Mileva Boshkoska, my great instructor in big data management online course, for giving me advice, for giving me a way to be proud of my work every day. Thanks also to my classmates in big data management online course Mr. Brent Tisdell, Mrs. Avaneesh Kumar Singh, and Mr. Arthur Francis, for online conversations that have helped me with this work so much.

## Competing Interests

Authors have declared that no competing interests exist.

## References

[1] Hurwitz J, Nugent A, Dr. Halper F, Kaufman M. Big data for dummies. Special Edition. Wiley & Sons, Inc; 2013.

[2] Han J, Kamber M, Pei J. Data mining: Concepts and techniques. 3rd ed. Elsevier; 2013.

[3] Singh S, Garg R, Mishra PK. Performance analysis of apriori algorithm with different data structures on hadoop cluster. International Journal of Computer Applications. 2015;128(9):45-51.

[4] Sumithra R, Paul S, Primary D, Latha P. A hybrid algorithm combining weighted and hash T Apriori algorithm in mapreduce model using eucalyptus cloud platform. WSEAS Transactions on Computers. 2015;14(-):382-388.
E-ISSN: 2224-2872

[5] Ezhilvathani A, Raja K. Implementation of parallel apriori algorithm on Hadoop cluster. International Journal of Computer Science and Mobile Computing. 2013;2(4):513-516.

[6] Hazarika M, Rahman M. Mapreduce based eclat algorithm for association rule mining in datamining: Mr _ Eclat. International Journal of Computer Science and Engineering. 2014;3(1):19–28.

[7] Itkar S, Kulkarni U. Distributed algorithm for frequent pattern mining using hadoop-mapreduce framework. Proceeding of International Conference on Advances in Computer Science, AETACS. 2013;15-24.

[8] Jyoti LD, Kiran BD. A novel methodology of frequent itemset mining on hadoop. International Journal of Emerging Technology and Advanced Engineering. 2014;4(7):851–859.

[9] Kovacs F, Illes J. Frequent itemset mining on Hadoop. IEEE 9th International Conference on Computational Cybernetics. 2013;241–245.

[10] Modgi M. Mining distributed frequent itemset with hadoop. International Journal of Computer Science & Information Technologies (IJCSIT). 2014;5(3):3093–3097.

[11] Moens S, Aksehirli E, Goethals B. Frequent itemset mining for big data. Big Data IEEE International Conference. 2013;111–118.

[12] Oruganti S, Ding Q, Tabrizi N. Exploring HADOOP as a platform for distributed association rule mining. The Fifth International Conference on Future Computational Technologies and Applications. 2013;62–67.

[13] Qureshi Z, Bansal S. Improving apriori algorithm to get better performance with cloud computing. International Journal of Software & Hardware Research in Engineering. 2014;2(2):33-37.

[14] Vajk I. Performance evaluation of apriori algorithm on a hadoop cluster. Wseas.Us. 2013;114–121.

[15] Vinodhini S, Vinoth M. Frequent pattern identification using map reduce paradigm. International Journal of Engineering Research & Technology. 2014;3(3):1763-1768.

[16] Agrawal R, Imieliński T, Swami A. Mining association rules between sets of items in large databases. SIGMOD Rec. 1993;22(2):207-216.
DOI: 10.1145/170036.170072.

[17] Xun Y, Zhang J, Qin X. FiDoop: Parallel mining of frequent itemsets using mapreduce. IEEE transactions on systems, man, and cybernetics: Systems. 2016;46(3):313-325.

[18] Apiletti D, et al. Frequent itemsets mining for big data: A comparative analysis. Big Data Research; 2017.
Available:http://dx.doi.org/10.1016/j.bdr.2017.06.006

[19] Singh S, Garg R, Mishra PK. Observations on factors affecting the performance of MapReduce based Apriori on Hadoop cluster. International Conference on Computing, Communication, and Automation (ICCCA). 2016;87-94.
DOI: 10.1109/CCAA.2016.7813695

[20] Borgelt C. Efficient implementations of apriori and eclat. Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'03). 2003;90 of CEUR Workshop Proceedings, Melbourne, Florida, USA.

[21] Bodon F. A trie-based APRIORI implementation for mining frequent item sequences. In Proceedings 1st international workshop on open source data mining: frequent pattern mining implementations. ACM. 2005;56-65.

[22] Fanatic Programmer. Split size vs block size. CodeThatAint Blog posted on August 19; 2016. Available:http://codethataint.com/blog/split-size-vs-block-size/



*Peer-review history:*
*The peer review history for this paper can be accessed here (Please copy paste the total link in your browser address bar)*
*http://www.sciencedomain.org/review-history/24053*