A Big Data Framework for *Mining Sensor Data* Using *Hadoop*

Engy A. EL-SHAFEIY*, Ali I. EL-DESOUKY
Computers and Systems Department, Faculty of Engineering,
Mansoura University, Egypt
*(*Corresponding author)* e-mail: engy.elshafeiy@gmail.com.

Abstract: The data gathered from IOTs is considered of high business value. The IOTs devices sense the natural conditions using sensor network comprised of sensor nodes. Mining of big sensor data for useful knowledge extraction is a very challenging task. Frequent itemsets is one of the most effective mining techniques that find important itemsets from big sensor data. In this paper, a MapReduce Frequent Nodesets-based Boundary POC tree (MR-FNBP) framework is proposed for mining Frequent Nodesets for big sensor data. The MapReduce framework is used to implement MR-FNBP to enhance its performance in highly distributed environments. Additionally, the proposed Boundary (FNBP) creates a Boundary as an early stage to exclude the infrequent itemsets, and this may reduce the overall memory and time usage. Moreover, a number of experiments were performed to evaluate the performance of MR-FNBP framework. The results show high scalability rate and a less time consuming process for MR-FNBP framework over different recent systems.

Keywords: Big data, Internet of Things, MapReduce, Wireless Sensor Networks, Mining Frequent Nodesets.

1. Introduction

Nowadays, the Internet of Things (IoTs) is growing quickly as a subset of big data. Billions of recent physical devices, such as smart devices [4] and Wireless sensing Sensor Networks (WSNs) area unit [15] are expected to be connected in the near future. WSNs are available in various applications and services, mostly organizations, including public and private, especially in the medical field and health care Therefore, the data gathered and collected from the WSNs are considered to be a great source of big data. With the recent advancements in communication technology, more and more data are generated and collected, therefore, the big data will grow exponentially and this will increase the challenges of extracting and retrieving the complexity of the valuable hidden data. There are more than three billion users of smart objects including smart phones, smart homes, as well as business and entertainment applications [16]. These smart devices allow Machine to Machine (M2M) electronic communication with or without an intermediary-user. This has led to what is known as the "Internet of Things (IoTs) "[8]. The huge amount of data generation has been useful in various fields such as commercial, industrial, scientific, social and medical [11], as shown in Figure 1.

Big data is a collection of very huge datasets with a great diversity of types so that it becomes difficult to process by using state-of-the-art data processing approaches or traditional data processing platforms such as Processing Big Trajectory Data [19]. In 2012, Gartner retrieved and gave a more detailed definition as: Big data are high-volume, high-velocity, and/or high-variety information assets that require new forms of processing to enable enhanced decision making, insight discovery and process optimization. The main characteristic of Big data included the 3Vs characteristics (Veracity, Viability, and Value) and then was elaborated to include the following characteristics known as the 6Vs:

Volume: Describes the huge data size.
Velocity: Describes the data communication, processing speeds per time unit.
Variety: Describes the different data types (structured, semi-structured, and unstructured).
Value: Describes the valuable data knowledge
Veracity: Describes the data quality, such as data cleaning, filtering.

Viability: Describes the prediction possibilities.

More generally, a dataset can be called big data if it is formidable to perform capture, analysis and visualization on it using current technology. With diversified data provisions, such as sensor networks, telescopes, scientific experiments, and high throughput instruments, the datasets increase at exponential rate [18]. Other Big data applications lie in many scientific disciplines such as astronomy, atmospheric science, medicine, genomics, biologic, biogeochemistry and other complex and interdisciplinary scientific researches. Web-based

https://doi.org/10.24846/v26i3y201712

applications encounter big data frequently, such as recent hot spots social computing (including social network analysis, online communities, recommender systems, reputation systems, and prediction markets), Internet text and documents, Internet search indexing. Alternatively, there are uncountable number of sensors around us that generate less amount of sensor data which in a need to be utilized, for instance, intelligent transportation systems (ITS) [27] are based on the analysis of large volumes of complex sensor data. Large-scale e-commerce applications [10] are particularly data-intensive as they involve a large number of customers and transactions. Data mining is one of the greatest tasks needed for the management of big sensing data stream [2].

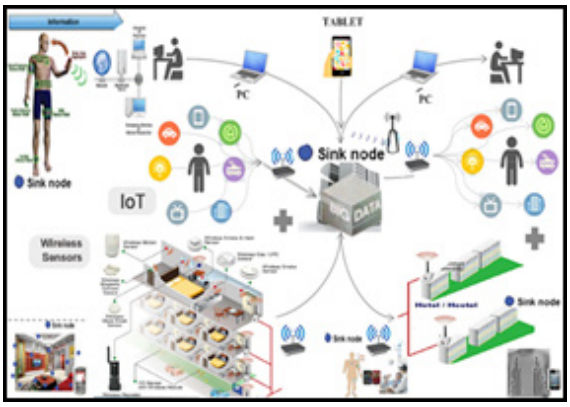

Figure 1. The Big data generate from Internet of Things in various fields

WSNs are successfully deployed in detection applications and diverse monitoring [16]. In these applications, WSNs generate Big data in the form of streams. Such data stream from WSNs can be mined to extract knowledge in real time about the sensed environment (e.g., mining certain behaviours [8, 10]) and the network itself (e.g., predicting faulty nodes [1]), and this presents new challenges for data mining techniques. Data mining techniques, which are well established in the traditional database systems [27], have recently received a great deal of attention as promising tools to extract interesting knowledge from sensor data streams.

Discovering associated rules from WSNs can be highly useful in applications that require a fine-grain monitoring of physical environments (e.g., buildings, transportation networks, and battlefield) which may face critical situations like fire, toxic gas leaks and explosion [18]. Behavioural patterns can also be used to predict the source of future events. Knowing the source of a future event may lead to detect faulty nodes, if any, in the network.

Data Mining has introduced techniques and tools to extract interesting Frequent itemsets from Big data. Frequent itemsets mining is an important subfield of data mining, which consists of discovering interesting and useful patterns in transaction databases. The traditional task of Frequent itemsets mining is to discover groups of itemsets that appear frequently together in transactions made by customers. Although itemsets mining was designed for market basket analysis, it can be viewed more generally as the task of discovering groups of attribute values frequently co-occurring in databases. Because of its numerous applications in domains such as bioinformatics, text mining, product recommendation, e-learning, and web click stream analysis, Frequent itemsets mining has become a popular research area. FIM is the most imperative techniques in Data Mining and Frequent itemsets Mining can be classified as follows: Apriori a based horizontal formatting method is the most established algorithm for finding Frequent itemsets from dataset, however; it needs to scan the dataset many times to create many candidate itemsets [1]. Elastic methods are based on vertical formatting scanning. [24]. FP-tree method that uses compressed data format represented as a tree data structure and does not require candidate itemsets generation [9]. FP-tree has a better performance than Apriori algorithm, but when mining large amounts of datasets its execution time increases, Wang, 2010 proposed Node-list (ppv)[5], Deng et al 2012 proposed N-list (PrePost) [6] and 2014 FIN [7]. The high efficiency of PPV and PrePost which based on Node-lists and N-lists respectively is accomplished by these two properties. FIN Fast Mining itemsets using Nodesets [7] algorithm has been introduced to encode each node either with a pre-order or post-order. By avoiding scanning database repeatedly and mining without candidate generation. FIN achieves the high efficiency when compared to others. However, execution time is increased when facing large datasets. Emerging platforms like IoTs will generate big amounts of sensor data. Therefore, this type of assumption will no longer be valid [18]. To process Big data in transactional databases, researchers have focused on large scale parallel and distributed frequent itemsets mining techniques [8, 20, 25, 26] to improve scalability and to resolve the sequential bottlenecks and response time. However, these

techniques are not suitable to handle big sensor datasets. Parallel and distributed big sensing data stream techniques assume that data are transmitted and partitioned to the computing nodes in advance. This approach is impractical in distributed systems for mining of large sensor data. To handle big sensing data stream, some researchers have proposed the use of MapReduce [26] to mine the search space in a distributed manner. It assumes a data-centric method of distributed computing with the principle of 'moving computation to data'. It uses a distributed file system that is particularly optimized to improve the I/O performance while handling big sensing data stream. Hadoop is an open source implementation of the MapReduce framework. MapReduce needs to share and pass the support of individual candidate nodesets rather using the whole sensor dataset. Therefore, communication cost is low compared to the traditional distributed environments. These several parallel mining algorithms [20, 25] which have been proposed consist of parallel construction of Frequent itemsets trees and parallel mining of the tree structure in a distributed memory environment. Parallel mining of Frequent itemsets using MapReduce [23, 26]. MapReduce is a scalable programming model in which the programmer writes two functions; a map and reduced functions. Each of these functions defines a mapping from one set of key-value pairs to another [12]. The map function takes an input as the key/value and produces a set of intermediates key/values. It groups all the intermediate values associated with the same key and passes are grouped to be used in the reduced function. The reduced function takes an intermediate key and a set of values for the key and merges all values together to form smaller set of values [13]. One widely used implementation of MapReduce is Apache Hadoop [28] which is a collection of related services that compose an infrastructure for distributed computing. Hadoop is known for MapReduce and its Hadoop Distributed File System HDFS [17] it provides complementary services such as; Core, MapReduce, HBASE and HDFS. MapReduce is a distributed data processing model and execution environment that runs on large clusters of commodity machines. MapReduce provides an abstraction that hides many system-level details from the developer. Therefore, the developer can focus on what computations need to be performed as opposed to how those computations are actually carried out or how to get the data to the processes that depends on them [22]. MapReduce provides a means for distributing computation without burdening the programmer with the details of distributed computing [3]. Because of the benefits of MapReduce model it is used as a parallel programming model. In [7] the PrePost algorithm is implemented with MapReduce Framework as it is the fastest algorithm among others [14, 21]. We will briefly introduce MapReduce Framework (MR-FNBP) at the sink node based on Hadoop platform for distributed big sensor data stream mining technique over MapReduce. This paper is structured as follows: Section 2 describes our MapReduce Framework (MR-FNBP) which is based on the FNBP algorithm which based on the adaptive Nodesets, including POC-Tree, Boundary as an early stage to exclude the infrequent itemsets. Section 3 shows the effect of implementing FNBP algorithm and MR-FNBP framework, the results are compared with the results of the recent work using real and synthetic datasets. Section 4 introduces our conclusions and future works.

2. The Proposed MapReduce Frequent Nodesets-based Boundary POC tree (MR-FNBP) framework

The MapReduce model for a parallelized association rule for sensor data. This model intends to process sensor data at the sink node that stores the target data collected from sensor nodes, unlike other studies that either process data in Hadoop or edge node (sensor node). MR-FNBP proposed approaches will use the IoTs devices through a wired or wireless network. Our proposed approach will work locally on the sink node rather the cloud environment. The frequent mining aims to discover the knowledge of a large dataset, in our case, we will send the knowledge discovered by the association rule mining at the sink node instead of sending all the data streams of the sensor, hence reducing the amount of data transmission for a big data storage. The Modified MapReduce Framework (MR- FNBP) is proposed to solve the problem of processing large scale datasets by mining the Frequent Nodesets using the proposed FNBP algorithm. As shown in Figure 2 the Framework consists of four main layers:

1. Flume/Hadoop Data senor
2. MR- FNBP
3. Hadoop Cluster
4. Hadoop/MapReduce.

2.1 Flume/Hadoop Data senor

Apache Flume tool is used to capture stream data for transferring data to HDFS. Flume defines it as a distributed, reliable, and affordable way to efficiently aggregate, and move large amounts of data streams to HDFS. It consists of a simple, flexible, data-based architecture. Typical data sources for Flume are data produced by sensors and other IoTs devices. This data can be uploaded to HDFS by Flume for further analysis or archived only. Sensor nodes are distributed in this data centre. Task Tracker and data node services run on each Flume node/sensor in the centre. The Task Tracker accepts tasks to provide task trackers with the required sensor data, using HDFS in MapReduce layer.

2.2 Hadoop cluster

Data analysis and management server based on Hadoop cluster. Hadoop cluster for analysis of multiple sensor data sources to create reduced collections for higher level analytics.

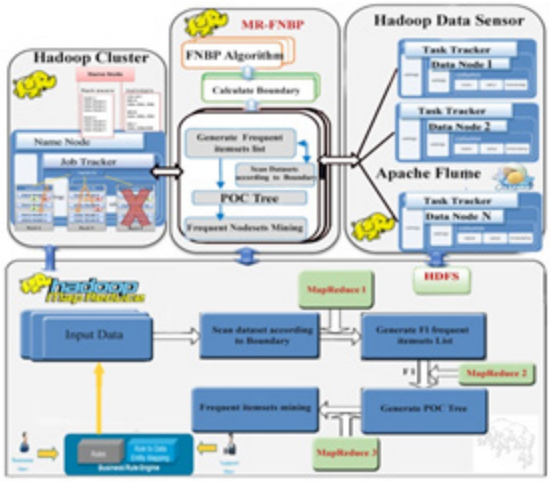

Figure 2. MapReduce Frequent Nodesets-based Boundary POC tree (MR-FNBP) framework

The master node is located in this layer. The job Tracker services running on the master nodes, is used to coordinate job requests sent to and from the Task Trackers in client's node using MapReduce.

MR- FNBP layer

This layer implementation is distributed across the nodes using Hadoop/MapReduce layer, and it consists of three main modules: Frequent Nodesets based on Boundary POC tree (FNBP). The proposed FNBP algorithm is a modified version of the FIN algorithm [7] and the Boundary POC tree. Also, this algorithm handles each node in the itemsets either with a pre-order or post-order code to overcome the lack of FIN algorithm in memory consumption and time used by creating a Boundary at an early stage before creating the POC tree used by FIN to exclude the infrequent items and this may lead to a reduction in overall memory and time usage. FNBP scans the dataset two times:

1- The first scan is used to create a list of initial Frequent single-itemsets F1 based on the Boundary Bo to detect the infrequent items and delete them before being used by POC tree, if an item appears in less than Bo then it is considered infrequent and excluded, this will decrease the memory consumption and run time. Finally, the POC tree is constructed without the infrequent items.

2- Based on the created POC tree, the second scan will be performed to create a list of Frequent double-itemsets F2, with updated Frequent items. Then F2 compared with each transaction in dataset. The Frequent itemsets count will be updated. Based on this list, the POC tree is updated. Then, POC tree is scanned. The FNBP algorithm shown in Figure2 is based on heuristic 1.

Heuristic 1: let $DB = \{S1, S2, ..., Sn\}$ is the transaction database, M(s) is the Minimum support threshold, $|T|$ is the total number of transactions, *Bo* is the Boundary for transactions, is the itemsets Support, *Tc* is the current Transaction, a_i is the current itemsets where $ai \subseteq T$. The main goal of the Mining of Frequent itemsets is to find the set of all Frequent itemsets. The Boundary is calculated, as follows $Bo = T - M(s) + 1$ (1)

The FNBP algorithm: starts with calculating (the Boundary for transactions) which is calculated as follows:

Bo = number of total Transactions – min support +1.

The dataset is scanned according to Minimum support threshold $M(s)$ defined by the user (depending on datasets size). The transactions of each dataset are scanned whether the transaction number *Tc* is less than or equal to the Boundary value (Bo). Any item greater than Bo will be added to the list Item (F1). Otherwise, it is excluded and increment the count of the list Item (F1) that is

used to construct the POC-tree. Scan DB again to form and use the POC-tree for mining the Frequent itemsets in each record and rearrange them in the same order of F1.

Output: F, (the set of all Frequent itemsets).Procedure:
F←Ø;
Calculate Boundary $(Bo)=T-M(s)+1$;
Scan Dataset once according to *Bo*;
For each Ti ⊆ DB do
If $TC(Current\_Transaction\_No) \le Bo$,Then
Split the transaction into items;
Get the item;
ai← currently item;
If ai new item **Then**
F1(the set of all Frequent 1-itemset) ← ai;
End if
End for
Filter Infrequent Items;
Sort F1 (the set of all Frequent 1-itemset) descending order as L1;
Create of POC-Tree root, Tr, label ← Null;
For each Ti ⊆ DB do
Sort the Frequent items according to order F1;
Sorted *Frequent item List*→ [*p*|*P*]; //p the first item and P the reminder List
INSERT tree ([p|P], Tr);
If *N.item - name = p.item-name Then*// Tr has child N
increase N's count by 1;
else
Create a new node N;
increase N's count by 1;
END if
End for
Scan POC-Tree to generate pre-order of each node by pre-order traversal;
F2←Ø; //store Frequent two-itemsets
Scan the POC-Tree by the pre-order traversal for each node N **do**
F2← F2{two-items}; //Insert all Frequent two-itemsets→ F2
F2← F2-{p}; //Delete all infrequent two-Itemsets from F2
P.Nodests← Ø; //Initial the Nodesets of all Frequent two-itemsets
Scan the POC-Tree by the pre-order traversal for each node N do
Generate the Nodesets of all Frequent two-itemsets
F← F F1; // to Generate all Frequent k-itemsets(k≥3)
For each Frequent itemset, is it, in F2 do //for each Frequent **two-Itemsets** such as is it
Create the root of a tree, Rst, and label it by List;
Call Pattern_*Tree* (*Rst*,{*i* | *i* ϵ *F*1,*i* > *is*},Ø);
End for
Return F

Figure 3. The proposed FNBP Algorithm

For this purpose, we use FIN algorithm, which is a breadth-first search algorithm used to identify the association rules that highlight the general trends of the database. Then apply the FIN algorithm on the pre-processed dataset. The FIN algorithm aims to identify the most frequent items, and prunes the infrequent items from the list, to generate the rules.

Assuming the List of items in each record is [*p*|*P*],*p*] where, p the first item in the list, and P is the reset of items. The function called ([*p*|*P*], *Ti*]) Insertis used to insert the first item and reminder of the list and its transaction in the tree, the tree formed respectively in pre-order traversal set pre order of each node to establish Nodesets-list of Frequent itemsets [7] as shown in Figure 3.

Example of working FNBP algorithm

Suppose we have a dataset of 10 transactions from data sensor, as shown in Table 1:

Table 1. Sensor data stream flow

Dataset Tid	Itemset	Sorted Frequent items
1	S_1, S_2, S_3, S_4, S_5	S_3,S_1,S_5,S_2
2	S_1, S_3, S_5, S_8	S_3, S_1,S_5,S_8
3	S_2, S_3, S_4, S_5, S_7,S_1	S_3,S_1,S_5,S_2
4	S_1, S_3, S_5, S_8	S_3,S_1,S_5,S_8
5	S_2, S_3, S_5,S_8	S_3,S_5,S_2,S_8
6	S_2, S_6, S_7, S_{10}, S_{11} ,S_3	S_3,S_2
7	S_1, S_8, S_{10}, S_{11}	S_1,S_8
8	S_1, S_2, S_6, S_8, S_{10}	S_1,S_2,S_8
9	S_2, S_6, S_8, S_{12}, S_5,S_1	S_1,S_5,S_2,S_8
10	S_1, S_8, S_9, S_{12}, S_{13}	S_1,S_8

Suppose that our Minimum support threshold *M*(*S*) is 0.6, Minimum support to total transaction database is 6 and *Bo* is 5. In the first scan, suppose that *Tc* = {*S*1, *S*2, *S*3, *S*4, *S*5, *S*7, *S*8}items are used for memory allocation and count updating operation. Based on FNBP algorithm, the itemsets are beyond the pre-defined Bo, as a result they will not be used in memory allocation as they aren't Frequent after transaction T5. But according to Minimum support Is count, the sorted one Frequent items= {*S*3, *S*1, *S*5, *S*2, *S*8}.

- Suppose that our Minimum support threshold M(S) is 0.4, Minimum support to total transaction database is 4 and *Bo* is 7. T1 to T7 used for memory allocation= {*S*1, *S*2, *S*3, *S*4, *S*5, *S*6, *S*7, *S*8, *S*10, *S*11} and new *Tc* = {*S*9, *S*12, *S*13} will be considered as infrequent items, and items {*S*4, *S*6, *S*7, *S*10, *S*11} will be removed due to Minimum support Is count and only {*S*1, *S*2, *S*3, *S*4, *S*8} will be available as Frequent Items out of total unique 13 items, so the sorted one-Frequent Items= {*S*3, *S*1, *S*5, *S*2, *S*8}. The example shows that using Boundary in FNBP algorithms reduces the execution time by reducing the storage space of candidate items and their counting operations. The Threshold and Boundary Generation: Threshold and Boundary Generation are implemented as: *M*(*S*) Let be the predefined minimum support and |DB| be the number of transactions in DB. An itemset

P is Frequent if its support is no less than $M(S)$ x |DB| Given a transaction database DB and a threshold $M(S)$, the old task of Mining Frequent itemsets is to find the set of all itemsets whose supports are not less than $M(S)$ x |DB|. But in this task the calculating Bo (the Boundary for transactions), which calculated as Bo = |DB| - $M(S)$ + 1. The dataset is scanned according to Bo.

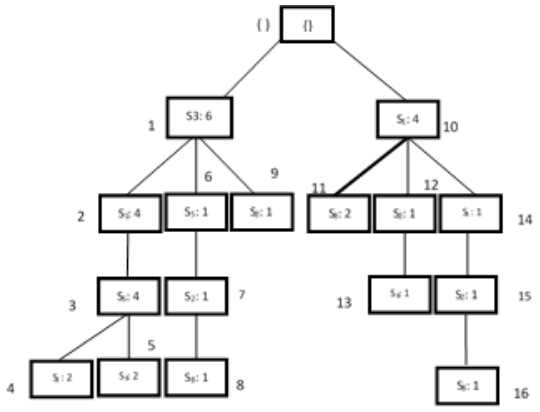

Figure 4. The POC tree after running Algorithm 1 on database shows in table

2.3 Hadoop/MapReduce

Hadoop/MapReduce service runs on each node. It consists of Hadoop Distributed File System (HDFS), used for data storage, mapper and reducer functions used by MR-FNBP. This layer contains three main phases as follows; MapReduce1, MapReduce2 and MapReduce3. These phases are responsible for calculating the Frequent itemsets based on Nodesets-list using the proposed FNBP algorithm. F1-list is created in MapReduce1 while MapReduce2 creates POC tree and finally all Frequent itemsets are created through MapReduce3. These can be shown from layer 4 in Figure 2. The core of this layer is the FNBP algorithm and the three parallel MapReduces.

The main steps of the proposed FNBP algorithm. Each K-Frequent itemset FK corresponds to Frequent itemsets-list is organized in descending order according to the pre-order code. The results of Frequent itemsets-list construct the POC-Tree's main purpose which is to generate Nodesets-list. Finally, the POC tree is mining to produce all the Mining Frequent itemsets based on Nodesets-list. Then, we can delete the POC Tree to reduce memory overhead. The main steps are:

- Scan transaction database named DB according to the given Boundary (itemsets selected), output the F1-list in descending order according to the number of its support.

- Scan (itemsets selected), select the Frequent items in each record and arrange them in the order of F1, assuming the list of Items in each record is [p|P], p which is the first item in the list and P is the rest of the items to generate the POC tree.

- The formed tree created in preorder on each node is grouped to establish Nodesets-list of the Frequent itemsets.

- Mining Frequent itemsets based on the Nodesets-list using the FIN algorithm.

2.3.1 Phase 1: The MapReduce 1 functions

In the first MapReduce function two steps are conducted (i) the database is divided into (m) itemsets (ii) the itemsets processing is performed using MapReduce computations.

The Pseudocode for Map 1 function

Output: F_1-list {set of Frequent one Itemsets list in descending order from sensor data}
1. Procedure: Mapper. (Key, value = Ti)
2. **If** Ti ≤ Bo **Then**
3. **For** each item a_i in T_i **do**
4. Output (<key= a_i , value=1>)
5. **end for**
6. **end if**
7. **end**

The Pseudocode for Reduce1 function

1. **Procedure**: Reducer (key= a_i, value=S (a_i))
2. count = 0;
3. **For** each 1 *in* S (*ai*) **do**
4. count + = 1;
5. **end for**
6. **If** (count >= M(S))**Then**
7. Output (<key= a_i ; value=count >); //output the
8. **call function Sort** (F_1);

Output the F_1-list

Figure 5. Pseudo code of Map and Reduce functions in the first phase

Each map function takes one split as input. The output of this phase is Frequent itemsets and their occurrence for each split as a list of intermediate key/values. In this phase, each nodes independently perform map function, reduce function combined statistical results and dropped infrequent itemsets according to Boundary calculate. Figure 5 shows the map and reduce function for phase 1. This phase blocks the database level whereas the process uses the default file block policy of Hadoop, and then the data block is called shard which is allocated on each worker node.

Count the number of items in each shard set in map stage, reduce function merges output of map

stage and generates Frequent itemsets Mining (F1-list) according to the Frequent Boundary Bo, generate descends F1-list. In this step the Frequent itemsets are generated for each block resulted from the previous step and the MapReduce model outputs the itemsets along their occurrences in the block using one map and one reduce function.

2.3.2 Phase 2: The MapReduce 2 functions

This phase generates Frequent itemsets and their occurrence in all shards: based on the generated itemset from the previous phase. The itemsets and its occurrence in the whole shards are generated using one map and one reduce functions. In Map 2 shard functions. Each map filters shard based on the input F1-list, for each transaction of sort Frequent items based on the sequence of F1-list and outputs the same F1-list.

Input: The shards and F_1-list.
Output: (POC-tree)
1. **Procedure**: Mapper (key, T_i)
2. **For** each T_i **do**
3. Select the Frequent items in T_i sort out the according to the order of F_1-list
4. Produce a path [*pIP*] as the value to output <key, [*pIP*]>
5. **end for**
6. **end**

The Pseudocode for reduce 2 Function

[POC- Tree Constriction]
Procedure: Reducer (key, [*pIP*])
1- F←Ø;
2- *Create POC TreeNode root,Tr*;
3- *POC TreeNode label ← Null*;
4- **For** each [*pIP*] **do**
5- Call *INSERT tree*([*p*|*P*],*Tr*);
6- **If** *N.item - name = p.item - name* ***Then***// *Tr has child N*
7- Increase N's count by 1;
7. **Else**
8. Create a new node N with its count, Initialized to 1;
9. **If** P is nonempty **then**
10. Call INSERT tree([*p*|*P*],*Tr*) recursively
11. Scan POC-Tree to generate the pre-order of each node by the pre-order traversal;
12. **end if**
13. **end if**
14. **end**

Figure 6. Pseudo code of Map and Reduce functions in the second phase

Then, reduce function constructs the compressed POC-Tree. Only Preorder traversal is made to determine and generate Nodeset-lists of one Frequent itemsets. Algorithm 3 is shown in Figure 6 depicts the pseudo code of the second map and reduce functions 2.

2.3.3 Phase 3: The MapReduce 3 functions

The third phase: firstly group the generated F-list resulted from phase 1 and phase 2 in a single Frequent itemsets nodeset and 2' Frequent itemsets nodeset according to the Pseudo code 6, Secondly, these Nodesets are mined using the last POC tree as shown in Figure 7.

The Pseudocode of generation Nodesets- List of 2' Frequent Itemsets
15. POC-tree and group of the F_1-List, the set of Frequent one- itemset List
16. **Output**: the set of the Nodesets -lists of Frequent two-itemsets.
17. *F2*←Ø; //store Frequent two-itemsets
18. **Scan** the POC-Tree by the pre-order traversal;
19. **for** each ancestor of two-node N,Nado
20. **If** *p.support*< *M*(*S*) ×|*DB*| **Then**
21. *F2*← *F2*-{*p*};
22. **Else**
23. P.Nodeset←Ø;
24. **End if**
25. **End for**
26. **Scan** the POC-Tree by the pre-order traversal;
27. **for** *each ancestor of two-node Nd,Nda* **do**
28. **if** two-*item registered in Nd,Nda* ϵ *F2* **then**
29. *two - item. Nodeset*← *two - item. Nodeset* ∪ *Nda.N_info*;
30. **end if**
31. **end for**
32. *F*←*F* ∪ *F2*
33. **For** *i* = 0 *to F2.size()* **do**
34. **groups** [*i* % *grouped_size*] *add* (*F2*[*i*]) ;
35. **end for**
36. **end**

Figure 7. Pseudo code of generating F2-list of two Frequent itemsets and grouped in Nodeset

The mining_kItemSetFreq (F, M(S), Nd, itemsets_ childnodesgenerate, parent Nd_frequent)
1. **For** (j=0 to F) **do**
2. *Nd.equivalent_items* ← Ø; //Nd is the current node
3. *Nd.childnodes*← Ø;
4. Next_ itemsets_childnodesgenerate ← Ø; // itemsets_childnodesgenerate is the available items to generate child nodes of Nd
5. For *each i ϵ itemsets_childnodesgenerate do*
6. *X* ←*Nd.itemset*
7. *Y* {*i*} ∪ (*X* _ *X*[1])
8. P {*i*} ∪ X;
9. P.*Nodeset*← *X.Nodeset* ∩ *Y.Nodeset*
10. If *P.support = X.supportthen*
11. Nd.equivalent_items=Nd.equivalent_items {i};
12. Else if P.support ≥ |DB| × M(S),
13. *Create node Ndi*
14. *Ndi.label* ←*i*;
15. *Ndi.itemset*← *P*;
16. *Nd.childnodes*← *Nd.childnodes* ∪ {Ndi};
17. Next_ itemsets_childnodesgenerate ← Next_ itemsets_childnodesgenerateU {i};
18. **End if**
19. **End for**
20. If Nd.equivalent_items ≠ Ø then
21. SS the set of all subsets of Nd.equivalent_items;
22. PSet {A | A = Nd.label , À ϵ SS};
23. If parentNd_Frequent =Ø, then
24. FIT_Nd← PSet;
25. Else
26. FIT_Nd {Ƥ | Ƥ = P1 [P2, (P1 ≠ Ø ^ P1 ϵ PSet) and (P2 ≠ Ø ^ P2 ϵparentNd_Frequent };
27. End if
28. F ← F FIT_Nd;
29. End if
30. End for

Figure 8. Pseudo code of Map functions in the third phase

3. Performance Evaluation

A number of experiments were conducted to evaluate the performance of both FNBP algorithm and (MR-FNBP) Framework.

3.1 Evaluation of FNBP algorithm

To evaluate the performance of the FNBP, a number of experiments are conducted on three different datasets: BMS-Webview-2, Connect, and T25I10D100 K which are often used in most previous researches of Frequent itemsets Mining. The BMS-Webview-2 and Connect datasets are downloaded from SPMF repository. The BMS-Webview-2 dataset contains click-stream data from a web store used in KDD-Cup 2000 while the connect dataset is derived from game steps. The T25I10D100 K dataset is a synthetic dataset and generated by the IBM generator.

Table 2 shows the characteristics of these datasets. It shows the average transaction length (denoted by Avg. Length), the number of items (denoted by Items) and the number of transactions (denoted by Trans) in each dataset.

Table 2. The characteristics of these datasets under study

Database	Avg. length	Items	Transaction
BMS-Webview-2	160	3.340	77.512
Connect	43	130	67,557
T25I10D100 K	25	990	100000

In FNBP implementation, the FIN, PrePost and FP-growth algorithms are used as the baseline algorithms. FIN and PrePost have proven to be the best algorithms among all node-based methods [6, 7]. FP-growth is the best algorithm among FP-tree-based methods [9]. All these algorithms are implemented in Java. Using 8G memory PC with an Intel Core i5 processor. And windows server 2012 operating system standard x64 Edition.

3.1.1 Experiment one

These Experiment testes the running time of FNBP algorithm, when compared with three recent Frequent itemSets Mining algorithms. This is made for the three datasets shown in Table 2. The results obtained are shown in Figure 9. Figure 9 shows the relation between running time (measured in s) and Support (%) and Memory consumption for the datasets.

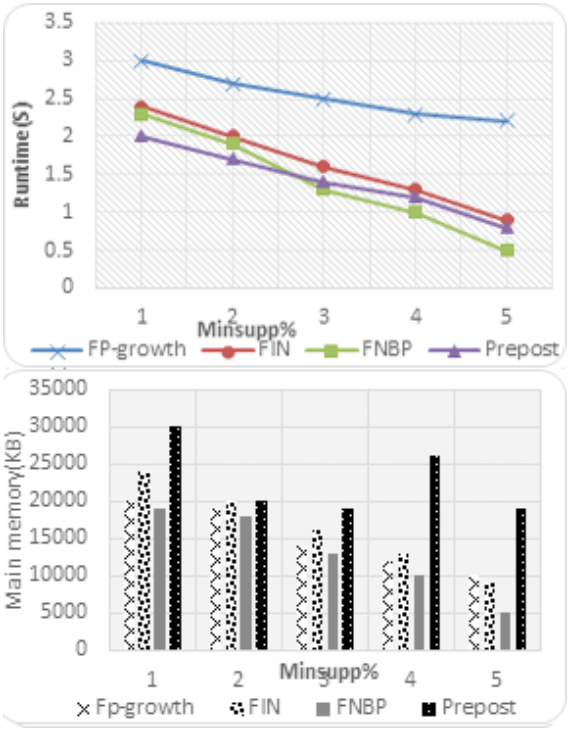

Figure 9(a). Running time and consuming Memory on T25I10D100 K

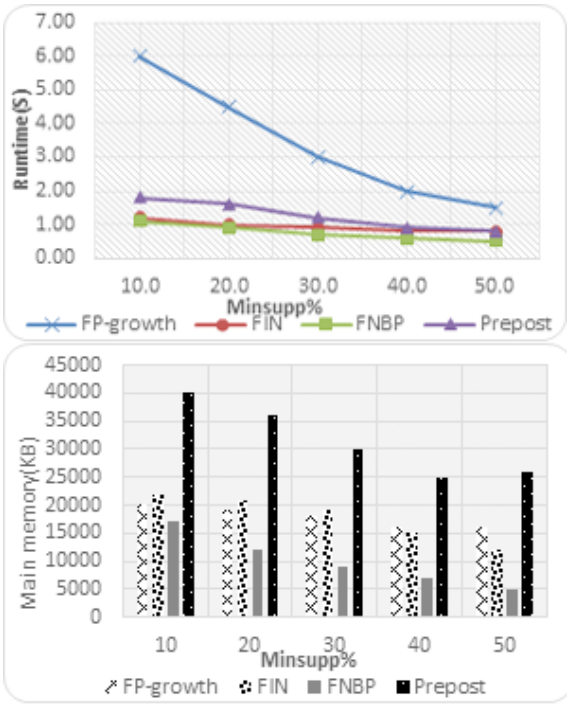

Figure 9(b). Running time and consuming Memory on BMS-Webview-2

Figure 9(a) shows that the running time and memory consumption of FNBP are much lower than FP-growth and FIN with the same support more than 3%. FNBP running time and memory consumption get slightly higher and great

improvement in memory consumption than FIN. Figure 9(b) and 9(c) also show that the running times of FNBP algorithm is lower than the other algorithms for all Minimum support values and Memory consumption.

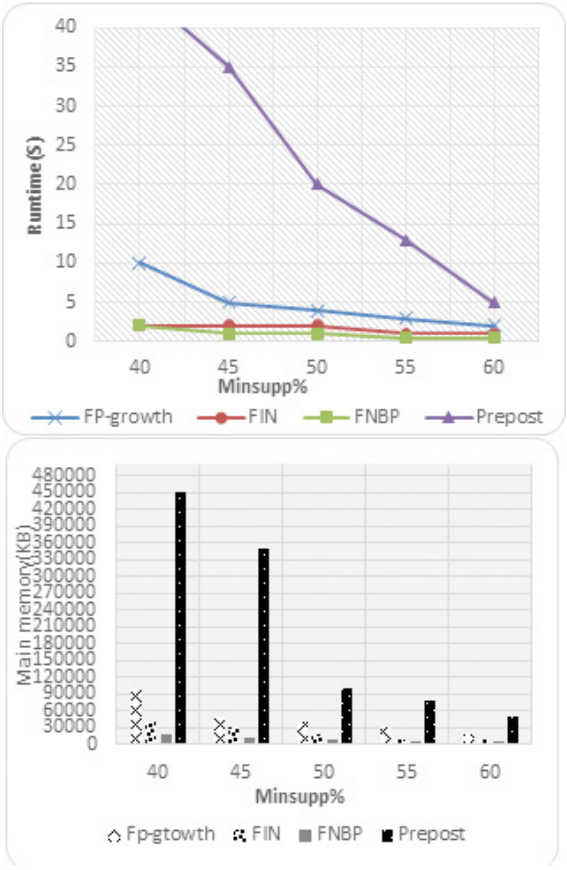

Figure 9(c). Running time and consuming Memory on Connect

Figure 9: comparison study

3.2 Evaluation of MR-FNBP algorithm

3.2.1 Experiment two

A number of experiments have been performed in order to validate and evaluate our framework. These experiments are used to test the run time and measure the performance of MR-FNBP Framework compared with MrPrePost (3MR) [14] and MrPrePost (5MR) Framework [21]. MrPrePost has proven to be the best parallel algorithm based on Hadoop platform and MrPrePost algorithm can adapt to mining large data's association rules. But, there have been improvements in MrPrePost (3MR) algorithm with using3 MapReduce and then it was developed with the use of 5MapReduce till finding N-List from Frequent Itemset. The datasets Connect T10I4D100K and T10I18D1000K which are used to test the performance of the (MR-FNBP) Framework. The first two are available in http://fimi.ua.ac.be/dataJ or http://archive.ics.uci.edu/ml/datasets.html. The dataset T10I18D1000K are generated with Spawner Data Generator tools. Table 3 shows the parameters used to generate T10I18D1000K datasets.

Table 3. The Parameters used to generate T10I18D1000K dataset

Average maximal potentially Frequent Itemsets size	4,10,000 distinct items
Average transaction size	10
Number of transactions in the dataset generated	1000 K
Number of transactions in the different items used in the dataset	100 K

Three desktop computers are used with Ubuntu 14.4 and core i5 processor and memory size 8GB RAM to test the validity of the (MR-FNBP) Framework in highly distributed environments. The results obtained are shown in Figure 10. The Figure show that the running time of MR-FNBP Framework is much better than all others frameworks in all Minimum support in large or small dataset, but in small datasets we must adjust the minimum support to improve time and memory. The method setnumMapTasks (int num) method in the configuration mapping is used to test the validity of the MR-FNBP Framework compared with the others framework, the number of maps in each node is increased. The results obtained are shown in Figure 10(d).

The method setnumMapTasks(int,num) method in the configuration mapping is used to test the validity of the MR-FNBP Framework compared with the others framework, the number of maps in each node is increased.

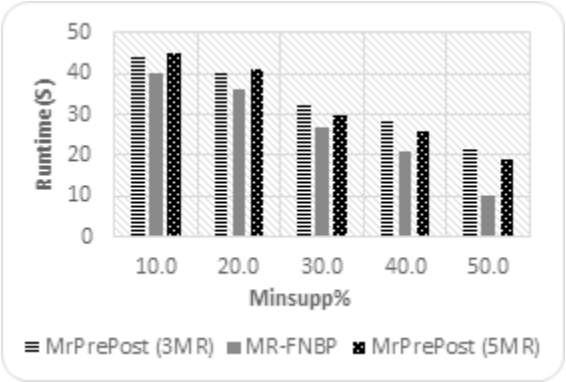

Figure 10(a). Running time of MR-FNBP on T1014D1000K

The results obtained are shown in Figure 10(d). The results showed that increasing the number of map will decrease the running time used for MR-FNBP, this is due to MR-FBNP Framework pre-defined the Frequent itemsets in an early stage.

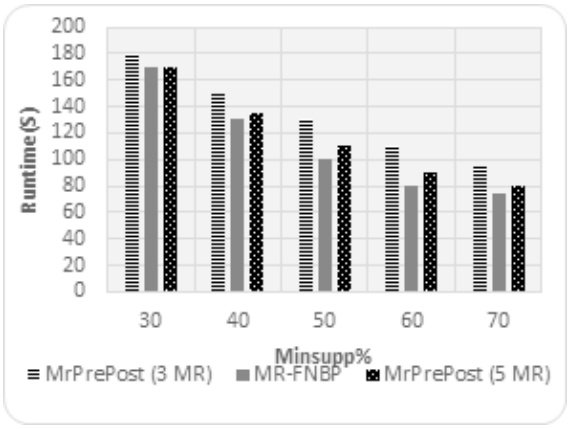

Figure 10(b). Running time of MR-FNBP onT-10118D1000K

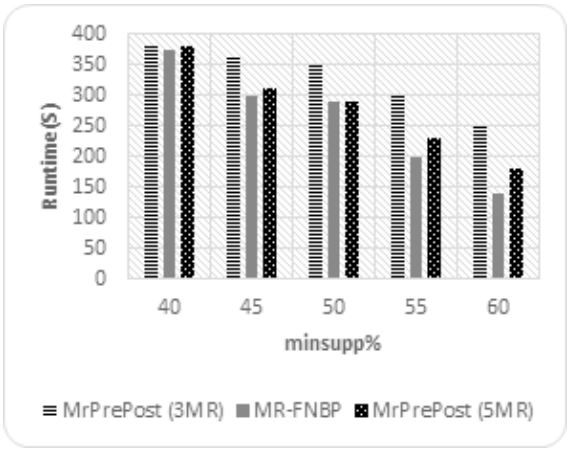

Figure 10(c). The Running time of MR-FNBP on Connect

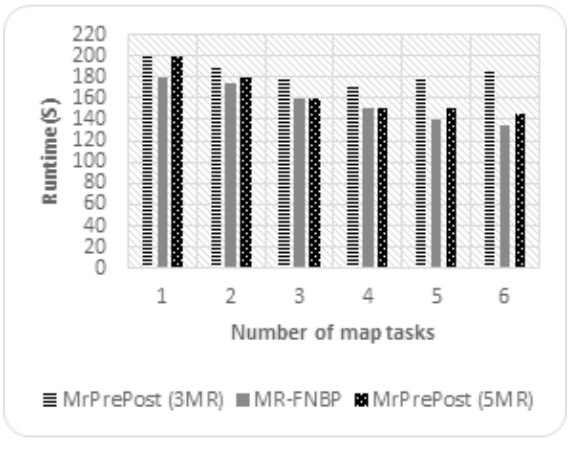

Figure 10(d). The runtime of MR-FNBP with different of map tasks

Figure 10: compression study

This results in decreasing the number of blocks in the input file of the MapReduce functions. From the Figure 10(d) we cannot notice a slight reduction in the running time after (3) maps due to the limited capacity of the single node.

3.2.2 Experiment three

Our experiment has been using widely used measures including sizeup and speeding up. For the details of the measures refer to [21] our experiments are performed in a Hadoop 2.7.2 cluster of 12 nodes, one of which worked as master and the others worked as slaves. The slave nodes are the same hardware settings: core i7 processor, memory size 16G RAM, the (MR-FNBP) Framework is implemented with Java. We generate two datasets called (Dataset 1, Dataset 2). The synthetic datasets Dataset 1, Dataset 2 are generated to be used in the experiments and the size of these datasets 500 MB, 1GB respectively.

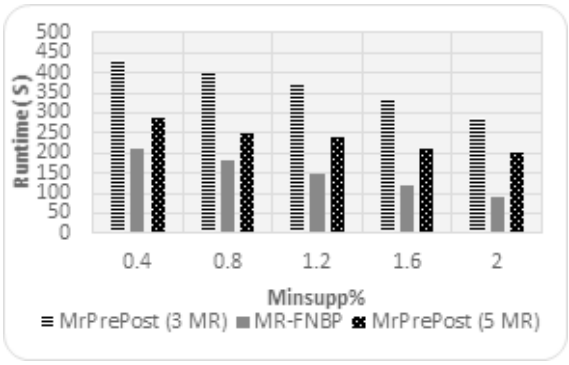

Figure 11(a). Dataset 1 (500MB)

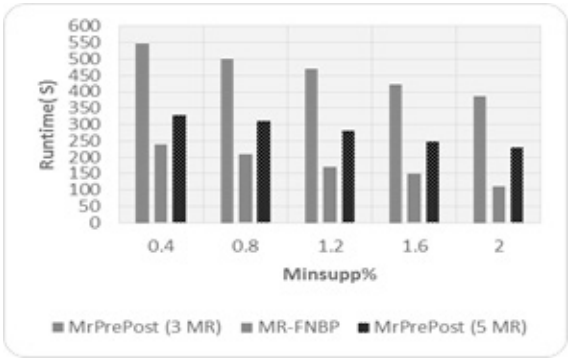

Figure 11(b). Dataset 2 (1 GB).

Figure 11: runtime comparison by varying the Min Support %

Experiments for different datasets and different threshold (0.4 to 2) were done. The results are shown in Figures 11, 12. Figure 11 shows very much decreasing in running time of MR-FNBP comparing to others. Figure 12 shows a running time with enlargement of the size of the datasets. Figure 13 shows Speed-up S(P) comparing to others on different cluster size for the datasets (Dataset 1, Dataset 2). The (MR-FNBP) Framework can deal with large datasets better.

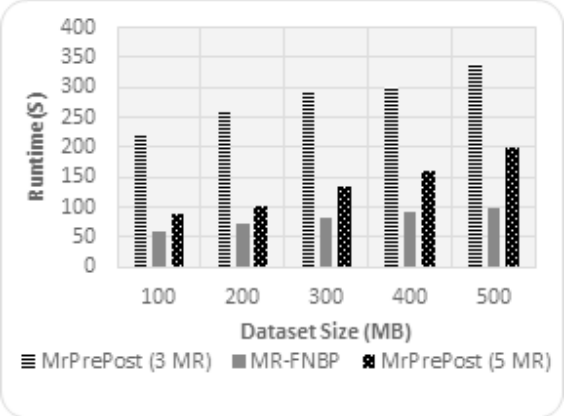

Figure 12(a). Dataset 1 (500MB, Minsupp=2.2%)

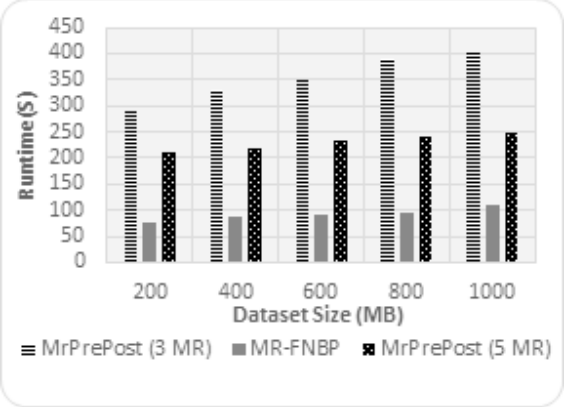

Figure 12(b). Dataset 2 (1 GB, Minsupp=2.8%)

Figure 12: runtime comparison by varying the dataset size

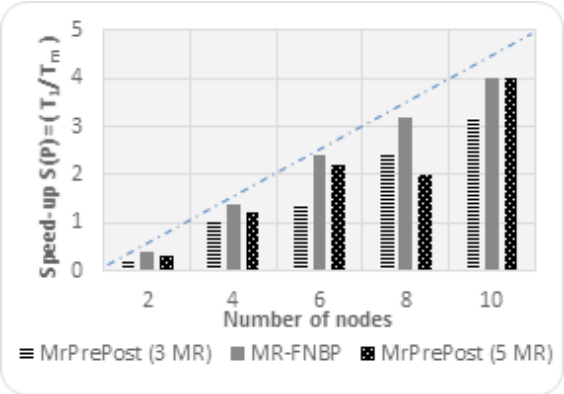

Figure 13(a). Dataset 1 (500MB, Minsupp=2.2%)

Figure 13 is Speed-up for different cluster size. *Speed-up S*(*P*) is defined in literature as follow: $S(P) = T(1)/T(m)$(2).

Where $T(1)$ running time of an algorithm on one single node, and $T(m)$: running time of an algorithm on multiple (m) node.

The dashed line is the ideal workflow to linear speed-up and it intersects the mean values of all boxplots.

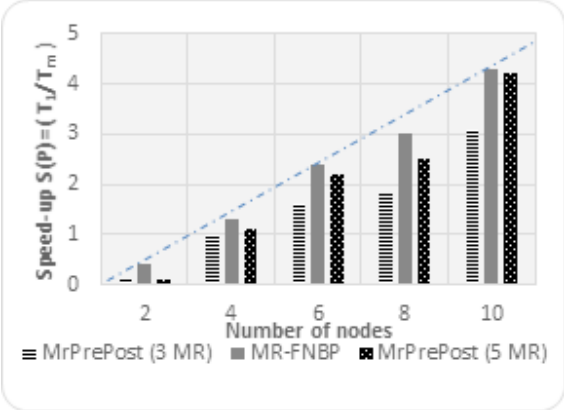

Figure 13(b). Dataset 2 (1 GB, Minsupp=2.2%).

Figure 13: speed-up S(P) for different cluster size

4. Conclusions and Future Work

This paper proposed a modified FNBP Nodesets algorithm with an early Boundary stage based on the Boundary POC tree to ignore the infrequent items to resolve memory problems. The paper also presents a modified MapReduce Framework capable of implementing on several computing nodes and achieves highly parallel computing. Several experiments have been done; the results obtained showed that either FNBP algorithm or MR-FNBP Framework using MapReduce are viable and efficient in Mining Frequent itemsets for complex data or huge amount of data.

In the era of Big Data, cost efficient high performance computing proved to be the only viable option for most scientific disciplines. Frequent itemsets Mining is one of the most representative fields in this area, as the data explosion has exceeded current hardware capabilities. The rate of producing new data is expected to increase significantly faster as well as the cost in hardware computational capabilities. Data-aware optimization can be a powerful weapon in our arsenal when it is utilized from data mining to develop sciences and to provide new insights.

REFERENCES

1. Agrawal, R., Imieliński, T., & Swami, A. (1993, June), Mining association rules between sets of items in large databases. In *Acm sigmod record* (Vol. 22, No. 2, pp. 207-216). ACM.

2. Aggarwal, C. C. (Ed.) (2013), *Managing and mining sensor data*. Springer Science & Business Media.

3. Dean, J., & Ghemawat, S. (2008), MapReduce: simplified data processing on large clusters. *Communications of the ACM*, *51*(1), 107-113.

4. Dhope, T., Simunic, D., & Djurek, M. (2010), Application of DOA estimation algorithms in smart antenna systems. *Studies in informatics and Control*, *19*(4), 445-452.

5. Deng, Z., & Wang, Z. (2010), A new fast vertical method for mining frequent patterns. *International Journal of Computational Intelligence Systems*, *3*(6), 733-744.

6. Deng, Z., Wang, Z., & Jiang, J. (2012), A new algorithm for fast mining frequent itemsets using N-lists. *Science China Information Sciences*, *55*(9), 2008-2030.

7. Deng, Z. H., & Lv, S. L. (2014), Fast mining frequent itemsets using Nodesets. *Expert Systems with Applications*, *41*(10), 4505-4512.

8. Elmangoush, A., Alhazmi, A., Magedanz, T., Schuch, W., Estevez, C., Ehijo, A., & Mukudu, N. (2015, October), Towards Unified Smart City Communication Platforms. In *Proceedings of the Workshop on Research in Information Systems and Technologies, Chillán, Chile* (Vol. 16).

9. Han, J., Pei, J., & Yin, Y. (2000, May), Mining frequent patterns without candidate generation. In *ACM sigmod record* (Vol. 29, No. 2, pp. 1-12). ACM.

10. Ionita, I. (2013). SAM-An Automated System Based on Data Mining for Credit Scoring. *STUDIES IN INFORMATICS AND CONTROL*, *22*(4), 291-298.

11. Ionita, I., & Ionita, L. (2016), Applying Data Mining Techniques in Healthcare. *STUDIES IN INFORMATICS AND CONTROL*, *25*(3), 385-394.

12. 12. Lin, K. C., Liao, I. E., Chang, T. P., & Lin, S. F. (2014), A frequent itemset mining algorithm based on the Principle of Inclusion–Exclusion and transaction mapping. Information Sciences, 276, 278-289.

13. Lin, K. C., Liao, I. E., Chang, T. P., & Lin, S. F. (2014), A frequent itemset mining algorithm based on the Principle of Inclusion–Exclusion and transaction mapping. *Information Sciences*, *276*, 278-289.

14. Leung, C. K. S., MacKinnon, R. K., & Jiang, F. (2014, June), Reducing the search space for big data mining for interesting patterns from uncertain data. In *big data (BigData congress), 2014 IEEE international congress on* (pp. 315-322). IEEE.

15. Liao, J., Zhao, Y., & Long, S. (2014, May), MRPrePost—A parallel algorithm adapted for mining big data. In *Electronics, Computer and Applications, 2014 IEEE Workshop on* (pp. 564-568). IEEE.

16. Merezeanu, D., Vasilescu, G., & Dobrescu, R. (2016), Context-aware Control Platform for Sensor Network Integration in IoT and Cloud. *Studies in Informatics and Control*, *25*(4), 489-498.

17. Rathore, M. M., Ahmad, A., Paul, A., & Rho, S. (2016), Urban planning and building smart cities based on the internet of things using big data analytics. *Computer Networks*, *101*, 63-80.

18. Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010, May), The hadoop distributed file system. In *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on* (pp. 1-10). IEEE.

19. Sridhar, P., & Dharmaji, N. (2013), A comparative study on how big data is scaling business intelligence and analytics. *Int. J. Enhanced Res. Sci. Technol. Eng*, *2*(8), 87-96.

20. Stojanovic, N., & Stojanovic, D. (2015), A Hybrid MPI+ OpenMP Application for Processing Big Trajectory Data. *Studies in Informatics and Control, ISSN12201766*, *24*(2).

21. Tan, K. L., & Sun, Z. H. (2006), An algorithm for mining FP-trees in parallel. *Computer Engineering and Applications*, *13*, 155-157.

22. Thakare, S., Rathi, S., & Sedamkar, R. R. (2016), An Improved PrePost Algorithm for Frequent Pattern Mining with Hadoop on Cloud. *Procedia Computer Science*, *79*, 207-214.

23. Woon, Y-K., W-K. Ng, and E-P. Lim. "A support-ordered trie for fast frequent itemset discovery." IEEE Transactions on Knowledge and Data Engineering 16.7 (2004): 875-879.

24. Woon, Y. K., Ng, W. K., & Lim, E. P. (2004), A support-ordered trie for fast frequent itemset discovery. *IEEE Transactions on Knowledge and Data Engineering*, *16*(7), 875-879.

25. Zaki, M. J., Parthasarathy, S., Ogihara, M., & Li, W. (1997, August), New Algorithms for Fast Discovery of Association Rules. In *KDD* (Vol. 97, pp. 283-286).

26. Zaïane, O. R., El-Hajj, M., & Lu, P. (2001), Fast parallel association rule mining without candidacy generation. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on* (pp. 665-668). IEEE.

27. Zhou, L., Zhong, Z., Chang, J., Li, J., Huang, J. Z., & Feng, S. (2010, November), Balanced parallel fp-growth with mapreduce. In *Information Computing and Telecommunications (YC-ICT), 2010 IEEE Youth Conference on* (pp. 243-246). IEEE.

28. Zhang, J., Wang, F. Y., Wang, K., Lin, W. H., Xu, X., & Chen, C. (2011), Data-driven intelligent transportation systems: A survey. *IEEE Transactions on Intelligent Transportation Systems*, *12*(4), 1624-1639.

29. Hadoop, A. (2016), Welcome to apache hadoop. *2011-06-20]. http.-//hadoop, apache, org*.