

Improved FTWeightedHashT Apriori Algorithm on 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

Mining frequent itemset on big datasets is almost being the most important problem on data mining. Apriori is the best-known basic algorithm for frequent itemset mining. Due to the drawbacks of Apriori algorithm, many improvements were 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 study, 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 a sea of data and the level of the sea is increasing rapidly. So-seeming, we are actually living in the data age. Terabytes or petabytes of data pours into our computer networks, the World Wide Web (WWW), and various data storage devices every day from business, society, science and engineering, medicine, and almost every other aspect of daily life. This explosive growth of available data volume is a result of the computerization of our society and the fast development of powerful data collection and storage tools. Big data is the science of analyzing high volumes of diverse data in near-real time [1]. The computerization of our society has substantially enhanced our capabilities for both generating and collecting data from diverse sources. A tremendous amount of data has flooded almost every aspect of our lives. This explosive growth in stored 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 useful information and knowledge. This has led to the generation of a promising and brilliant frontier in computer science called data mining. Data Mining is the method to gain new models for data analysis in order to get benefits of business [2]. It is also a process of discovering unknown and useful information from large databases. Data mining methods involve seven common classes of data mining techniques according to function and application purpose: outlier detection, association rule mining (ARM), classification, clustering, regression, summarization, and mining time-series

data. The need for data mining algorithms is to handle and process the massive data. Big Data algorithms analyze the huge amount of information, reveal the association rules, hidden patterns, trends and the other meaningful information. Among them, association rules is the most popular one. Many business enterprises accumulate large quantities of data from their day-to-day operations. For example, huge amounts of customer purchase data are collected daily at the checkout counters of grocery stores. Retailers are interested in analyzing the data to learn about the purchasing behavior of their customers. Such valuable information can be used to support a variety of business-related applications such as marketing promotions, inventory management, and customer relationship management. To address the aforementioned issues, association analysis comes in, which is useful for discovering interesting relationship hidden in a large datasets. The uncovered relationships can be represented in the form 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, 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 on 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 fast way by using 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 parallel way. Thus, the proposed work aims to improve the algorithm presented in [4] by reducing the complexity of time execution 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 in 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]. A number of Apriori based algorithms have been proposed to minimize the execution time, reduce the memory consumption, and to enhance the scalability [3][4][5-15][17-19].

Y. Xun et. Al. [17] designed a parallel frequent itemsets mining algorithm called FiDooP using the MapReduce programming model. To achieve compressed storage and avoid building conditional pattern bases, their proposed algorithm, FiDooP, incorporates the frequent items ultrametric tree, rather than conventional FP trees. In FiDooP, three MapReduce jobs are implemented to complete the mining task. In the crucial third MapReduce job, the mappers independently decompose itemsets, the reducers perform combination operations by constructing small ultrametric trees, and the actual mining of these trees separately. They implemented FiDooP on their in-house Hadoop cluster. They also developed a workload balance metric to measure load balancing across the cluster's computing nodes. Their developed metric called FiDooP-HD, an extension of FiDooP, to

90 speed up the mining performance for high-dimensional data analysis. The reason behind
91 their proposed algorithm is to solve the scalability and load balancing challenges in the
92 existing parallel mining algorithms for frequent itemsets.

93
94 D. Apiletti et. Al. [18] did a comparative analysis which reviewed Hadoop- and Spark-based
95 scalable algorithms addressing the frequent itemset mining problem in the Big Data domain
96 through both theoretical and experimental comparative analyses. The main contributions of
97 their survey are the following: 1) A theoretical analysis of the algorithmic choices that have
98 been proposed to address the itemset mining problem in the Big Data context by means of
99 MapReduce, with the analysis of their expected impact on main memory usage, load
100 balancing, and communication costs. 2) An extensive evaluation campaign to assess the
101 reliability of our expectations. Precisely, we ran more than 300 experiments on 14 synthetic
102 datasets and 2 real datasets to evaluate the execution time, load balancing, and
103 communication costs of five state-of-the-art parallel itemset mining implementations. 3) The
104 identification of strengths and weaknesses of the algorithms with respect to the input dataset
105 features (e.g., data distribution, average transaction length, number of records), and specific
106 parameter settings. 4) The discussion of promising open research directions for the
107 parallelization of the itemset mining problem.

108
109 S. Singh et. Al. [19] introduced six factors which affecting performance of MapReduce based
110 Apriori on Hadoop cluster. They have investigated the factors that significantly affects the
111 execution time of MapReduce based Apriori running on homogenous and heterogeneous
112 Hadoop clusters. Factors are specific to both algorithmic and non-algorithmic improvements.
113 Considered factors specific to algorithmic improvements are filtered transactions and data
114 structures. The non-algorithmic include speculative execution, nodes with poor performance,
115 data locality & distribution of data blocks, and parallelism control with input split size. They
116 claimed that these factors are important that influence the performance.

117
118 Singh et. Al. [3] has implemented the Apriori algorithm on MapReduce framework for the
119 central structures hash tree, trie, and hash table trie and evaluated the execution time of
120 Apriori algorithms for these data structures in MapReduce context. This study considers
121 three data structures in Java to be used in MapReduce code. In sequential computing
122 environment, trie outperformed hash tree experimentally as well as theoretically. Hash table
123 trie theoretically accelerates the Apriori algorithm but experimentally could not success. In
124 their experiment on Hadoop cluster, trie outperforms hash tree as usual but hash table trie is
125 outstanding among the three data structures for both real-life and synthetic datasets. Their
126 paper discussed the execution times of three algorithms on dataset T10I4D100K with
127 minimum support of 0.02.

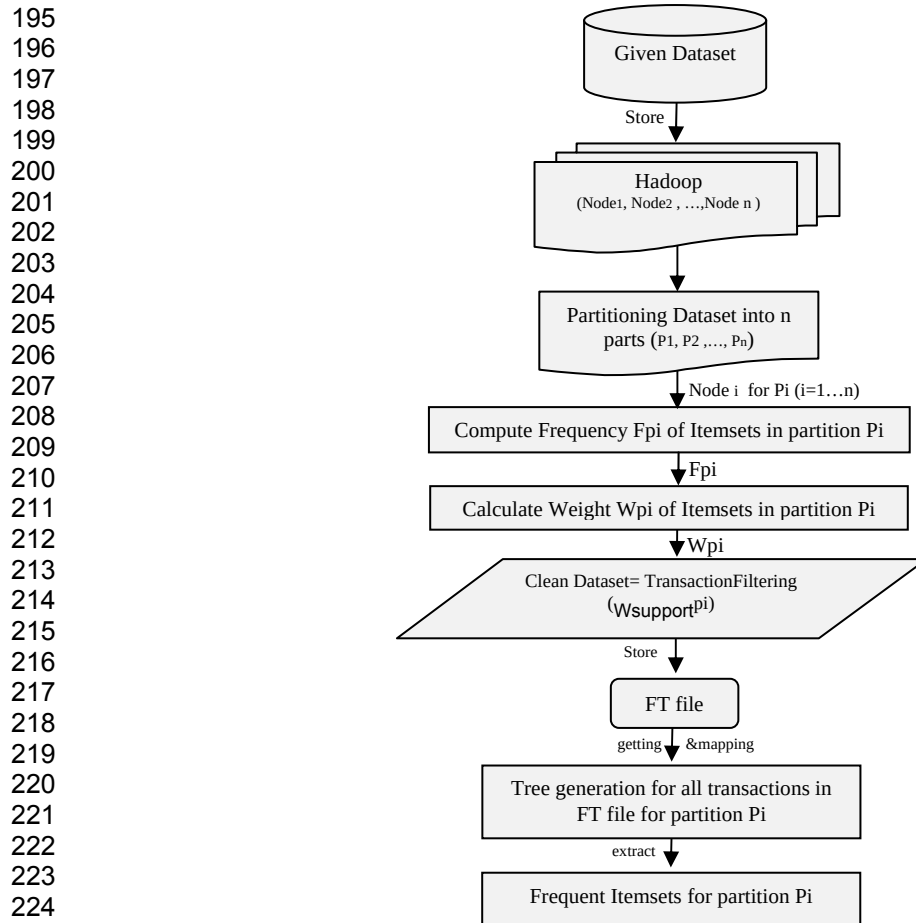
128
129 However, Sumithra et. al. [4] have proposed hybrid Apriori association rule algorithm which
130 combined the benefits of both hash tree and weighted Apriori algorithms "its tree-based
131 hybrid approach". In their proposed system, the selected synthetic data sets contain
132 numerous numbers of transaction. From these transactions, it is needed to predict the items
133 in the transactional dataset. For predicting the frequent items and association rules in the
134 transactional dataset, it generally consume lot of memory storage and time. To overcome
135 the problem, their new proposal computes the frequent pattern without the generation of
136 candidate itemsets. Tree based mining approach is used to reduce the consumption of
137 memory usage in the association rule prediction during reducing phase. It uses correlation-
138 based approaches to optimize the process. Their work conclude on implementation of new
139 improved WeightedHashT Apriori algorithm which termed as hybrid algorithm in Hadoop-
140 MapReduce environment. In their proposed algorithm, they selected retail and mushroom
141 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 FP_i). The obtained FP_i is used to compute the weight of itemsets (denoted by WP_i) for transaction filtering as the third step. The way of calculating WP_i is similar to [4]. 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 itemsets obtained by removing infrequent itemsets 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 itemset 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 itemsets). 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 itemset 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 relevance of the itemset with other itemsets while considering overall unique transactions; then filters the itemsets by removing infrequent itemsets (i.e. itemsets 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 direct and indirect relationship. The relevancy computation (direct relationship) is the process of predicting the relationship between the nodes in the tree which means the itemsets in the frequent itemset tree. The reachability verification (indirect relationship) refers to the ability to reach the end point of the transaction of each itemset in the tree. More explanation of tree generation is in the next subsection.



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

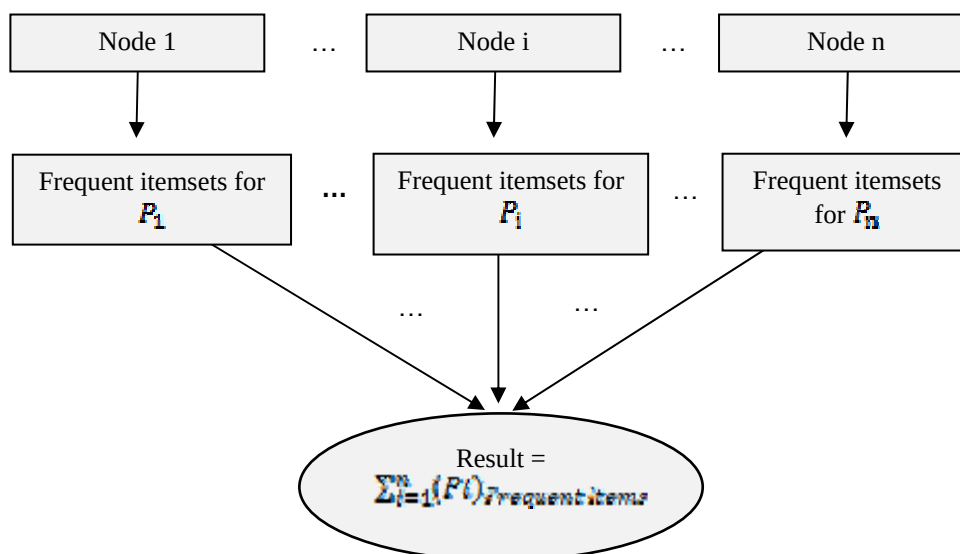


Figure 2: Framework of the Proposed FTWeightedHashT Algorithm for final result

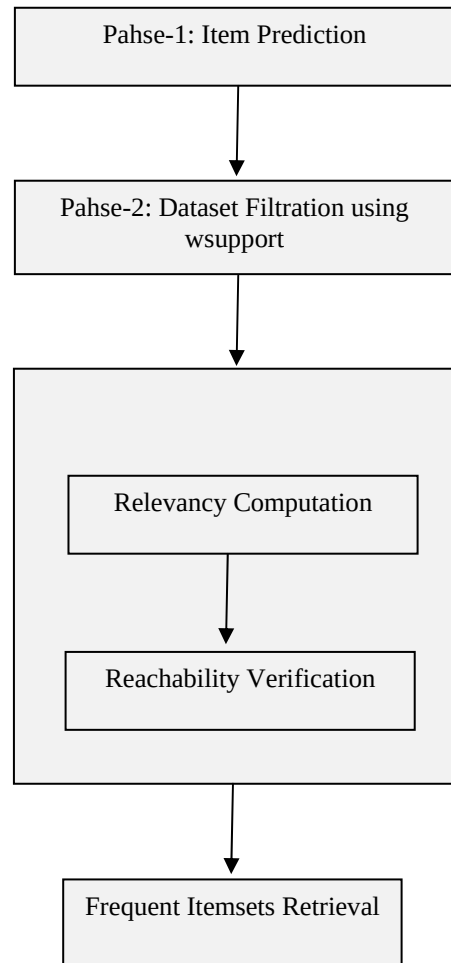


Figure 3: Flow Diagram of the Proposed Algorithm

The proposed algorithm mines the transactional dataset which is considered to be the data for distribution. The algorithm uses weighted support (wsupport) measure instead of using support count measure. The diagram above depicts the way of representing the logic of the algorithm formation. 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 follows:

```

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

```

Figure 4: Pseudo Code of the Proposed Algorithm

As aforementioned, the proposed algorithm consists 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 items by removing the infrequent items. Here all itemsets that satisfy the wsupport measure are decided as frequent itemsets. 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 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.

354 4. IMPLEMENTATION AND EXPERIMENTAL DETAILS

355 To take advantage of the parallel processing that the Hadoop provides, we need to submit
356 our query as a MapReduce Job on Hadoop cluster. A job is configured using a Driver class.
357 A driver class is defined with a Mapper class and a Reducer class of MapReduce
358 framework. Also input/output directory in HDFS, and other problem specific parameters are
359 specified in driver class. To implement Apriori algorithm on MapReduce framework we have
360 to split it into two independent sub-problems corresponding to map and reduce tasks. Each
361 mapper processes a chunk of input datasets and generates local candidates with local
362 weighted support count. Each reducer receives local candidates and sums up the local count
363 to generate frequent itemsets. All the mappers and reducers execute in parallel across
364 different nodes of cluster but final result cannot be obtained until all reduce tasks are not
365 completed. Apriori is an iterative algorithm so we have to submit job each time a new
366 iteration starts.

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

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

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

379

380 4.1 Dataset Description

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

384

385 4.2 Environment Setup

386 The experiment is carried out in a machine with Intel core i5-G570 with 2.40GHz and RAM of
387 12GB. VirtualBox workstation is installed. Most of Literature works have used 2 VMs, one VM
388 for Hadoop and the other VM for Eclipse. However, the present work has used only 1 VM for
389 both Hadoop and Eclipse. The VM is used for Hadoop 2.6.0 – cdh 5.10.0 setup and Eclipse
390 3.3.2 with the configuration of 167GB SSD hard disk and 8192 MB base memory. RedHat
391 64-bit is installed in it. Virtual machine are created using VirtualBox workstation 7.1.3.
392 Cloudera-quickstart-vm-5.10.0-0 is the platform which the Hadoop and Eclipse work in.

393

394 4.3 Hadoop Setup

395 This section describes the experimental environment for testing our proposed approach. We
396 implement the proposed FTWeightedHashT Apriori algorithm using Java programming
397 language. The experimental environment is built on a MapReduce cluster with 4 nodes. One
398 of these nodes is configured as Hadoop Master or as the NameNode which controls the data
399 distribution over the Hadoop cluster and the other nodes acting as DataNodes.

400 HDFS splits dataset into 64 MB chunks each presented as a map task and then distribute
401 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. Split is a logical division of the input data while block is said to be a physical division of data. HDFS default block size is default split size if input split is not specified in the code. Split is said to be user defined and its size can be controlled in the MapReduce program. One split can be mapping to multiple blocks and there may be multiple split of one block. The number of splits specify the number of map tasks (Mapper). 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.

411
412

413 5. RESULTS AND DISCUSSION

414

415 Here, we summarize and discuss the results of the experiments that are conducted. We use
416 Retail and Mushroom datasets. We evaluate the performance of the improved
417 FTWeightedHashT Apriori algorithm with respect to the execution time of the former work
418 [4]. We notice that only two passes over the dataset to obtain the frequent itemsets are
419 required (one scan for frequency of itemsets and the other scan to build the tree of frequent
420 itemsets) rather than the algorithm in [4] which take three passes (one scan for frequency of
421 itemsets, one scan to build the tree, and one scan to obtain the frequent itemsets).

422

423 5.1 Investigation of Execution Time for FTWeightedHashT (Retail dataset)

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

430

431

432

433

434

435

436

437

438

439

440

441

442

443

444

445

446

447

448

449

450

451

452

453

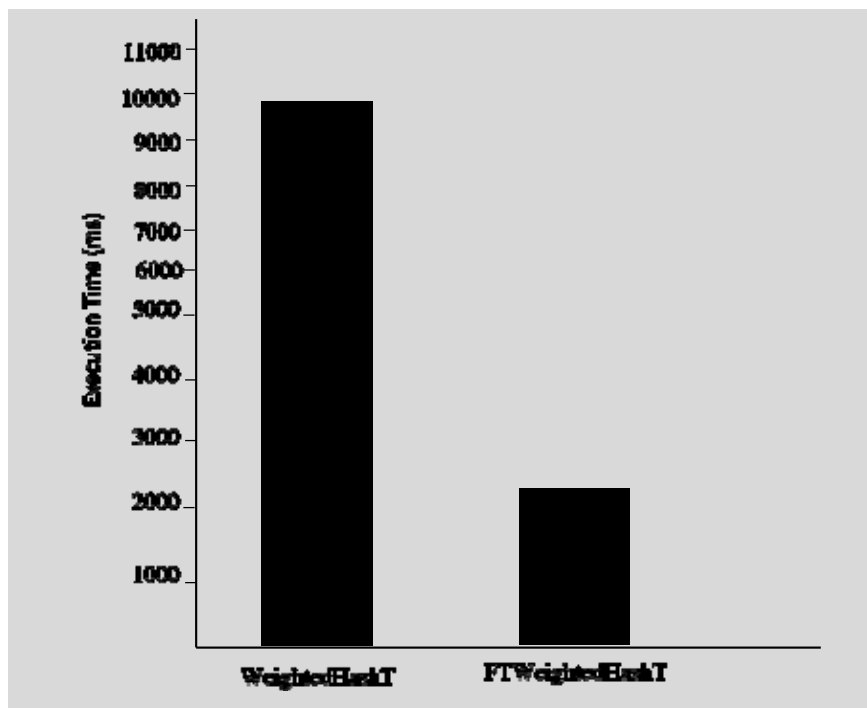


Figure 5: Execution time Comparison with Retail dataset

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.

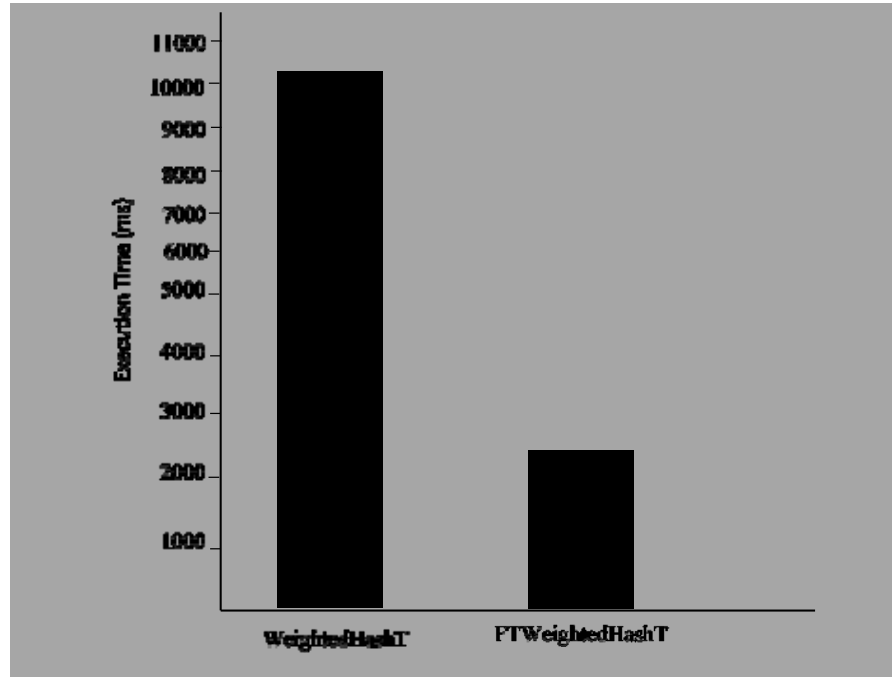


Figure 6: Execution time Comparison with Mushroom dataset

6. CONCLUSION

Re-designing data mining algorithms on MapReduce paradigm to analyze and mine big data is the new movement in research. So we proposed a novel approach called "FTWeightedHashT" Apriori algorithm for extracting frequent patterns. In our experiment on Hadoop cluster, various comparison charts are given to analyzed the performance and it is made sure that the presented work is the best 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. Further possible study could be done in order to improve and enhance of the Hadoop-based Apriori. One can apply the existing idea with auto-multi minimum support because for many datasets it is difficult to choose the value of minimum support. Too small values may cause rule explosion, and too large values may cause rare item deadlock. Thus, implement auto-multi minimum support expected to give 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 Proff. 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.

COMPETING INTERESTS

Declaration of competing interest should be placed here. All authors must disclose any financial and personal relationships with other people or organizations that could inappropriately influence (bias) their work. Examples of potential conflicts of interest include employment, consultancies, honoraria, paid expert testimony, patent applications/registrations, and grants or other funding. If no such declaration has been made by the authors, SDI reserves to assume and write this sentence: "Authors have declared that no competing interests exist."

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). All authors read and approved the final manuscript."

REFERENCES

1. A.L.Sayath Saabith, Elankovan Sundararajan, Azuraliza Abubaker, PARELLEL IMPLEMENTATION OF APRIORI ALGORITHMS ON THE HADOOP-MAPREDUCE PLATFORM- AN EVALUATION OF LITERTURE, JATIT, Vol.85(3), (March-2016) pp.321-351.
2. Jiawei Han, Micheline Kamber, and Jian Pei, Data Mining: Concepts and Techniques, Elsevier, Third Edition (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 Vol.128(9) (Oct-2015) pp.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 Vol.14 . E-ISSN: 2224-2872, (2015) pp.382-388.

- 555 5. Ezhilvathani, A., & Raja, K. Implementation of parallel apriori algorithm on hadoop cluster.
556 International Journal of Computer Science and Mobile Computing. Vol. 2(4), (2013) pp.
557 513-516.
- 558 6. Hazarika, M., & Rahman, M. Mapreduce Based Eclat Algorithm for Association Rule
559 Mining in Datamining : Mr _ Eclat. International Journal of Computer Science and
560 Engineering, Vol. 3(1), (2014) pp.19-28..
- 561 7. Itkar, S., & Kulkarni, U. Distributed Algorithm for Frequent Pattern Mining using Hadoop-
562 Map Reduce Framework. Proceeding of International Conference on Advances in
563 Computer Science, AETACS, (2013) pp.15-24.
- 564 8. Jyoti, L. D., & Kiran, B. D. A Novel Methodology of Frequent Itemset Mining on Hadoop.
565 International Journal of Emerging Technology and Advanced Engineering, Vol. 4(7),
566 (2014) pp. 851-859.
- 567 9. Kovacs, F., & Illes, J. Frequent itemset mining on hadoop., 2013 IEEE 9th International
568 Conference on Computational Cybernetics, (2013) pp. 241-245.
- 569 10. Modgi, M. P. Mining Distributed Frequent Itemset with Hadoop. International Journal of
570 Computer Science & Information Technologies(IJCSIT), Vol5(3), (2014) pp. 3093-
571 3097.
- 572 11. Moens, S., Aksehirli, E., & Goethals, B. Frequent Itemset Mining for Big Data. Big Data,
573 2013 IEEE International Conference on, (2013). pp. 111-118.
- 574 12. Oruganti, S., Ding, Q., & Tabrizi, N. Exploring HADOOP as a Platform for Distributed
575 Association Rule Mining. In FUTURE COMPUTING 2013, The Fifth International
576 Conference on Future Computational Technologies and Applications (2013). pp. 62-67.
- 577 13. Qureshi, Z., & Bansal, S. Improving Apriori Algorithm to get better performance with
578 Cloud Computing. International Journal of Software & Hardware Research in
579 Engineerin, Vol. 2(2), (2014). pp. 33-37.
- 580 14. Vajk, I. Performance Evaluation of Apriori Algorithm on a Hadoop Cluster. Wseas.Ua,
581 (2013). pp.114-121.
- 582 15. Vinodhini, S., & Vinoth, M. Frequent Pattern Identification using Map Reduce Paradigm.
583 International Journal of Engineering Research & Technology, Vol. 3(3), (2014). pp.
584 1763-1768.
- 585 16. Rakesh Agrawal, Tomasz Imieliński, and Arun Swami, "Mining association rules between
586 sets of items in large databases", SIGMOD Rec. 22, 2, 207-216.
587 DOI=10.1145/170036.170072. (June 1993).
- 588 17. Yaling Xun, Jifu Zhang, and Xiao Qin, FiDooP: Parallel Mining of Frequent Itemsets
589 Using MapReduce, IEEE transactions on systems, man, and cybernetics: systems, vol.
590 46, no. 3, march 2016.
- 591 18. D Apiletti et. Al., Frequent Itemsets Mining for Big Data: A Comparative Analysis, Big
592 Data Research (2017), <http://dx.doi.org/10.1016/j.bdr.2017.06.006>
- 593 19. S. Singh, R. Garg and P. K. Mishra, "Observations on factors affecting performance of
594 MapReduce based Apriori on Hadoop cluster," 2016 International Conference on
595 Computing, Communication and Automation (ICCCA), Greater Noida, India, 2016, pp.
596 87-94. doi: 10.1109/CCAA.2016.7813695
- 597 20. Christian Borgelt, Efficient Implementations of Apriori and Eclat, In B. Goethals and M. J.
598 Zaki, editors, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining
599 Implementations (FIMI'03), volume 90 of CEUR Workshop Proceedings, Melbourne,
600 Florida, USA, (2003).
- 601 21. Ferenc Bodon, "A trie-based APRIORI implementation for mining frequent item
602 sequences," in Proceedings 1st international workshop on open source data mining:
603 frequent pattern mining implementations, ACM, (2005).