A Novel Nodesets-Based Frequent Itemset Mining Algorithm for Big Data using MapReduce

Original Scientific Paper

Borra Sivaiah
Research Scholar,
Department of Computer Science and Engineering, Jawaharlal Nehru Technological University, Kakinada, Andra Pradesh, India,
CMR College of Engineering &Technology, Hyderabad
sivabetld@gmail.com

Ramisetty Rajeswara Rao
Professor of CSE,
Department of Computer Science and Engineering, Jawaharlal Nehru Technological University, Gurajada, Andra Pradesh, India
raob4u@jntukucev.ac.in

Abstract *– Due to the rapid growth of data from different sources in organizations, the traditional tools and techniques that cannot handle such huge data are known as big data which is in a scalable fashion. Similarly, many existing frequent itemset mining algorithms have good performance but scalability problems as they cannot exploit parallel processing power available locally or in cloud infrastructure. Since big data and cloud ecosystem overcomes the barriers or limitations in computing resources, it is a natural choice to use distributed programming paradigms such as Map Reduce. In this paper, we propose a novel algorithm known as A Nodesets-based Fast and Scalable Frequent Itemset Mining (FSFIM) to extract frequent itemsets from Big Data. Here, Pre-Order Coding (POC) tree is used to represent data and improve speed in processing. Nodeset is the underlying data structure that is efficient in discovering frequent itemsets. FSFIM is found to be faster and more scalable in mining frequent itemsets. When compared with its predecessors such as Node-lists and N-lists, the Nodesets save half of the memory as they need only either pre-order or post-order coding. Cloudera's Distribution of Hadoop (CDH), a MapReduce framework, is used for empirical study. A prototype application is built to evaluate the performance of the FSFIM. Experimental results revealed that FSFIM outperforms existing algorithms such as Mahout PFP, Mlib PFP, and Big FIM. FSFIM is more scalable and found to be an ideal candidate for real-time applications that mine frequent itemsets from Big Data.*

Keywords*: Big Data, Frequent Itemset Mining (FIM), MapReduce Programming Paradigm (MRPP), Fast and Scalable Frequent Item set Mining (FSFIM)*

1. INTRODUCTION

Frequent Itemset Mining (FIM) is a phenomenon in data mining used to extract frequently occurring items that exhibit latent relationships in the data. FIM leads to the generation of association rules that provide Business Intelligence (BI) when interpreted by domain experts. Association rule mining is the process of finding patterns, associations, and correlations among sets of items in a database. The Association Rules generated have an antecedent and a consequent. An Association Rule is a pattern of the form $X \wedge Y \Longrightarrow Z$ [support, confidence], where X, Y, and Z are items in the dataset. The left-hand side of the rule $X \wedge Y$ is called the antecedent of the rule and the right-hand side Z is called the consequent of the rule. Within the dataset, confidence and support are two measures to determine the certainty or usefulness of each rule. Support is the probability that a set of items in the dataset contains both the antecedent and consequent of the rule i.e $P\ (X \cup\ Y \cup\ Z)$. Confidence is the probability that a set of items containing the antecedent also contains the consequent. i.e., $P\ (Z/(X \cup\ Y)$

Many FIM algorithms that came into existence are classified into Apriori-based and pattern-growth-based algorithms. These algorithms cannot exploit the parallel processing power of cloud data centers.

Therefore, they suffer from the capability of dealing with big data. Big Data is data that has characteristics such as volume, variety, and velocity as shown in Fig.1. Big Data is voluminous (often measured in petabyte-scale), and has a variety of data such as structured, unstructured, and semi-structured besides having continuous growth or streaming data. The importance of Big Data processing signifies that when big data (complete data) is not considered, it results in biased conclusions. The amount of data being generated by different sources of Big Data such as social media, World Wide Web (WWW), and Internet of Things (IoT) use cases is unprecedented and essentially needs a specialized modus operandi to mine it efficiently to arrive at Business Intelligence (BI). There are two types of approaches used in frequent item sets mining of Big Data: Apriori-based and FP-Growth based. The Apriori-based algorithms consume more memory and time in the generation of frequent item sets from Big Data. The FP-Growth-based algorithms were developed for faster generation of the frequent itemsets instead of the Aprori-based algorithm. Therefore, FP-growth-based algorithms are faster than the Apriori-based approaches. However, FP-growth algorithms also suffer from storing conditional FP-trees in memory and then mining from them may require more time and memory. The drawbacks of the existing works:

1. Some of the existing frequent item sets mining algorithms consumed more memory and more mining time.
2. Some of the existing algorithms don't use distributed programming paradigms to solve the scalability problem.

To address the existing drawbacks, the proposed research is introduced. The main purpose of the research in this article is to develop an efficient MapReduce-based algorithm for faster and scalable discovery of frequent itemsets from big data. The proposed method needs distributed programming frameworks like Hadoop for mining big data as explored in Section 1.2.

1.1. MOTIVATION

Many researchers developed frequent pattern algorithms for handling Big Data. The algorithms are divided into two categories: Apriori-based and tree-based. Tree-based algorithms are faster than Apriori-based algorithms. The traditional tools and techniques cannot handle big data, due to their limited capabilities. Frequent itemset mining algorithms are faster but cannot use local or cloud-based parallel processing power. Researchers used parallel and distributed Hadoop MapReduce systems to quickly generate frequent item sets. Large data makes it difficult to extract frequent item sets from transactional databases, if extracted faster, then it could be helpful for better decision-making. Using the best data structures can reduce half of the memory and execution time of frequent item generation from big data. When a case study like healthcare is considered, there is ever-growing data size leading to big data. Unless there is faster convergence in frequent itemset mining, it takes more time for frequent itemset mining. In many applications, there is a need for quick convergence. It is the motivation behind the research carried out and presented in this paper. Our work has focused on building a new algorithm for mining frequent item sets based on the best data structure known as Nodeset and tree structure known as POC tree. Towards this end, we proposed a framework for FIM that is significantly faster than existing methods. Our contributions to this paper are as follows.

1. A novel algorithm known as Fast and Scalable Frequent item set mining (FSFIM) is proposed to extract frequent item sets from big data.
2. A Pre-Order Coding (POC) tree is used to represent data and improve speed in processing leading to improved performance.
3. A prototype is built to evaluate the FSFIM and compare it with the state-of-the-art FIM techniques.

The remainder of the paper is structured as follows. Section 2 reviews the literature on FIM techniques, especially parallelized ones. Section 3 presents the proposed algorithm and the underlying mechanisms and data structures. Section 4 presents experimental results and discussion. Section 5 concludes the work in the paper and gives suggestions for improving the research further in the future.

1.2. CLOUDERA'S DISTRIBUTION OF HADOOP

Apache Hadoop framework is widely used in the real world for processing big data. Cloudera's Distribution of Hadoop (CDH) is based on the Apache Hadoop framework. It supports the MapReduce paradigm where the number of worker nodes in the distributed environment acts as a mapper and reducer.

Hadoop Distributed File System (HDFS) plays a crucial role in the framework as it stores inputs and outputs. HDFS can access data from various servers computed distributed across the globe. The framework involves a job tracker and task tracker to keep track of a given job and underlying tasks respectively. The map phase is carried out by thousands of worker nodes and the results are given to reducers (worker nodes). On the other hand, the reducers act on the data to produce the final output.

Fig. 2 shows the MapReduce phenomenon which is essentially meant for dealing with large volumes of data. It supports parallel processing to process data faster. Input data is divided into many pieces and assigned to worker nodes that complete a given job and return the results.

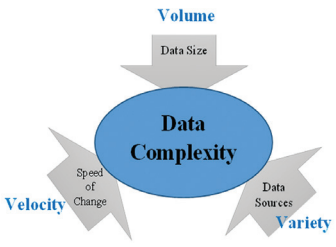

Fig. 1. Illustrates characteristics of big data

2. RELATED WORK

Literature is found rich in the research on frequent itemset mining algorithms. Many FIM algorithms that target big data came into existence which are briefly presented in this section.

Qiu *et al.* [1] proposed a parallel FIM algorithm known as Yet Another Frequent Itemset Mining (YAFIM) which is based on Apriori. They defined the YAFIM algorithm to run using a distributed programming framework known as Spark.

Yasir *et al.* [2] proposed an FIM algorithm to handle sparse big data. They named it TRimmed Transaction LattICE (TRICE). It generates trimmed subsets iteratively to leverage performance in terms of memory consumption and execution time. They intended to improve it to have other variants of frequent itemsets such as maximal frequent itemsets besides working on streaming data.

Gole and Tidke [3] proposed a MapReduce-based method for FIM on big data. It is named ClustBigFIM. It is derived from the BigFIM algorithm, for better speed and scalability.

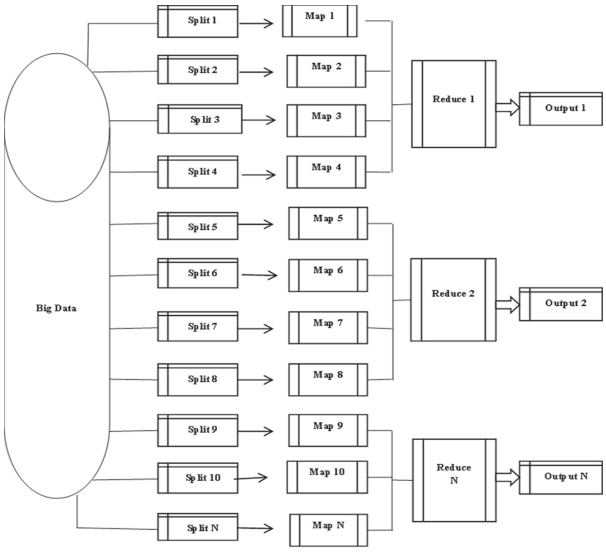

Fig. 2. Map Reduce Frame Work in Hadoop

Djenouri *et al.* [4] proposed two FIM algorithms namely the Enhanced Approach for Single Scan approach for Frequent Itemset Mining (EA-SSFIM) and MapReduce Single Scan approach for Frequent Itemset Mining (MR-SSFIM) for big data. They are designed to deal with sparse big data and big data respectively for improvements in terms of reducing execution time and improving efficiency in processing big data.

Apiletti *et al.* [5] review different FIM algorithms that are developed for big data. Fernandez-Basso *et al.* [6] proposed a distributed method for FIM which is meant for extracting frequent itemsets from streaming data. Sethi and Ramesh [7] proposed an FIM algorithm known as Hybrid Frequent Itemset Mining (HFIM) that works in two phases. In the first phase, it extracts frequent itemsets and in the second phase, it obtains frequent itemsets of k-cardinality where k is greater than or equal to 2. It could improve speedup and execution time. Chon *et al.* [8] proposed a Graphics Processing Unit (GPU) based FIM known as GMiner which is much faster as it could exploit the power of GPU.

Liang and Wu [9] proposed a distributed FIM algorithm known as Sequence-Growth. It makes use of a lexicographical sequence tree that follows the idea of lexicographical order. It also has a pruning strategy known as breadth-wide support-based which makes it scalable and efficient. In the future, they intend to make it an incremental FIM algorithm.

Joy and Sherly [10] proposed a parallel FIM algorithm known as Faster-IAPI that is executed using the Spark RDD environment. It is employed to have symptom correlations in patients' data in the healthcare domain for disease prediction. Djenouri *et al.* [11] proposed three versions of High-Performance Computing (HPC) that make use of a single database scan. They are known as Single Scan on GPU (GSS), Single Scan on Cluster (CSS), and Single Scan on Cluster and GPU (CGSS). Out of which CGSS was found to have better performance.

A distributed FIM algorithm known as BIGMiner has been proposed in [12] which is scalable and causes less network communication overhead. This algorithm exploits the GPU and MapReduce programming model to have significantly improved performance.

Raj *et al.* [13]. have proposed an EAFIM method for an efficient apriori-based frequent itemset mining algorithm on Spark for big transactional data. Spark is gaining attention in big data processing due to its in-memory processing capabilities. EAFIM uses parallel and distributed computing environments and introduces two novel methods to improve efficiency. Unlike apriori, candidate generation and count of support values occur simultaneously during input dataset scanning. The updated input dataset is calculated by removing useless items and transactions, reducing the size of the input dataset for higher iterations.

Moens *et al.* [14] investigate the usage of the MapReduce paradigm to execute different FIM algorithms. They studied two such algorithms such as BigFIM and Dist-Eclat thoroughly. They found that MapReduce models outperform their predecessors. Xun *et al.* [15] proposed parallel FIM based on Hadoop clusters. The algorithm is characterized by its data partitioning approach that paves the way for a locality-based approach to enhance the performance of the algorithm.

Asbern and Asha [16] explored different algorithms for FIM that operate on big data using the MapReduce paradigm. Kumar and Mohbey [17] investigated different parallel FIM algorithms that are executed in distributed environments. Different issues they identified in such algorithms include scalability, privacy, complex data types, load balancing, and gene regulation patterns.

Zitouni *et al.* [18] proposed a parallel FIM known as CloPN. It follows a prime number-based approach for FIM from big data. It is supposed to mine closed fre-

quent itemsets (CFI). Leung *et al.* [19] on the other hand proposed an alternative data mining approach for big data. It is known as scalable vertical mining using Spark. Galetsi *et al.* [20] studied big data analytics associated with the healthcare domain. They investigated on different machine learning techniques used for data analytics including FIM algorithms.

Fernandez-Basso *et al.* [21] defined a fuzzy mining approach to have FIM on big data. They implemented a method known as the automatic fuzzification method for this purpose. It takes weather forecast data and generates association rules with energy efficiency using the Spark environment.

Fumarola and Malerba [22] proposed a method known as approximate FIM that uses parallel processing using the MapReduce paradigm. Djenouri *et al.* [23] proposed a parallel framework for FIM using a metaheuristic approach. It is known as Cluster for FIM (CFIM). At nodes in the cluster, the algorithm partitions data. The framework is integrated with different metaheuristics such as Genetic Algorithm (GA), BSO (Bees Swarm Optimization), and Particle Swarm Optimization (PSO) for better performance.

Aggarwal *et al.* [24] investigated air quality data with location and time awareness for performing FIM. First, it understands spatiotemporal dependencies in the data and then employs the FIM process to generate frequent itemsets. Luna *et al.* [25] proposed different parallel versions of Apriori to work on big data using the MapReduce paradigm. Their Hadoop-based implementation showed better performance than the existing methods. From the literature, it is understood that parallel approaches used for FIM can perform better than traditional FIM methods. However, the performance achieved due to distributed environments is insufficient as the underlying method for FIM is expected to have a better approach. Towards this end, we proposed a framework for FSFIM that is significantly faster than existing methods.

S. Nalousi *et al.* [26] introduced a novel efficient approach called weighted frequent itemset mining using weighted subtrees (WST-WFIM) to identify the average weight of frequent rules. The average weight of found rules is calculated using special trees and some novel data structures on the frequent pattern growth (FP-Growth) method. It works with the data set that each item in each transaction has a certain weight and saves them in the dedicated tree.

Fayuan Li *et al.* [27], Based on the calculation of item set fuzziness, this approach incorporates the unpredictability of potential world models to tackle the problem of mining fuzzy frequent item sets based on probability threshold. Fuzzy theory and uncertainty are based on linguistic information and have been expanded to cope with partial truth concepts. A dynamic programming-based approach is used to compute the frequent fuzzy probability.

TR-FC-GCM (Transaction Reduction - Frequency Count - Generate Combination Method) created by Ajay Sharma et al [28] discovers all significant frequent patterns by creating all potential combinations of an item with a single database search and performs better for null and full datasets. B Sivaiah et al [29], Reviewed Incremental mining, which aims to extract patterns from dynamic databases that have applications in domains such as product recommendation, text mining, market basket analysis, and web click stream analysis.

Reshu Agarwal [30] suggested a method for finding high average-utility item sets (HAUIs) that takes into account both the length of the itemsets and their utilities. HUIs are found using the standard method based on the individual utility of an item set, which is calculated as the sum of the utilities of individual items. The difficulty is that the aforementioned method of computing HUIs does not take the length of the item set into account.

Wanyong Tian *et al.* [31] suggested a technique for uncertain frequent item sets called UFP-ECIS (Uncertain Frequent Pattern Mining with Ensembled Conditional Item-wise Supports). The difficulties of information redundancy and loss caused by a single probabilistic frequent threshold can be successfully improved by assembling numerous conditional item-wise supports. Furthermore, by employing several pruning algorithms based on the sorted downward closure feature and the concept of least minimal probability frequent threshold. Many existing frequent itemset mining algorithms have good performance but scalability problems as they cannot exploit parallel processing power available locally or in cloud infrastructure. Since big data and cloud ecosystem overcomes the barriers or limitations in computing resources, it is a natural choice to use distributed programming paradigms such as Map Reduce.

3. FAST AND SCALABLE FREQUENT ITEMSET MINING (FSFIM) ALGORITHM

The proposed algorithm is known as Fast and Scalable Frequent Itemset Mining (FSFIM) used to extract frequent item sets from big data. The data representation before discovering frequent itemsets is made using POC-tree. Therefore, the construction of the POC tree is an important part of the FSFIM. However, POC construction is made after producing frequent 1-itemsets. Based on the minimum support (statistical measure to know the quality of frequent pattern), after scanning the entire database, a set of frequent 1-itemsets, denoted as F1, with corresponding support is generated. Then the items in F1 are ordered using support-descending order. The ordered items are denoted as L1 where the frequent items that have the same support are just taken in any order. Then POC tree is constructed as follows. The first root of the tree, denoted as Tr, is created but it is set to "null". Then for every transaction in the given database frequent itemsets, in the order of F1, are selected and sorted. Let [p|P] denote a sorted frequent item list where the first element is denoted as p while others are de-

noted as P. Then the insert tree function is carried out. If the transaction has as child node N with the same item name as that of p, N's count is increased by 1, if not new node N is created and its count is initialized to 1. Then it is added to the transaction's children list. As far as P has some items, the insert tree is invoked recursively to complete POC tree construction. Afterward, the POC tree is scanned to, following pre-order traversal, have a pre-order of each node in the tree. Once the POC tree is constructed, it is possible to find all frequent 2-itemsets and corresponding node sets. Afterward, frequent k(>2)-itemsets are discovered. The algorithm has a pruning strategy known as promotion which depends on the notion of superset equivalence property. To represent each search space, the algorithm uses a set-enumeration tree as shown in Fig. 3.

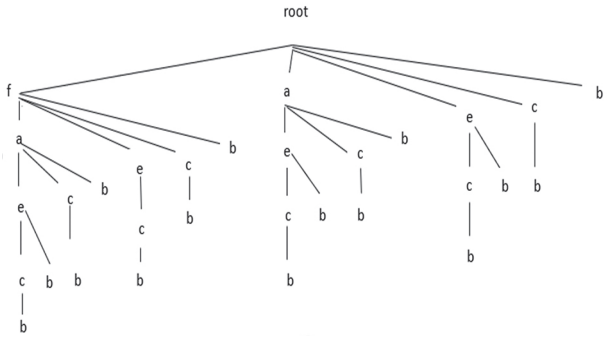

Fig. 3. Illustrates a sample set-enumeration tree

3.1. POC-TREE

The node sets data structure used in this paper is based on the POC (Pre-Order Coding) tree. Unlike its predecessor PPC-tree which needs encoding of nodes with pre-order and post-order codes causing overhead, POC-tree needs only either pre-order or post-order and not both. Since it is efficient, in this paper, we adapted it from [26]. POC-tree can be understood based on the data in Table 1 and the POC-tree representation in Fig.4. The tree has a root node labeled "null" and many subtrees where each node is prefixed by an item. Each node in the tree has different fields like item-name, children-list, count, and pre-order. Item node refers to an item the node represents. Count indicates the number of transactions denoted by the path till this node. Children-list refers to the children of the node and pre-order is the node's pre-order code. The POC tree is similar to that of the PPC tree where each node in the POC tree is encoded by its pre-order while the PPC tree has each node encoded by its pre-order and post-order. After generating the POC tree, the node sets are created. Once node sets are created, the POC tree is not required. The transactional items are provided in support-descending order in the last column of the table. The POC-tree representation helps in the faster generation of frequent itemsets. In other words, it accelerates the frequent itemset mining process. When there are large volumes of data, in the presence of cloud computing resources, it may be of use to leverage performance benefits.

Table 1. Shows a simple transactional database

ID	Ordered Frequent Items	Support-Descending Items
1	a, c, g, f	c, f, a
2	e, a, c, b	b, c, e, a
3	e, c, b, i	b, c, e
4	b, f, h	b, f
5	b, f, e, c, d	b, c, e, f

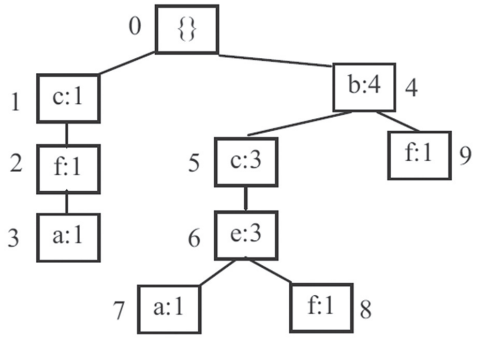

Fig. 4. POC-tree constructed for the data in Table

The FSFIM algorithm for mining Big Data works as follows: FSFIM takes transactional database T and minimum support threshold th as inputs. It produces frequent itemsets (results) R. Step 1 constructs the POC tree as discussed earlier in this section. Step 3 discovers all frequent 1-itemsets. Step 3 starts an iterative process for each node in the POC tree. Step 5 extracts the item associated with the visiting node in the POC tree. Step 5 through Step 11, there is an iterative process used to obtain all frequent 2-itemsets into F2. Steps 12 through 18, prunes infrequent 2-itemsets present in F2. Step 19 scans the POC tree and an iterative process from Step 20 through Step 29 discovers frequent (>2) itemsets. The map() function ends here by returning F representing frequent itemsets for the given portion of transactions. This way multiple map() functions operate based on the worker nodes involved in the map phase of the MapReduce paradigm. The intermediate results (F of each mapper) are given to reducers to combine results and return final frequent itemsets obtained from the given T.

Algorithm: Fast and Scalable Frequent Itemset Mining algorithm

Pseudocode: A Novel Nodesets Based Fast and Scalable Frequent Itemset Mining

Input: A transactional database T, minimum support threshold ***th***

Output: Discovered frequent itemsets R

Map() Function
1. Construct POC tree
2. $F1 \leftarrow$ FindFrequent1Itemsets ()
3. For each node n in the POC tree
4. item $\leftarrow$ FindItem(n)
5. For each ancestor of n and n'
6. $item_{n'} \leftarrow$ FindItem(n')
7. IF item and $item_{n'}$ belong to $F2$ Then
8. $support\_of\_item\_item_{n'}$
$support\_of\_item\_item_{n'} + n.acc$

```
9.            F2 ← F2U{item, item_n}
10.         End If
11.       End For
12.       For each itemset Q in F2
13.         IF Q.sup<th x |T| Then
14.           F2 ← F2-{Q}
15.         Else
16.           Q.nodeset ← null
17.         End If
18.       End For
19.       Scan POC tree
20.       For each node n in the POC tree
21.         item ← FindItem(n)
22.         For each ancestor of n' and n''
23.           item_n'' ← FindItem(n'')
24.           IF item and item_n' belong to F2 Then
25.             item_item_n': nodeset ← item_item_n'' nodeset U item_n''.N_info
26.           End If
27.         End For
28.       F=F U F1
29.       Return F
Reduce() Function
30.       For each F in intermediate Frequent Itemsets
31.         R+=F
32.       End For
33. Return R
```

4. EXPERIMENTAL RESULTS AND ANALYSIS

We experimented with the Cloudera environment to evaluate the FSFIM algorithm and compare that with state-of-the-art methods such as Mahout PFP [32], Mlib PFP [33], and Big FIM [34]. For experiments, the real dataset known as Delicious explored in [35] is used. This dataset is a collection of web tags. Each record represents the tag assigned by a user to a URL and it consists of 4 attributes: date, user id (anonymized), tagged URL, and tag value. The transactional representation of the delicious dataset includes one transaction for each record, where each transaction is a set of four pairs (attribute, value), i.e., one pair for each attribute. The dataset stores more than 3 years of web tags. It is very sparse because of the huge number of different URLs and tags. A prototype application is built using Java language on top of the MapReduce paradigm to implement the proposed algorithm. The dataset has 41,949,956 transactions and 57,372,977 items. Observations are made in terms of execution time against different minimum support values and several attributes.

Table 2. Shows execution time of different algorithms against various mins up values

Minsup (%)	Execution Time (Seconds)			
	Mahout PFP	Mlib PFP	Big FIM	FSFIM
0	90000	10000	7500	5000
0.2	9500	8000	6200	3000
0.4	500	2500	500	300
0.6	400	900	400	250
0.8	400	600	400	150
1	400	400	400	100

As presented in Table 2, the execution time of the algorithms is provided for different minsup (%) values.

As presented in Fig. 5, it is evident that the minimum support values used for experiments are presented on the horizontal axis while the execution time is shown on the vertical axis. The results revealed that the proposed method FSFIM outperforms the state-of-the-art methods.

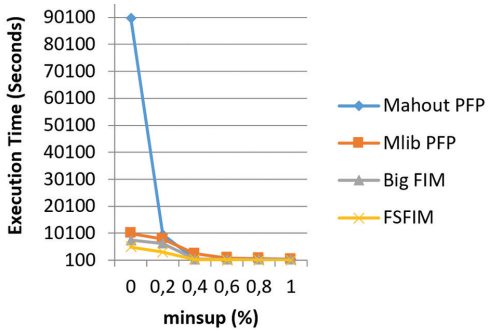

Fig. 5. Performance comparison in terms of execution time against different mins up (%)

As presented in Table 3, the execution time of the algorithms is provided for different transaction length values.

Table 3. Shows the execution time of different algorithms against various transaction lengths

Transaction Length	Execution Time (Seconds)			
	Mlib PFP	Mahout PFP	Big FIM	FSFIM
10	100	120	120	70
20	100	125	125	115
30	120	150	150	100
40	450	300	300	200
50	1250	425	425	354
60	3200	550	550	530
70	5100	600	600	585
80	7500	950	650	620
90	8600	1400	900	835
100	9800	1800	1200	1100

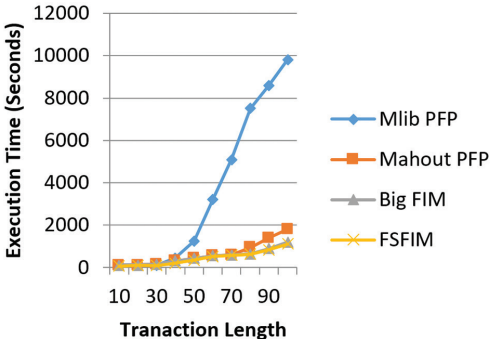

Fig. 6. Performance comparison in terms of execution time against transaction length

As presented in Fig. 6, it is evident that the transaction length values used for experiments are presented on the horizontal axis while the execution time is shown on the vertical axis. The results revealed that the proposed method FSFIM outperforms the state-of-the-art methods.

The execution time of the algorithms is provided for a different number of attributes is presented in Table 4.

Table 4. Shows the execution time of different algorithms against several attributes

Number of Attributes	Execution Time (Seconds)		
	Mahout PFP	Mlib PFP	FSFIM
0	100	400	50
10	1400	4200	600
20	1800	12100	800
30	2500	13900	1200
40	7000	18760	4500
50	18100	19850	15200

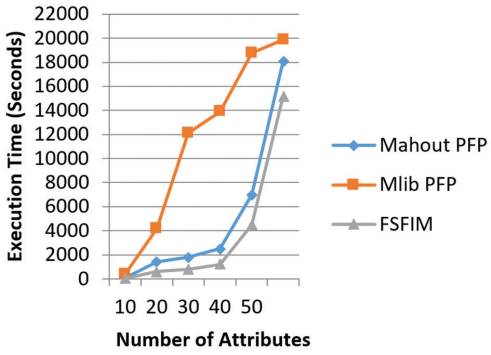

Fig. 7. Performance comparison in terms of execution time against the number of attributes

The FSMFI is better than the existing Mahout PFP, and Mlib PFP. As presented in Fig. 7, it is evident that the number of attributes used for experiments is presented in the horizontal axis while the execution time is shown in the vertical axis. The results revealed that the proposed method FSFIM outperforms the state-of-the-art methods. Experimental results revealed that FSFIM outperforms existing algorithms such as Mahout PFP, Mlib PFP, and Big FIM. FSFIM is more scalable and found to be an ideal candidate for real-time applications that mine frequent itemsets from big data.

5. CONCLUSION AND FUTURE WORK

In this paper, we propose a novel algorithm known as A Novel Nodesets Based Fast and Scalable Frequent Itemset Mining (FSFIM) to extract frequent itemsets from big data. Pre-Order Coding (POC) tree is used to represent data and improve speed in processing. Nodeset is the underlying data structure that is efficient in discovering frequent itemsets. When compared with its predecessors such as Node-lists and N-lists, the Nodeset saves half of the memory as it needs only either a pre-order or post-order code. Cloudera's Distribution of Hadoop (CDH), a MapReduce framework, is used for empirical study. A prototype application is built to evaluate the performance of the FSFIM. Experimental results revealed that FSFIM outperforms existing algorithms such as Mahout PFP, Mlib PFP, and Big FIM. FSFIM is more scalable and found to be an ideal candidate for real-time applications that mine frequent itemsets from big data. Findings in the empirical study include faster execution and scalability. In the future, we would like to extend the FSFIM to support incremental mining of frequent itemsets to avoid scanning the entire database and reinventing wheel everything when the algorithm is executed.

6. REFERENCES

[1] H. Qiu, R. Gu, C. Yuan, Y. Huang, "YAFIM: A Parallel Frequent Itemset Mining Algorithm with Spark", Proceedings of the IEEE International Parallel & Distributed Processing Symposium Workshops, Phoenix, AZ, USA, 19-23 May 2014, pp. 1-8.

[2] M. Yasir et al. "TRICE: Mining Frequent Itemsets by Iterative TRimmed Transaction LattICE in Sparse Big Data", IEEE Access, Vol. 7, 2019, pp. 181688-181705.

[3] S. Gole, B. Tidke, "Frequent itemset mining for Big Data in social media using ClustBigFIM algorithm", Proceedings of the International Conference on Pervasive Computing, Pune, India, 8-10 January 2015, pp. 1-6.

[4] Y. Djenouri, D. Djenouri, J. C.-W. Lin, A. Belhadi, "Frequent Itemset Mining in Big Data with Effective Single Scan Algorithms", IEEE Access, Vol. 6, 2018, pp. 1-15.

[5] D. Apiletti, E. Baralis, T. Cerquitelli, P. Garza, F. Pulvirenti, L. Venturini, "Frequent Itemsets Mining for Big Data: A Comparative Analysis", Big Data Research, Vol. 9, 2017, pp. 67-83.

[6] C. Fernandez-Basso, A. J. Francisco-Agra, M. J. Martin-Bautista, M. D. Ruiz, "Finding tendencies in streaming data using Big Data frequent itemset mining", Knowledge-Based Systems, 2018, pp. 1-21.

[7] K. K. Sethi, D. Ramesh, "HFIM: a Spark-based hybrid frequent itemset mining algorithm for big data processing", The Journal of Supercomputing, Vol. 73, No. 8, 2017, pp. 3652-3668.

[8] K.-W. Chon, S.-H. Hwang, M.-S. Kim, "GMiner: A fast GPU-based frequent itemset mining method for large-scale data", Information Sciences, Vol. 439-440, 2018, pp. 19-38.

[9] Y.-H. Liang, S.-Y. Wu, "Sequence-Growth: A Scalable and Effective Frequent Itemset Mining Algorithm for Big Data Based on MapReduce Framework", Proceedings of the IEEE International Congress on Big Data, New York, NY, USA, 2015, pp. 1-8.

[10] R. Joy, K. K. Sherly, "Parallel frequent itemset mining with spark RDD framework for disease prediction", Proceedings of the International Conference on Circuit, Power and Computing Technologies, Nagercoil, India, 18-19 March 2016, pp. 1-5.

[11] Y. Djenouri, D. Djenouri, A. Belhadi, A. Cano, "Exploiting GPU and cluster parallelism in single scan frequent itemset mining", Information Sciences, Vol. 496, 2018, pp. 1-15.

[12] K.-W. Chon, M.-S. Kim, "BIGMiner: a fast and scalable distributed frequent pattern miner for big data", Cluster Computing, Vol. 21, 2018, pp. 1-14.

[13] S. Raj et al. "EAFIM: efficient apriori-based frequent itemset mining algorithm on Spark for big transactional data", Knowledge and Information Systems, Vol. 62, 2020, pp. 3565-3583.

[14] S. Moens, E. Aksehirli, B. Goethals, "Frequent Itemset Mining for Big Data", Proceedings of the IEEE International Conference on Big Data, Silicon Valley, CA, USA, 6-9 October 2013, pp. 1-8.

[15] Y. Xun, J. Zhang, X. Qin, X. Zhao, "FiDoop-DP: Data Partitioning in Frequent Itemset Mining on Hadoop Clusters", IEEE Transactions on Parallel and Distributed Systems, Vol. 28, No. 1, 2017, pp. 101-114.

[16] A. Asbern, P. Asha, "Performance evaluation of association mining in Hadoop single node cluster with Big Data", Proceedings of the International Conference on Circuits, Power and Computing Technologies, Nagercoil, India, 19-20 March 2015, pp. 1-5.

[17] S. Kumar, K. K. Mohbey, "A review on big data parallel and distributed approaches of pattern mining", Journal of King Saud University - Computer and Information Sciences, Vol. 34, No. 5, 2019, pp. 1-24.

[18] M. Zitouni, R. Akbarinia, S. B. Yahia, F. Masseglia, "A Prime Number Based Approach for Closed Frequent Itemset Mining in Big Data", Proceedings of Database and Expert Systems Applications, Valencia, Spain, pp. 509-516.

[19] C. K. Leung, H. Zhang, J. Souza, W. Lee, "Scalable Vertical Mining for Big Data Analytics of Frequent Itemsets", Proceedings of Database and Expert Systems Applications, 2018, pp. 3-17.

[20] P. Galetsi, K. Katsaliaki, S. Kumar, "Big data analytics in the health sector: Theoretical framework, techniques, and prospects", International Journal of Information Management, Vol. 50, 2020, pp. 206-216.

[21] C. Fernandez-Basso, M. D. Ruiz, M. J. Martin-Bautista, "A fuzzy mining approach for energy efficiency in a Big Data framework", IEEE Transactions on Fuzzy Systems, Vol. 28, 2020, pp. 1-12.

[22] F. Fumarola, D. Malerba, "A parallel algorithm for approximate frequent itemset mining using MapReduce", Proceedings of the International Conference on High-Performance Computing & Simulation, Bologna, Italy, 21-25 July 2014, pp. 1-8.

[23] Y. Djenouri et al. "A Novel Parallel Framework for Metaheuristic-based Frequent Itemset Mining", Proceedings of the IEEE Congress on Evolutionary Computation, Wellington, New Zealand, 10-13 June 2019, pp. 1-7.

[24] A. Aggarwal, D. Toshniwal, "Frequent Pattern Mining on Time and Location Aware Air Quality Data", IEEE Access, Vol. 7, 2019, pp. 98921-98933.

[25] J. M. Luna, F. Padillo, M. Pechenizkiy, S. Ventura, "Apriori Versions Based on MapReduce for Mining Frequent Patterns on Big Data", IEEE Transactions on Cybernetics, Vol. 48, No. 10, 2017, pp. 2851-2865.

[26] S. Nalousi, Y. Farhang, A. B. Sangar, "Weighted Frequent Itemset Mining Using Weighted Subtrees: WST-WFIM", IEEE Canadian Journal of Electrical and Computer Engineering, Vol. 44, No. 2, 2021, pp. 206-215.

[27] F. Li, Z. Zhang, B. Cheng, P. Zhang, "Probabilistic Fuzzy Frequent Item Sets Mining (PPFIM)," Proceedings of the 7th International Conference on Cloud Computing and Big Data Analytics, Chengdu, China, 2022, pp. 127-132.

[28] A. Sharma, R. K. Singh, "An Efficient Approach to Find Frequent Item Sets in Large Database", Proceedings of the 1st Odisha International Conference on Electrical Power Engineering, Communication and Computing Technology, Bhubaneswar, India, 2021, pp. 1-6.

[29] B. Sivaiah, R. R. Rao, "A Survey on Fast and Scalable Incremental Frequent Item Set Methods for Big Data", Proceedings of the International Conference on Intelligent Controller and Computing for Smart Power, Hyderabad, India, 2022, pp. 1-5.

[30] R. Agarwal, A. Gautam, A. K. Saksena, A. Rai, S. V. Karatangi, "Method for Mining Frequent Item Sets Considering Average Utility", Proceedings of the International Conference on Emerging Smart Computing and Informatics, Pune, India, 2021, pp. 275-278.

[31] W. Tian, F. Li, Y. Liu, Z. Wang, T. Zhang, "Depth-First Uncertain Frequent Itemsets Mining based on Ensembled Conditional Item-Wise Supports", Proceedings of the International Conference on Intelligent Supercomputing and BioPharma, Zhuhai, China, 2023, pp.121-128.

[32] S. Bagui, K. Devulapalli, J. Coffey, "A heuristic approach for load balancing the FP-growth algorithm on MapReduce", Array, Vol. 7, 2020, p. 100035.

[33] X. Meng et al. "Mllib: Machine learning in Apache spark", Journal of Machine Learning Research, Vol. 17, No. 1, 2016, pp. 1235-1241.

[34] Y. Rochd., I. Hafidi, B. Ouartassi, "A Review of Scalable Algorithms for Frequent Itemset Mining for Big Data Using Hadoop and Spark", Real-Time Intelligent Systems, Lecture Notes in Real-Time Intelligent Systems, Springer, 2017, pp. 90-99.

[35] R. Wetzker, C. Zimmermann, C. Bauckhage, "Analyzing social bookmarking systems: adel.icio.us cookbook", Mining Social Data Workshop Proceedings, 2008, pp. 26-30.