

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

Sarem M. Ammar^{1*}, Assoc.Prof. Fadl M. Ba-Alwi²

¹Departement of IT - Yemen Academic for Graduate Studies – Yemen

²Faculty of Computer & IT - Sana'a University – Yemen

ABSTRACT

The most significant problem of data mining is mining frequent itemset on big datasets. The best-known basic algorithm for mining frequent itemset is Apriori. Due to the drawbacks of Apriori algorithm, many improvements have 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 manifest that the proposed algorithm is efficient and effective in term of execution time.

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 are literally living within the data age. Terabytes or petabytes of data pours 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 kind of data source that manages a huge 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 in a keep 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 referred to as data mining. Data mining is the method to achieve new models for data analysis so as to urge edges of business [2]. It's additionally a method of discovering unknown and helpful information from giant databases. Data mining methods involve seven

common categories of data mining techniques consistent with operate 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 therefore the alternative purposeful information. Among them, association rules square measure the foremost common one. Several business enterprises accumulate giant quantities of data from 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 the 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 for association rule mining on big data. The most used algorithm is Apriori. To address the drawbacks of Apriori, a number of methods have proposed such as [3][4][5-15]. The common drawback in these algorithms is a delay in execution time. We have noted that the latest study [4] is efficient to improve the performance of Apriori on MapReduce implementation in term 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. With respect to their study, we consider that there is a delay in execution time especially when the data increasing exponentially. Increasing the volume of data with various types; structured, unstructured, and/or semi-structured needs to be processed in a fast way by using a different architecture than the existed ones. A distributed architecture is able to 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 boundaries such that, we focused only 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 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 to enhance the scalability [3-15] [17-19].

Y. Xun et. al. [17] designed a parallel algorithm for mining frequent itemset named FiDooP using the model of MapReduce. To obtain squeezed storage and avert constructing conditional pattern bases, their proposed algorithm FiDooP integrates frequent items ultrametric tree, instead of classic FP trees. To complete the mining mission in FiDooP, only three MapReduce jobs are executed. In the pivotal third MapReduce job, the itemsets are

decomposed independently by the mappers, small ultrametric 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 over the cluster of computing nodes. Their developed metric called FiDooP-HD, which is expansion of FiDooP, to improve the performance of mining for elevated-dimensional data analysis. The reason behind their proposed paper is to resolve automatic parallelization and load balance challenges in the present parallel algorithms for frequent itemset mining.

D. Apiletti et. al. [18] did 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 relation between the performance of the algorithms and its strategies of parallelization.

S. Singh et. al. [19] introduced six factors which affect the performance 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 homogenous and heterogeneous. The factors are certain to jointly algorithmic and nonalgorithmic refinements. Treated factors certain to algorithmic refinements are data structures and transaction filtering. While the nonalgorithmic factors involve the nodes of bad performance, locality of data and distribution of data blocks, reflective execution, and the control of parallelism with input split size. They claimed that these factors are important to influence the performance.

Singh et. al. [3] has executed the Apriori algorithm on Hadoop for three central structures which are trie, hash tree, and hash table trie. They evaluated the time execution for each one of these data structures. This study considers these structures in Java to be used in MapReduce code. They stated that 'In sequential computing environment, trie outperformed hash tree experimentally as well as theoretically. Hash table trie theoretically accelerates the Apriori algorithm but experimentally could not success. 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.'

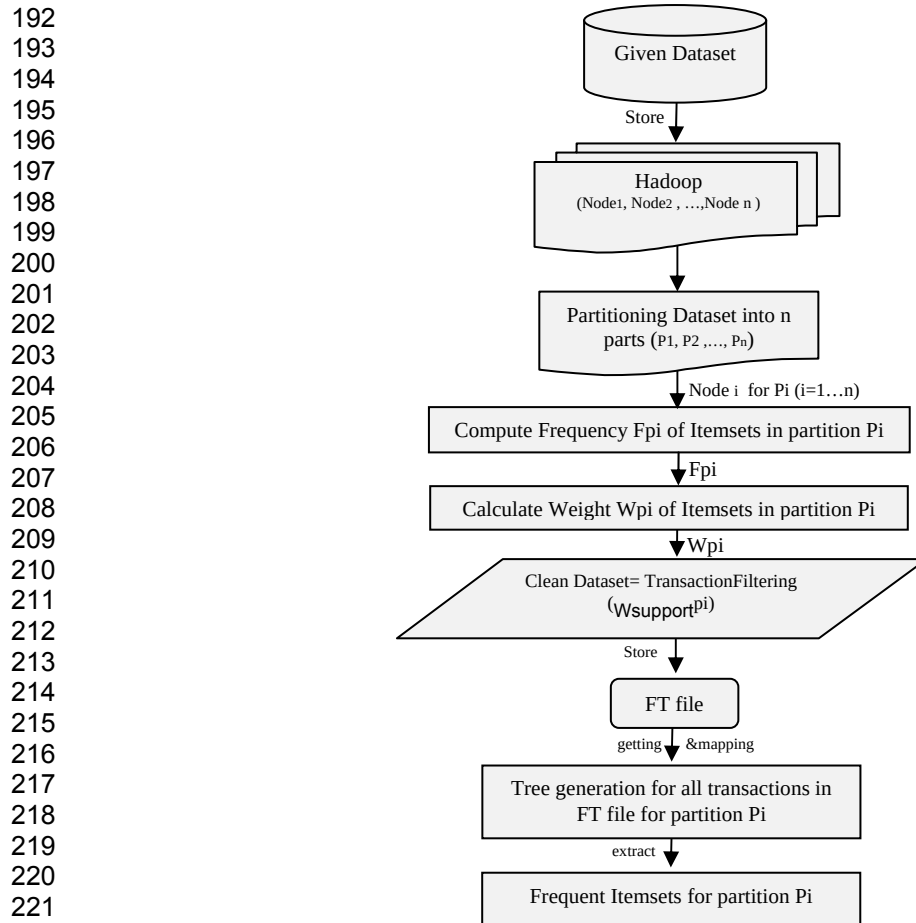
However, Sumithra et. al. [4] have proposed hybrid Apriori algorithm which combined the interests of jointly Hash tree Apriori algorithm and Weighted based-Apriori 'tree-based hybrid approach'. In their proposed algorithm, the selected datasets have large numbers of transactions. In these transactions, it is necessary to presage and expect the items in the given dataset. For expecting the frequent itemsets and associations in 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 on implementation of new improved WeightedHashT Apriori algorithm that called as 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 consumes more time to execute because of multi scans of dataset where the scanning process include frequent and infrequent itemsets. To address this problem, we proposed a novel approach named "FTWeightedHashT" (Filtered Transaction Weighted Hash Tree). The framework (Figures 1-2) of the proposed algorithm works as follows. A given dataset will be processed in parallel way in Hadoop cluster with n nodes. Figure 1 shows the different phases of the proposed algorithm for j^{th} node. At the first step, the entire dataset will be stored in HDFS. JobTracker will partition the dataset into n small parts ($P_1, P_2, \dots, P_n$). TaskTracker will assign each part P_i ($i=1,2,\dots,n$) to j^{th} node. At the second step, each partition P_i ($i=1,2,\dots,n$), will be scanned (i.e. first scan) to compute the frequency of itemsets (denoted by F_{Pi}). The obtained F_{Pi} is used to compute the weight of itemsets (denoted by W_{Pi}) 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 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 in efficient parallel implementation of Apriori using Hadoop-MapReduce model. The transaction filtering phase is used to scan the partition P_i at the j^{th} node to determine all frequent itemsets. In 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 dataset. This will address the main drawback of the consumed time calculated by the hybrid WeightedHashT algorithm [4] (i.e. wasting time on 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 fixed order in the tree (i.e. Second scan) as fifth step. These steps will be repeated until all transactions in j^{th} node are covered to get the frequent itemsets for the partition P_i and return back 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 back to the application as shown in Figure 2.

3.1 Design of Proposed FTWeightedHashT Apriori Algorithm

FTWeightedHashT Apriori algorithm contains three main phases: Item Prediction phase, Filtration phase, and Tree Construction phase (see Figure 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 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 $wsupport$ less than the minimum $wsupport$ value). The last phase contains two sub-phases namely relevancy computation and reachability verification. These sub-phases depicts for constructing frequent tree with both types of relation, direct and indirect. The relevancy computation (direct relationship) is the process of expecting the relevance between nodes in the tree that means the itemsets in the tree of frequent itemset. The reachability verification (indirect relationship) refers to the ability of reaching the the end point of the transaction for each itemset in the tree. More explanation of tree generation is in the next subsection.



223 **Figure 1: Framework of the Proposed FTWeightedHashT Algorithm**

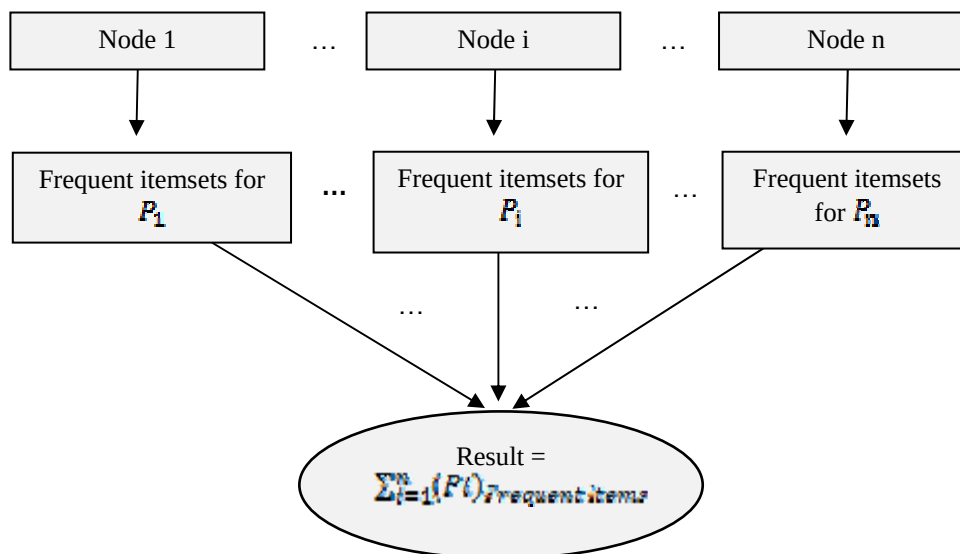


Figure 2: Framework of the Proposed FTWeightedHashT Algorithm for final result

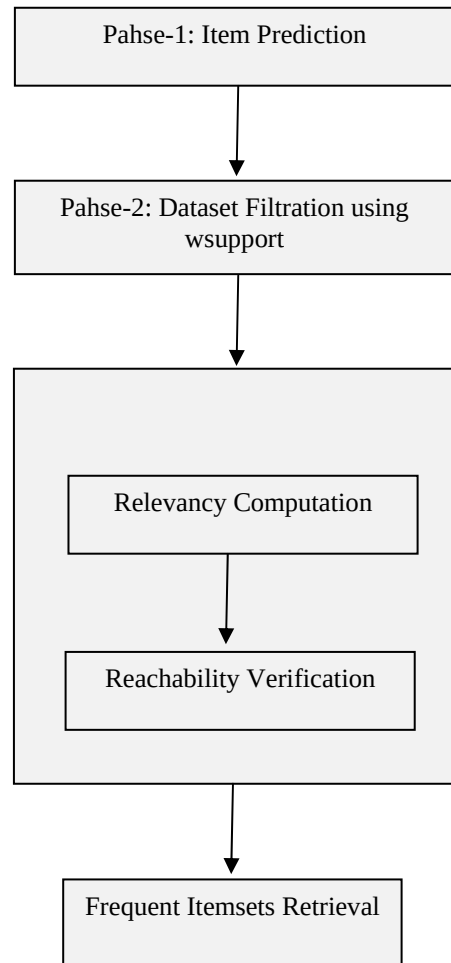


Figure 3: Flow Diagram of the Proposed Algorithm

The proposed algorithm mines the transactional dataset that is deemed to be the data that can be distributed. The algorithm uses weighted support (wsupport) measure instead of using support count measure. The diagram above describes the way of explaining the dialectics of the algorithm formulation. It iterates for 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 reduce task as shown in Figure 2.

3.2 Pseudo Code of the Proposed FTWeightedHashT Apriori Algorithm

The phases presented in the flow diagram of Figure 3 are implemented in Java programming language as given in Figure 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 followws.

```

298
299
300 Input:  $D = \{p_1, p_2, \dots, p_i\}$  where  $p_i$  is the partition of transactional dataset
301 Output: Frequent Itemsets
302 FTWeightedHashT Apriori Algorithm
303 BEGIN
304 (1)..... TD = Read_TransactionalDataset (D);
305 (2).....  $F_i = \emptyset$  ; //  $F_i$  = key
306 (3).....  $F_i = \emptyset$  ; //  $F_i$  = value
307           //Phase-1: Compute Frequency for  $F_i$ 
308 (4)..... For  $t = 1 \dots n$  Then //  $n$  = Number of Transactions
309 (5).....   For  $i=1 \dots m$  then //  $m$ = Number of items in a particular transaction
310 (6).....      $F_i = F_i \cup \text{Indexof}(F_i)$ ;
311 (7).....      $F_i = F_i + 1$  ;
312 (8).....   End For  $i$ 
313 (9)..... End For  $t$ 
314           // Phase-2: Transaction Filtering for
315 (10).....  $WV_i = \text{Cal\_Weight}(F_i, F_i)$  ; //  $wv$ = weighted value
316 (11)..... If ( $WV_i < \text{wsupp}$ ) then
317 (12).....    $F_i \cdot \text{Remove}(F_i)$ ;
318 (13)..... End If
319           // Phase-3: Tree Construction
320 (14)..... Tree  $T_r = \text{generateTree}(F_i)$ ; //  $Tr$  = the name of the tree
321 (15)..... For  $t = 1 \dots n$  Then //  $n$  = Number of Transactions
322 (16).....   If  $T_r$  contains( $T_r$ ) then
323 (17).....     For each combination (denoted as CT) of  $T_r$  // CT = combination tree
324 (18).....       Generate pattern CT U RT with  $\text{wsupp}_t \geq \min(\text{wsupp}_t)$  ;
325 (19).....   Else for each  $N_i \in T_r$ 
326 (20).....     Header of Tree {
327 (21).....       Generate pattern CT= $N_i$  U RT with  $\text{wsupp}_t \geq N_i \cdot \text{wsupp}_t$ 
328 (22).....       Construct CT's conditional Frequent TreeCT;
329 (23).....       If TreeCT!= 0 then
330 (24).....         RT = CT; // RT = result tree
331 (25).....       Repeat;
332 (26).....     }
333 (27).....   RT = CT;
334 (28)..... End IF
335 (29)..... End For  $t$ 
336 (30)..... End algorithm

```

Figure 4: Pseudo Code 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 indexof() to obtain the frequency of each itemset in a particular partition of dataset. Whereas the second phase is to calculate the weight of itemsets by cal_weight() function, and then filters the transactions by removing the infrequent items. Here all items that satisfy the wsupport measure are decided as frequent ones. The last phase is the tree construction. The tree is constructed by generatetree() 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 for 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.

351 4. IMPLEMENTATION AND EXPERIMENTAL DETAILS

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

364 The proposed FTWeightedHashT Apriori algorithm is implemented over Hadoop distributed
365 data processing platform as MapReduce model that reflects the phases of the approach:
366 Item Prediction phase, Transaction Filtering phase, and Tree Construction phase. The
367 overall results of the implementation is to increase the performance of Apriori in term of
368 execution time. The implementation of the proposed Apriori algorithm using Hadoop-
369 MapReduce involves the following steps:

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

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

376

377 4.1 Dataset Description

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

381

382 4.2 Environment Setup

383 The experiment is carried out in a machine with Intel core i5-G570 with 2.40GHz and RAM of
384 12GB. VirtualBox workstation is installed. Most of Literature works have used 2 VMs. Two
385 virtual machines are used because one virtual machine is needed for Hadoop configuration
386 and the other virtual machine for Eclipse that is java programming language. However, the
387 present work has used only one virtual machine for both Hadoop and eclipse. One VM is
388 used for Hadoop 2.6.0 – cdh 5.10.0 setup and Eclipse 3.3.2 with the configuration of 167GB
389 SSD hard disk and 8192 MB base memory. RedHat 64-bit is installed in it. Virtual machine
390 is created using VirtualBox workstation 7.1.3. Cloudera-quickstart-vm-5.10.0-0 is the platform
391 which the Hadoop and Eclipse work in.

392

393 4.3 Hadoop Setup

394 This section describes the experimental environment for testing our proposed approach. We
395 implement the proposed FTWeightedHashT Apriori algorithm using Java programming
396 language. The experimental environment is built on a Hadoop cluster with 4 nodes. One of
397 these nodes is configured as Hadoop Master or as the NameNode which controls the data
398 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 distribute them among workers with 3 replications by default, the input split includes location information for 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 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 depends on file size and the block size. Block size is fixed as 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 improved FTWeightedHashT Apriori algorithm with respect to the time executed of the former work [4]. We notice that only two scans through the dataset to obtain the frequent of itemsets are required (one scan for frequency of itemsets and the other scan to build the tree of frequent itemsets) rather than the algorithm in [4] which take three passes (one scan for frequency of itemsets, one scan to build the tree, and one scan to obtain the frequent itemsets).

5.1 Investigation of Execution Time for FTWeightedHashT (Retail dataset)

We measure the execution time by execute the improved FTWeightedHashT Apriori algorithm on Retail dataset with wsupport 0.3 and run on 4 nodes. The time consumed in WeightedHashT was 9900 milliseconds while in the proposed algorithm is 2170 milliseconds. We get improved results about 60% with comparison to WeightedHashT because of employing transaction filtering technique. Figure 5 depicts the execution time with Retail dataset.

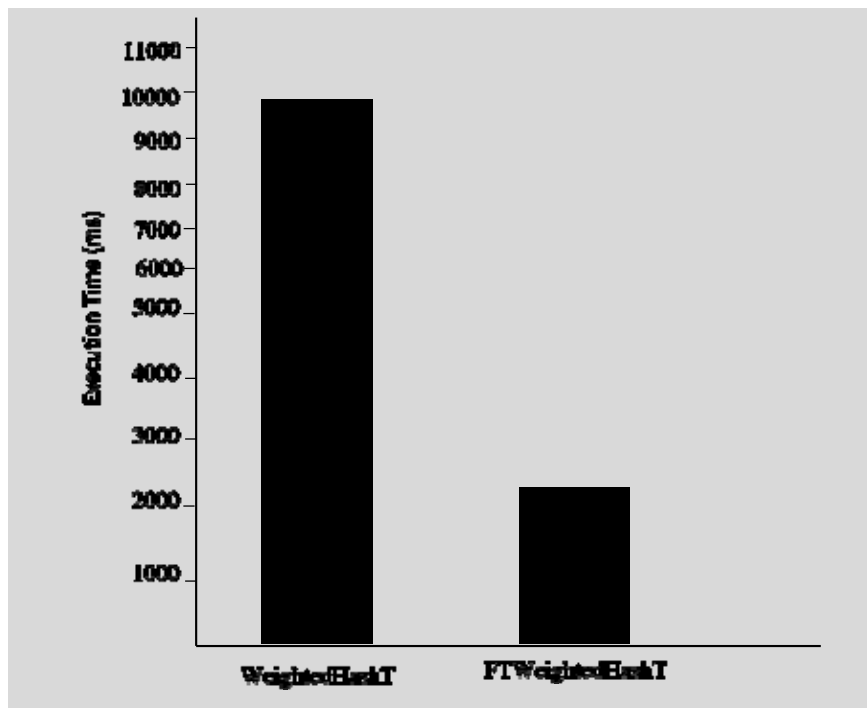


Figure 5: Execution time Comparison with Retail dataset

5.2 Investigation of Execution Time for FTWeightedHashT (Mushroom dataset)

As described in Retail dataset, we also measure the execution time with Mushroom dataset and get improved results about 65% compared to WeightedHashT. The time consumed in WeightedHashT was 10200 ms while in the proposed algorithm is 2458 ms. Figure 6 explains the execution time with Mushroom dataset.

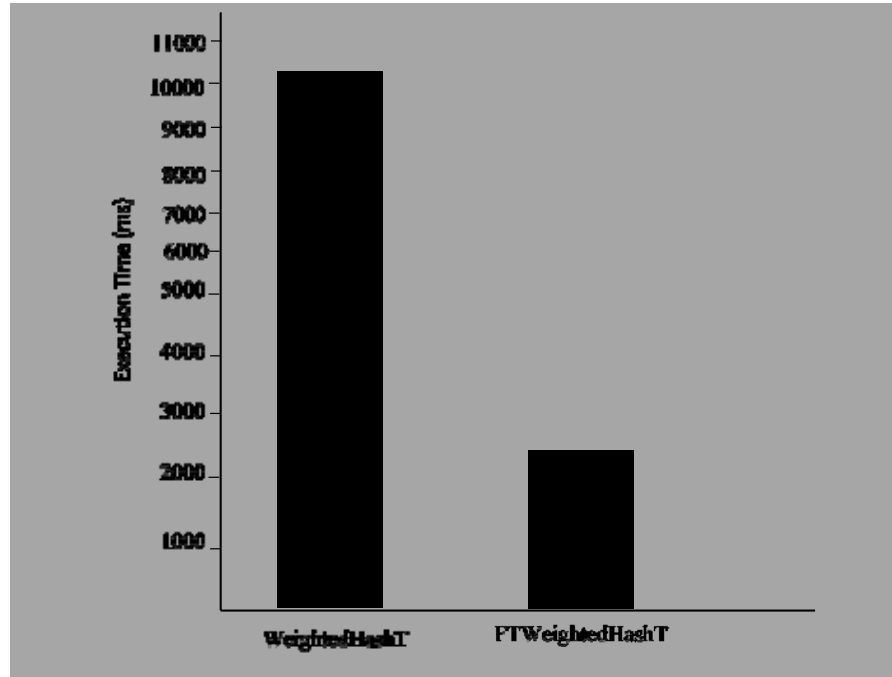


Figure 6: Execution time Comparison with Mushroom dataset

6. CONCLUSION

Re-designing the algorithms of data mining on Hadoop environment to analyze and mine big data is the new movement in the field of research. So we proposed a new algorithm called "FTWeightedHashT" Apriori algorithm for extracting frequent patterns. In our experiment on Hadoop cluster, two comparison figures are given to analyze the performance and it is made sure that the presented work is sufficient in term of time execution where the proposed algorithm obligates the process to read only the frequent itemsets from FT file instead of reading in each iteration from the dataset which contains both frequent and infrequent ones. A further possible study could be done in order to enhance the Hadoop-based Apriori. One can apply the existing idea with auto-multi minimum support 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, implement auto-multi minimum support expected to give a better result. 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 into this work. Special recognition goes to Mr. Abdullah Al-matary whose thoughtful perspectives and attention to detail helped me to mold the work into a 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 advise, 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.

AUTHORS' CONTRIBUTIONS

'Author A' 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 B' managed the analyses of the study, review the results and the all proposed work in general (as a supervisor of author 1 on master's degree). The final manuscript read and approved by all authors.

REFERENCES

1. Judith Hurwitz, Alan Nugent, Dr. Fern Halper, and Marcia Kaufman. Big Data For Dummies. Special Edition. Wiley & Sons, Inc; 2013.
2. Jiawei Han, Micheline Kamber, and Jian Pei. Data Mining: Concepts and Techniques. 3rd ed. Elsevier; 2013.
3. Sudhakar Singh, Rakhi Garg, P.K.Mishra. Performance Analysis of Apriori Algorithm with Different Data Structures on Hadoop Cluster. International Journal of Computer Applications. 2015; 128(9):45-51.
4. R. Sumithra, Sujni Paul and D. Ponmary Pushpa Latha. 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, L. D., & Kiran, B. D. 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.

- 552 10. Modgi, M. P. Mining Distributed Frequent Itemset with Hadoop. International Journal of
553 Computer Science & Information Technologies (IJCSIT). 2014; 5(3):3093– 3097.
- 554 11. Moens, S., Aksehirli, E., & Goethals, B. Frequent Itemset Mining for Big Data. Big Data
555 IEEE International Conference. 2013; 111–118.
- 556 12. Oruganti, S., Ding, Q., & Tabrizi, N. Exploring HADOOP as a Platform for Distributed
557 Association Rule Mining. The Fifth International Conference on Future Computational
558 Technologies and Applications. 2013; 62–67.
- 559 13. Qureshi, Z., & Bansal, S. Improving Apriori Algorithm to get better performance with
560 Cloud Computing. International Journal of Software & Hardware Research in
561 Engineering. 2014; 2(2):33-37.
- 562 14. Vajk, I. Performance Evaluation of Apriori Algorithm on a Hadoop Cluster. Wseas.U.S.
563 2013; 114–121.
- 564 15. Vinodhini, S., & Vinoth, M. Frequent Pattern Identification using Map Reduce Paradigm.
565 International Journal of Engineering Research & Technology. 2014; 3(3):1763-1768.
- 566 16. Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between
567 sets of items in large databases. SIGMOD Rec. 1993; 22(2): 207-216.
568 DOI=10.1145/170036.170072.
- 569 17. Yaling Xun, Jifu Zhang, and Xiao Qin. FiDooP: Parallel Mining of Frequent Itemsets
570 Using MapReduce. IEEE transactions on systems, man, and cybernetics: systems.
571 2016; 46(3): 313-325.
- 572 18. D Apiletti et. al., Frequent Itemsets Mining for Big Data: A Comparative Analysis. Big
573 Data Research. 2017; <http://dx.doi.org/10.1016/j.bdr.2017.06.006>
- 574 19. S. Singh, R. Garg and P. K. Mishra. Observations on factors affecting performance of
575 MapReduce based Apriori on Hadoop cluster. International Conference on Computing,
576 Communication and Automation (ICCCA). 2016; 87-94. doi:
577 10.1109/CCAA.2016.7813695
- 578 20. Christian Borgelt. Efficient Implementations of Apriori and Eclat. Proceedings of the IEEE
579 ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'03). 2003; volume
580 90 of CEUR Workshop Proceedings, Melbourne, Florida, USA.
- 581 21. Ferenc Bodon. A trie-based APRIORI implementation for mining frequent item
582 sequences. In Proceedings 1st international workshop on open source data mining:
583 frequent pattern mining implementations, ACM. 2005; 56-65.
- 584 22. Fanatic Programmer. Split Size vs Block Size. CodeThatAint Blog posted on August 19,
585 2016. <http://codethataint.com/blog/split-size-vs-block-size/>
586

587

588

589

590

591