INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2022, VOL. 68, NO. 2, PP. 177–186
Manuscript received June 4, 2021; revised April, 2022. DOI: 10.24425/ijet.2022.139865

High Frequency Rule Synthesis in a Large Scale Multiple Database with MapReduce

Sudhanshu Shekhar Bisoyi, Pragnyaban Mishra, and Saroja Nanda Mishra

***Abstract*—Increasing development in information and communication technology leads to the generation of large amount of data from various sources. These collected data from multiple sources grows exponentially and may not be structurally uniform. In general, these are heterogeneous and distributed in multiple databases. Because of large volume, high velocity and variety of data mining knowledge in this environment becomes a big data challenge. Distributed Association Rule Mining(DARM) in these circumstances becomes a tedious task for an effective global Decision Support System(DSS). The DARM algorithms generate a large number of association rules and frequent itemset in the big data environment. In this situation synthesizing high-frequency rules from the big database becomes more challenging. Many algorithms for synthesizing association rule have been proposed in multiple database mining environments. These are facing enormous challenges in terms of high availability, scalability, efficiency, high cost for the storage and processing of large intermediate results and multiple redundant rules. In this paper, we have proposed a model to collect data from multiple sources into a big data storage framework based on HDFS. Secondly, a weighted multi-partitioned method for synthesizing high-frequency rules using MapReduce programming paradigm has been proposed. Experiments have been conducted in a parallel and distributed environment by using commodity hardware. We ensure the efficiency, scalability, high availability and cost-effectiveness of our proposed method.**

***Keywords*—Multiple Database; Frequent Itemset; Association Rule; Rule Synthesis; MapReduce; HDFS**

I. Introduction

DARM in a large database [1], [2] is rapidly becoming the popular strategy for rule-based DSS in large-scale multiple databases. Many larger organization maintains their data in multiple sites. Development of communication and information technology, adoption of digital marketing and social media promotes the organizational data to grow exponentially and distributed over multiple databases. Each local database site of a global organization or company may contain different categories of data in all or some local database sites. For example, a retail sector need a database for product details, a finance sector may need multiple databases for the transaction like regular accounts, loan accounts and credit card, to build an efficient transactional model, a pharmaceutical sector need a database for medicines etc. So integrating multiple databases perfectly and extracting meaningful knowledge from this large-scale heterogeneous and distributed databases are becoming a challenging task. Each organization maintains a large amount of data in their databases regularly for building effective mining models. The exponential growth of data into terabyte or petabyte scale characterizes the volume and velocity while mining at multiple databases. Because of this, multi-database mining can be termed as Big Database Mining(BDM). So extracting meaningful, hidden knowledge and understandable information from the huge amount of data stored in large multiple databases is the main concern of BDM. This big data problem cannot be solved with the help of traditional methods and frameworks because of the volume, velocity and veracity [3], [4]. In a rule-based multi-database mining process, a very large set of frequent patterns and association rules from each database site are generated. All these rules are transferred to a central site for designing a global effective model for decision making [5]. Because of the very large dataset, it is also difficult to amass all the frequent itemset along with rule in a central storage and synthesize these rules for the DSS. So synthesizing global frequent rule at multiple big database environments with the help of traditional technique, methods and systems is a challenging task. The challenges are mainly concern with scalability, efficiency, load balancing, storage and processing cost etc. So during the synthesis process of high frequent rules from a big database, the application needs to be data-centric in nature. In this situation, frequent patterns from each database need to be stored in a partition based distributed big data warehouse System. Each partition or splits are replicated for high availability and scalability. A single mining task needs to be initiated over the specified partition or split of the data node and collect all the mining information in a parallel and distributed fashion [6]. To increase efficiency in distributed storage and parallel processing on the cost of commodity hardware, Hadoop becomes the de-facto open source framework [7], [8]. It consists of HDFS as the distributed storage and MapReduce as a parallel processing framework. It can handle the big data challenges in the cluster of commodity hardware [9], [10]. In this paper, we present a model to collect data from multiple databases to HDFS for the distributed storage. High-frequency rule selection and synthesis algorithms are proposed by using MapReduce as the parallel and distributed processing framework of Hadoop.

S. S. Bisoyi is with Department of Computer Science and Information Technology, Siksha 'O' Anusandhan Deemed to be University (SOA), Institute of Technical Education and Research (ITER), Bhubaneswar, Odisha, India (e-mail: sudhanshu.bisoyi@gmail.com).

P. Mishra is with Dept. of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation , Vaddeswaram, Guntur, AP, India (e-mail: pragnyaban@gmail.com).

S. N. Mishra is with Dept. of CSE&A, IGIT, Sarang, Dhenkanal, Odisha, India (e-mail:sarose.mishra@gmail.com).

Rest of the paper is organized as follows: In section 2, basic terminologies and related works are described. In section 3, the parallel and distributed rule synthesize methods are presented. The experimental setup and result analysis are described in section 4 and final conclusion in section 5.

II. Related Work

A. *Rule Mining in Distributed Database*

Knowledge mining in a distributed environment refers to the use of distributed computing framework. A single mining process can be initiated on more than one computing node in parallel. Over the year many data mining methods have been proposed to identify the scalability and performance of association rule mining in multiple distributed databases [11], [12]. Many large and small organization having multiple branches are geographically distributed with own database. Generation of global high frequent rules concern to all sites of the organization is a challenging task. In the recent years, many methods have been proposed for multi-database mining. H. Liu. et.al. [13] have proposed a method for identifying the database relevant to a data mining task. A set of semantically related database are chosen with respect to a relevance measure. This method is used to obtain multiple relevant databases where each of them contains certain relevant pattern or attributes. It can be used to overcome a situation where multiple databases are required to be joined to a single large database and carry out the mining operation. This classification by considering relevance factor is completely dependent on the databases and application. Wu. X. et.al. [14] discuss the multi-database classification based upon the similarity measure which helps in reducing the searching cost and improves efficiency in the performance of multi-database mining system. It helps in the identification of pattern association by considering an application independent method to classify the database.

Over the time each site of an organization generates a large number of rules associated with the local database. Selecting globally high frequent rules and synthesizing these rules become necessary for effective DSS. S. Zhang. et.al. [15] have proposed a multi-database model(MDM) by describing various issues related to mono-database and multi-database mining. They classify the patterns associated with multiple branches of an organization broadly into 4 types. These are local, high voting, exceptional, and suggested patterns. C. Zhang et.al [16] proposed a new algorithm for identifying the global exceptional pattern in multiple databases. The Candidate Exceptional Pattern(CEP) have been generated based upon the average vote counting. The identification of CPE from the multi-database is considered as the post-processing operation but the focus has not been given to organizational business database. Wu and Zhang [17] have proposed a weighted model for selection and synthesis of high frequent rules from different data sources. The rules generated from each site need to be collected in a central rule base system. In their approach, they have described how multiple database participate equally in the DSS. The participation of database site is based upon their weight. The database with uniform size is considered, if not then need to be resized by preprocessing. They also describe the use of local and global support to synthesize the high frequent rules. Ramkumar et.al [18] extended this work where a new approach was proposed based upon database site weight for selection and synthesis of high frequent rules. Their approach is based upon the varying size of database site and may be practically applicable to the organizational database. The weight of each database site was computed by considering the transaction population. They have proposed a new method for computing the global support and confidence for the rules selected form different database sites. It has also been proved that this support and confidence are nearly similar while performing mono-database mining by integrating all partitions. The author has proposed a procedure for synthesizing global negative association rule in multiple database. The local pattern analysis strategy was adapted for mining negative association rule from frequent items. The global negative rules were synthesized with the help of forwarding negation relations. They have also proposed a weighting model based on transaction population for synthesizing global negative association rule where uninterested negative rules are pruned by effective vote rate [19]. Adhikari et.al [20], have proposed an algorithm for effective database grouping in multi-database mining. They have introduced an approach for non-local pattern analysis(NLPA) by combining database grouping algorithm and pipelined feedback technique(PFT) while mining multiple databases. They have judged the improvement in global pattern mining by sacrificing local pattern properties in multiple databases.

The different proposed methods for synthesizing rules in the multiple databases leads to the difficulty in data storage and processing as described earlier. Each database site has a large number of frequent patterns. They generate a large number of possible association rules individually. Synthesizing high frequent rules by the traditional algorithms leads to a huge search space and reduce the efficiency. So to synthesize high frequent rules from a large number of possible rules leads to the big data problem in multiple databases.

B. *MapReduce Based Association Rule Mining*

Many parallel and distributed algorithms have been proposed for the Apriori-based association rule mining. Most of them are having some limitations with respect to speed up, scalability, fault tolerance, high-availability, load balancing, data distribution etc. Among all the proposed method MapReduce based Apriori algorithm for association rule mining is becoming very much popular because of its processing capabilities [21]–[23]. MapReduce programming paradigm implemented by Apache Hadoop is most popular and cost-effective framework for solving the computational complexity of big data [24]. The main components of Hadoop which make it more scalable, robust and fault tolerance are *MapReduce* for processing, *HDFS* for storage and *YARN* for efficient task scheduling [25]. Hadoop can be more comfortably configured with the commodity hardware to prepare a parallel and distributed execution framework with a master-slave architecture [26]. The tasks using MapReduce are processed with the help of following steps:

1. Mapper: It generates map task by providing input in the form of key-value pairs. These key-value pairs are prepared

by input reader. The inputs can be taken from the database or flat files with many supported file format. The data are collected in the form of blocks which is by default 128MB in Hadoop-2.x.x and can be configurable. The input reader read each block and prepare the splits. Mapper generates map task based upon the split and outputs the result in a key-value pair. This is stored in the main memory buffer and then spills into the secondary storage.

2. Combiner: It usually play the role of local reducer and comes into action after the mapper. It is prepared with help of UDFs or same reducer may be used as the combiner and is optional to have a combiner. It is used in specific cases where each map task have repetition in intermediate keys. It collect all values of the key locally and sum them up and produce output to the reducer.

3. Shuffle and Sort: The mapper output is shuffled, sorted and grouped by keys in each slave machine. This step prepares a list by collecting values of similar keys so that each value of the key can be iterated and processed easily. It is used to shuffle and sort the output generated by the combiner.

4. Partitioner: A Partitioner is called before writing the final result by the reducer. A hash function is implemented to partition the intermediate key-value output from the map tasks to reduce tasks. The number of reducer task depends upon the number of partitioners. It is generally used to increase the efficiency in result analysis.

5. Reducer: The Reducer is used to produce the final output in HDFS which is in the form of key-value pair. Reducer generates reduce function and is invoked once for each distinct key and process their values presented in form of a list.

In general, the processing operation of MapReduce is performed through two functions known as *mapper* and *reducer*. The input and output are in the form of key-value pairs. Each mapper usually works on a single partition or more specifically file splits in parallel. The number of mappers for a particular task depends on number possible split in the dataset. The mappers tokenize each input split into a sequence of a key-value pair in parallel and produce the intermediate result in key-value pairs. The reducer receives output from each mapper as input, process it and produce the final result. The complete flow of job execution is given in Figure 1.

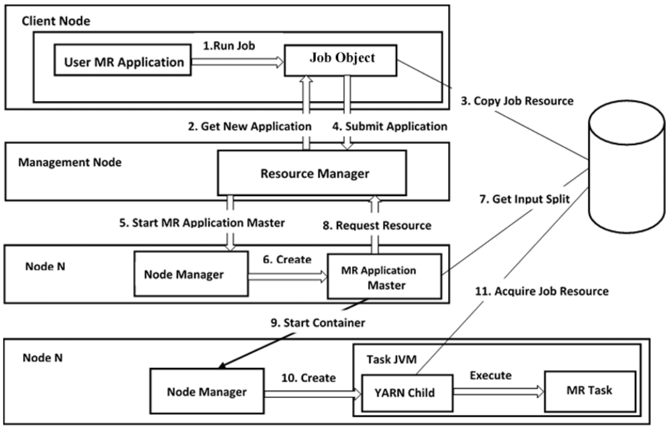

Fig. 1. YARN Job Execution work flow

There is no MapReduce based association rule mining technique available for synthesizing high frequent rules for the big database in a parallel and distributed environment. Our goal is to collect all the transactional data from multiple database sites into multi-partitioned HDFS storage. We will prepare a MapReduce based algorithm to select and synthesize high frequent association rules. The major challenges of mining hidden knowledge from this kind of environment are:

1. Collecting all the transactions from different databases to a multi-partitioned centralized storage system where processing can be done in a parallel and distributed fashion.

2. Finding all the high frequent association rules from the different partition.

3. Generate global high frequent rules from a large number of frequent itemset.

4. Synthesize the high frequent rules for a better DSS.

Based upon all above studies the problem statement is formulated by considering multiple large-scale databases distributed geographically over different locations, we have proposed (i) a model to collect and store data from multiple databases into multi-partitioned HDFS using Sqoop. (ii) MapReduce method for high-frequent rule selection based upon the transaction population (iii) Synthesizing these rules based on synthesized support and confidence using MapReduce. The experiments have been conducted for collecting data in multiple partitions of HDFS and executing the algorithms using MapReduce programming paradigm on top of Hadoop and YARN.

III. Methodology

Synthesizing high-frequency rule is mainly concern with large-scale multiple databases of a company. During this process, the rules are synthesized with the help of associated site weights. Many approaches have proposed to determine the weight of association rule that are associated with the individual sites of multiple databases. According to Wu and Zhang, a rule is said to be high frequency if it is rated or voted by multiple databases of an organization. Participation of each database site in DSS is determined with respect to the site weight. Our proposed technique for synthesis high frequent rule is suitable for multiple large-scale databases of an organization. In this paper a weighted multi-partitioned parallel and distributed model have been proposed for rule synthesis. The specified problem is considered to be a big data challenge because its capability to handle large multiple database sites. In order to make the process cost-effective and efficient, the rule selection and synthesis are carried out by using HDFS, MapReduce programming model and YARN of Hadoop with the help of a cluster of commodity computing system. The general steps are given as follows:

1. Collecting and storing huge amount of data from multiple databases to a multi-partitioned distributed data warehouse based upon HDFS.

2. Finding the high frequent k-itemset and association rules using MapReduce processing.

3. Select the association rules based on the synthesized rule weight and synthesize the global high frequent association rule

by ranking them based on their global minimum support and confidence using MapReduce programming paradigm.

A. *Collecting Data into HDFS*

It has been shown that many organization faces difficulty while collecting data from the geographically different database to a central site and analyze them for a better organizational DSS. In order to avoid this kind of problems, we propose HDFS based storage framework for a large-scale database in multiple partitions. We have used SQOOP to collect data from multiple databases into HDFS using the partition mechanism. Appropriate compression codecs can be used for increasing storage efficiency. This proposed model is very much cost effective because the large-scale data can be stored in a cluster of commodity machines. The proposed model for data collection is given in Figure 2.

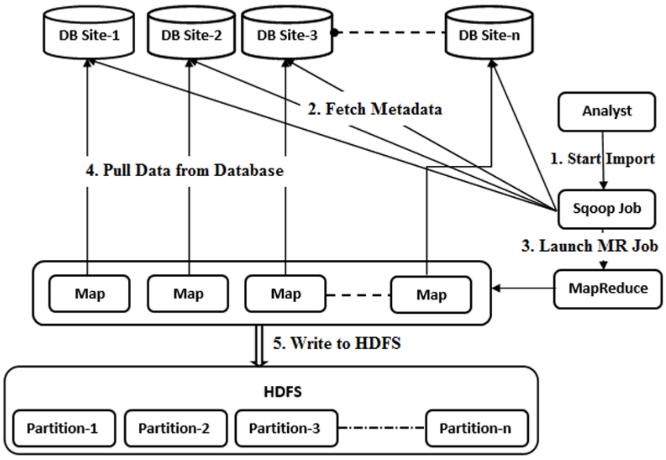

Fig. 2. Data Collection Model using SQOOP

B. *MapReduce Based Rule Selection and Synthesis*

Synthesizing high frequency association rule using MapReduce programming paradigm on top of Hadoop in a parallel and distributed environment is carried out with following steps.
Step-1 : Compute partition weight $W^a_{P_j}$ by using transaction population.
Step-2 : Generate all the local frequent k-itemset for each partition using MapReduce based Apriori algorithm.
Step-3 : Find $X.Supp_G$ for all the frequent k-itemset by using $X.Supp_G = \sum_{j=1}^{n} X.Supp(P_j) \times W^a_{P_j}$ in each partition P_j.
Step-4 : Generate the association rules in each partition based upon the $X.Supp_G$.
Step-5 : Apply *RuleSelectionMR* for selecting high-frequent association rules based upon the synthesized rule and partition weights.
Step-6 : Apply *RuleSynthesizerMR* for synthesizing high-frequent association rules based upon their global rule support and confidence.

The following notations are used in the rule selection and synthesis process.

- n : Number of rules
- m : Number of partitions
- t : Number of transactions
- $W^a_{R_i}$: Actual weight of rule R_i
- W^t_R : Total rule weight
- $W^s_{R_i}$: Synthesized weight of rule R_i
- $W^a_{P_j}$: Actual weight of the partition P_j
- W^t_P : Total partition weight
- $W^s_{P_j}$: Synthesized weight of partition P_j
- *minSynWt*: minimum rule synthesis weight.
- $R_i.Supp_S$: Synthesized support of R_i.
- $R_i.Conf_S$: Synthesized confidence of R_i.
- δ_1: *minimum synthesized support*
- δ_2: *minimum synthesized confidence*

The Association Rule selection and synthesis methods are having the following computations.

Synthesized Partition Weight: It is the ratio of partition weight to the sum of weights of all partitions. It can be computed as $W^s_{P_j} = \frac{W^a_{P_j}}{\sum_{j=1}^{m} W^a_{P_j}}$

Rule Weights: Rule weight $W^a_{R_i}$ is sum of weights of partitions containing the rule R_i which can be expressed as $W^a_{R_i} = \{\sum_{j=1}^{m} W^a_{P_j} | R_i \in P_j\}$

Total Rule Weights: It is the total weights of all the rules R_i in all the partition which can be computed as $W^t_R = \sum_{i=1}^{n} W^a_{R_i}$

Synthesized Rule Weights: The synthesized rule weight is the ratio of actual rule weight to the total rule weight. It is given as $W^s_{R_i} = \frac{W^a_{R_i}}{W^t_R}$

Assumption: From this point we assume that the actual partition weight and the association rule in each individual partition has been computed.

Example 1: Let us consider any 4 partition and corresponding weights, 5 selected rules are distributed in each partition along with their support and confidence. These values are tabulated in Table I.

TABLE I
DATA FOR EXAMPLE-1

P_i	$Rules(R_i)$	$R_i.Supp$	$R_i.Conf$	$W^a_{P_i}$
P_1	$R_1 : I_1 \to I_5$	0.2	0.5	0.8
	$R_2 : I_1 I_2 \to I_6$	0.3	0.5	
	$R_4 : I_1 I_3 \to I_4$	0.3	0.6	
P_2	$R_1 : I_1 \to I_5$	0.3	0.5	0.7
	$R_2 : I_1 I_2 \to I_6$	0.2	0.7	
	$R_3 : I_3 \to I_7$	0.2	0.4	
P_3	$R_1 : I_1 \to I_5$	0.3	0.5	0.5
	$R_4 : I_1 I_3 \to I_4$	0.2	0.6	
	$R_5 : I_3 I_5 \to I_6$	0.1	0.4	
P_4	$R_1 : I_1 \to I_5$	0.2	0.5	
	$R_5 : I_3 I_5 \to I_6$	0.3	0.5	0.5

Our proposed MapReduce based high-frequent rule selection and synthesis method are described in following steps.

Step-1: Rule weight count: The actual weight of the rules are calculated by considering the partition weights in which the rule has a participation. This is given as follows:

$W_{R_1}^{a} = 0.8 + 0.7 + 0.5 + 0.5 = 2.5$
$W_{R_2}^{a} = 0.8 + 0.7 = 1.5$
$W_{R_3}^{a} = 0.7$
$W_{R_4}^{a} = 0.8 + 0.5 = 1.3$
$W_{R_5}^{a} = 0.5 + 0.5 = 1.0$

The total rule weights is computed as $W_{R}^{t} = 2.5 + 1.5 + 0.7 + 1.3 + 1.0 = 7.0$

Step-2: Rule Selection: The rules are selected based upon their synthesized weight which is calculated as follows:

$W_{R_1}^{s} = \frac{1.5}{7.0} = 0.3571$ $\quad W_{R_2}^{s} = \frac{2.5}{7.0} = 0.2142$
$W_{R_3}^{s} = \frac{0.7}{7.0} = 0.1000$ $\quad W_{R_4}^{s} = \frac{1.3}{7.0} = 0.1857$
$W_{R_5}^{s} = \frac{1.0}{7.0} = 0.1428$

Let us consider the minimum synthesis rule weight(minSynthWt) a user-specified value for rule selection. If $minSynthWt > 0.15$ then the rules R_1, R_2, R_4 are selected and written into HDFS.

Step-3: Synthesizing rules: All the selected rules are now synthesized based upon their synthesized support and confidence. In order to compute these two value first, we have to calculate synthesized partition weights.

Total weight of all the partition is: $W_P{}^{t} = 0.8 + 0.7 + 0.5 + 0.5 = 2.5$

Synthesized weight of each partition are:

$$W_{P_1}^{s} = \frac{0.8}{2.5} = 0.32 \qquad W_{P_2}^{s} = \frac{0.7}{2.5} = 0.28$$
$$W_{P_3}^{s} = \frac{0.5}{2.5} = 0.20 \qquad W_{P_4}^{s} = \frac{0.5}{2.5} = 0.20$$

The synthesized support and confidence value of the selected rules R_1, R_2, R_4 are computed as follows:

$\underline{\boldsymbol{R_1 : I_1 \rightarrow I_5}}$
$R_1.Supp_S$
$=(0.32 * 0.2) + (0.28 * 0.3) + (0.2 * 0.3) + (0.2 * 0.2)$
$=0.248$
$R_1.Conf_S$
$=(0.32 * 0.5) + (0.28 * 0.5) + (0.2 * 0.5) + (0.2 * 0.5)$ $=0.5$
$\underline{\boldsymbol{R_2 : I_1 I_2 \rightarrow I_6}}$
$R_2.Supp_S = (0.32 * 0.3) + (0.28 * 0.2) = 0.152$
$R_2.Conf_S = (0.32 * 0.5) + (0.28 * 0.7) = 0.356$
$\underline{\boldsymbol{R_4 : I_1 I_3 \rightarrow I_4}}$
$R_4.Supp_S = (0.32 * 0.3) + (0.2 * 0.2) = 0.136$
$R_4.Conf_S = (0.32 * 0.6) + (0.2 * 0.6) = 0.312$

Now let us assume $\delta_1 = 0.13$ and $\delta_2 = 0.3$, with this constraints the rules R_1, R_2, R_4 are synthesized for organizational DSS.

C. Design of Algorithm

In this section, we have proposed two parallel and distribute algorithms known as *RuleSelectionMR* and *RuleSynthesizeMR* on Hadoop framework by using MapReduce for processing and HDFS for storage. All the frequent k-itemset and Association Rules are generated by using MapReduce based Apriori algorithm on YARN framework.

1) MapReduce Based Rule Selection: In this process, rules will be selected parallelly from different partitions using MapReduce programming approach. It uses mapper and reducer for selecting rules parallelly in all partition. Before synthesizing the rules it has to select association rule from different partition based on the rule strength. In order to determine the rule strength, we have to find synthesized weight $W_{R_i}^{s}$ of the rule R_i and effective rules are selected for synthesis process. The task of rule selection will be carried out with the help of two MapReduce jobs to perform the task parallelly in all partitions. The main purpose of both tasks are:

- Calculate $W_{R_i}^{a}$ and W_{R}^{t}.
- Calculate $W_{R_i}^{s}$ and select rule based on minSynWt.

These two tasks will be carried out by using two proposed MapReduce algorithm:

- *Algorithm 1: RuleWeightCountMR.*
- *Algorithm 2: RuleSelectionMR.*
- *Algorithm 3: RuleSynthesizeMR*

a) ***RuleWeightCountMR****:* This MapReduce based algorithm is used to find actual weight $W_{R_i}^{a}$ of the rule R_i in each partition and the total weight of all the rules W_{R}^{t}. This algorithm has been initiated by supplying partition-wise association rules generated by the MapReduce based Apriori algorithm from HDFS along with $W_{P_j}^{a}$ as an input argument. The MapReduce based pseudo code for mapper and reducer is given in Algorithm 1.

Mapper: The Mapper of Algorithm 1 tokenize each rule R_i in all partition P_j and produce the rule R_i and actual partition weight $W_{P_j}^{a}$, in the form of $< R_i, W_{P_j}^{a} >$ as a $< K, V >$ pair.

Reducer: The Reducer of Algorithm 1 will receive input from the Mapper which is in the form of $\{R_i, List < W_{P_j}^{a} >\}$ as $< K, V >$ pair and compute the sum of all the $W_{P_j}^{a}$ in the List. It is the actual weight $W_{R_i}^{a}$ of the rule R_i. This reducer is also used to calculate the total weight of all the rules by adding $W_{R_i}^{a}$. The reducer pairs $< W_{R_i}^{a}, W_{R}^{t} >$ and emit output in the form of $< R_i, [W_{R_i}^{a}, W_{R}^{t}] >$ as $< K, V >$ pair.

b) ***RuleSelectionMR****:* This algorithm is used to calculate the synthesized weight of the rules $W_{R_i}^{s}$ in each partition by using $W_{R_i}^{a}$ and W_{R}^{t}. After computing $W_{R_i}^{s}$ the rules are selected based upon the *minSynWt*. This algorithm read output written by the reducer of *RuleWeightCountMR* algorithm. The pseudo code for this MapReduce approach is written in Algorithm 2.

Mapper: The mapper of Algorithm 2 read lines from HDFS which are in the form of $< R_i, pair[W_{R_i}^{a}, W_{R}^{t}] >$, and split each line into tokens such that the rule R_i and $pair[W_{R_i}^{a}, W_{R}^{t}]$ are separated as $< K, V >$ pairs and supplied to the reducer.

Reducer: The Mapper write the output in a temp partition in the main memory. The reducer of Algorithm 2 receives input from mapper in the form of $< R_i, pair[W_{R_i}^{a}, W_{R}^{t}] >$ as $< K, V >$ pairs, separate $W_{R_i}^{a}$ and W_{R}^{t} from value part. Finally compute the synthesized rule weight $W_{R_i}^{s}$ and write the rules which satisfies *minSynWt*.

```
Algorithm 1 RuleWeightCountMR
Require: Partition wise all rules from HDFS.
Ensure: Rule $R_i$, $W^a_{R_i}$ and $W^t_R$
class RuleWeightCountMapper {
Initialize $W^a_{P_j}$ = Weight of $P_j$
Initialize total rule weighy $W^t_R$ = 0
procedure MAPPER(PartitionId pid, Rules R)
    for each partition $P_j \in HDFS$ do
        for each Rule $R_i \in P_j$ do
            Tokenize each Rule $R_i$
            Emit($R_i$, $W^a_{P_j}$)
        end for
    end for
end procedure
}
class RuleWeightCountReducer {
procedure REDUCER(Rule $R_i$, List< $W^a_{P_j}$ >)
    for each $W^a_{P_j} \in$ List< $W^a_{P_j}$ > do
        $W^a_{R_i} += W^a_{P_j}$
        $W^t_R += W^a_{R_i}$
    end for
    Emit($R_i$, pair[$W^a_{R_i}$, $W^t_R$])
end procedure
}
```

```
Algorithm 2 RuleSelectionMR
Require: Rules produced by Algorithm-1, and minSynWt
Ensure: Selected set of Rules $R_i$
class RuleSelectionMapper {
Initialize synthesized rule weight $W^s_{R_i}$ = 0
procedure MAPPER(RuleId id, Rules R)
    for each Rule $R_j$ do
        Separate $R_i$ and $< W^a_{R_i}, W^t_R >$
        Emit($R_i$, pair[$W^a_{R_i}$, $W^t_R$])
    end for
end procedure
}
class RuleSelectionReducer {
procedure REDUCER($R_i$, List< $[W^a_{R_i}, W^t_R]$ >)
    for each $W^a_{P_j} \in$ List< $W^a_{P_j}$ > do
        Extract $W^a_{R_i}$ and $W^t_R$
        $W^s_{R_i} = \frac{W^a_{R_i}}{W^t_R}$
        if $W^s_{R_i} > minSynWt$ then
            Emit($R_i$, $W^s_{R_i}$)
        end if
    end for
end procedure
}
```

2) MapeRduce Based Rule Synthesis : In the MapReduce based rule selection method, association rules are selected by taking the synthesized rule weight into account from all partitions in a parallel and distributed environment. Now each partition should contain only the association rule selected by the *RuleSelectionMR* algorithm. These selected rules from each partition are synthesized with the help of local support and confidence value. Finally, the synthesized support and confidence are used to determine high-frequent rules. The pseudo-code is given in Algorithm 3.

```
Algorithm 3 RuleSynthesizeMR
Require: Selected rules from each partition, $W^s_{P_j}$, $\delta_1$: minimum synthesized support and $\delta_2$: minimum synthesized confidence
Ensure: Synthesized high-frequent Rules $R_i$ with $R_i.Supp_S$ and $R_i.Conf_S$
class RuleSynthesizeMapper {
$R_i.Supp_S = 0$                         ▷ Synthesized support $R_i$
$R_i.Conf_S = 0$                         ▷ Synthesized confidence $R_i$
procedure MAPPER(PartitionId pid, Rules R)
    for each Partition $P_j$ do
        for each Rule $R_i \in P_j$ do
            Split $R_i$
            get $R_i.Supp$                ▷ Local supp. of $R_i$
            get $R_i.Conf$                ▷ Local conf. of $R_i$
            $supp = W^s_{P_j} \times R_i.Supp$
            $conf = W^s_{P_j} \times R_i.Conf$
            Emit($R_i$, $[supp, conf]$)
        end for
    end for
end procedure
}
procedure REDUCER($R_i$, $List < [supp, conf] >$)
    for each $[s, c] \in List < [supp, conf] >$ do
        split $R_i$
        $R_i.Supp_S += s$
        $R_i.Conf_S += c$
        if $R_i.Supp_S > \delta_1 and R_i.Conf_S > \delta_2$ then
            Emit($R_i$, $[R_i.Supp_S, R_i.Conf_S]$)
        end if
    end for
end procedure
}
```

Mapper: This is the first phase of rule synthesis process. Mapper of Algorithm 3 read each line of the partition containing only selected rules. The rules in each partition are associated with local support $R_i.Supp$ and confidence $R_i.Conf$ value respectively. Each line of rule from the partition is tokenized by separating rule R_i, $R_i.Supp$ and $R_i.Conf$. The synthesized weight of each partition $W^s_{P_j}$ is multiplied with $R_i.Supp$ and $R_i.Conf$ separately. These two values are paired and finally supplied to the reducer along with R_i in form of $< R_i, [supp, conf] >$ as $< K, V >$ pairs where *supp* and *conf* are the temporary value given by the mapper.

Reducer: This reducer of Algorithm 3 take list of support and confidence value $List < [supp, conf] >$ along with rule R_i as $< K, V >$ pair as input. It computes the synthesized support $R_i.Supp_S$ and confidence $R_i.Conf_S$ values. Finally synthesized high-frequent rules are produced.

TABLE II
SYSTEM CONFIGURATION

Hardware and Software platform used	
Hardware Specification	
Processor	Intel i5 2.5Ghz
Memory	4GB DDR3
HDD	500GB
Software Specification	
O. S	CentOS-7
Hadoop	2.6.0-cdh-5.5.1
Sqoop	1.4.6-cdh-5.5.1
Hive	1.1.0-cdh-5.5.1

IV. RESULT ANALYSIS

Performance of the proposed algorithm is evaluated by considering different parameter like the number of partitions, varying synthesized support and rule synthesizing time. The experimental setup is made with the help of a Hadoop cluster with 10 computing nodes. The configuration of our parallel and distributed computing cluster is given in Out of the 10 computing node in the Hadoop cluster, one is used as the master node with 16GB of primary memory. The YARN framework of Hadoop is chosen because of its scheduling and load balancing capability.

A. Data Collection and Preprocessing

A large amount of data has been collected from 4 different retail sources to perform the analysis. The sources are having multiple branches and each site maintains their own data in HP VERTICA and ORACLE database. All the data have been collected to the 10 node Hadoop cluster with the help of Sqoop.// The data from a particular site is collected to a partition in HDFS and then the replication factor of the partition has been set to 3.

TABLE III
CHARACTERISTICS OF 1ST DATASET

PNO	NT	NI	k-FI	ALT	NAR
P1	500K	2235	18	16	30271
P2	726K	5892	17	14	51737
P3	643K	4732	15	10	43859
P4	600K	5031	15	9	32539
P5	700K	8326	17	12	38700
P6	500K	3183	15	12	27043
P7	650K	5320	14	10	21800
P8	680K	6263	12	12	22742
P9	720K	6527	15	10	28320
P10	810K	8425	18	14	47631

The data in each partition contains a significant number of the nulls and short transaction. All these may impart an unexpected result during the analysis. So they need to be preprocessed by eliminating all the anomalies. This data cleansing operation has been done by MapReduce program. The association rules from all the partition have also been generated with the help of MapReduce based Apriori algorithm which has been implemented using Java. The characteristics of the collected data from 10 different data site have been given in the table III. It represents Partition number, Number of Transaction in each partition(NT). Number of Items(NI), frequent k-itemset(k-FI), Average Length of the Transaction (ALT) and Number of Association Rule Generated(NAR).

Apart from the above mentioned dataset we have also collected three other datasets like Retail from the anonymous store of Belgian retail supermarket, BMS-Web-View-1 and BMS-Web-View-2 can be found from KDD CUP 2000. The characteristics of these datasets are given in the following tables with their NT, NI, ALT, and AFI.

TABLE IV
CHARACTERISTICS OF 2ND DATASET

Dataset	NT	NI	ALT	AFI
Retail	88,162	10,000	11.31	99.67
BMS-Web-Wiew-1	1,49,639	1922	2.00	155.71
BMS-Web-Wiew-2	3,58,278	100	2.00	7165.56

For these datasets, 10 partitions are prepared by considering n-lines or n-Transactions per partition. Partitions are prepared with the help of split for the collected datasets of the table: IV. For preparing the split we have overridden the *NLineInputFormat* class by our own *NLineRecordReader* instead of default *LineReader* which is by default having 1 line per mapper. Thus we prepare our own split by assigning a fixed amount of transaction per split by using the *setNumLinesPerSplit()* method of *NLineInputFormat* class as follows.

$$NLINESTOPROCESS = getNumLinesPerSplit(context)$$

B. Job Execution Work-Flow

A comparative study between our proposed *RuleSynthesisMR* and *RuleSynthesis* algorithm in [18] has been done. Our total task is carried out with the help of three algorithms known as *RuleWeightCountMR*, *RuleSelectionMR* and *RuleSynthesisMR* which are completely written in Java-based MapReduce program. The job workflow is designed using OOZIE scheduler to carry out the task execution in Hadoop which is given in Figure 3.

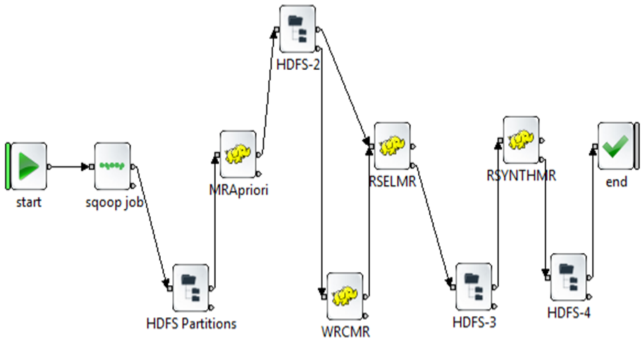

Fig. 3. Job Execution Workflow

C. Effect of Number of Partition in Performance

Figure 4 describes the performance of *RuleSynthesisMR* and *RuleSynthesis* by considering number of partition and execution time in seconds. The execution time has been determined by dumping all 5GB of transactional data into HDFS after preprocessing. All these data are collected from 10 different database site of a retail company into HDFS. The block size is configured to 128MB and approximately we have around 42 blocks in HDFS and the replication factor is set to 3. The split size is also configured to the max(block size) i.e 128MB. The *RuleSynthesisMR* algorithm is executed in a distributed Hadoop cluster with 10 data node. But the *RuleSynthesis* algorithm is executed in a single database server. It has been shown that our algorithm outperforms by increasing the partition number because of the distributed computing environment of Hadoop with YARN.

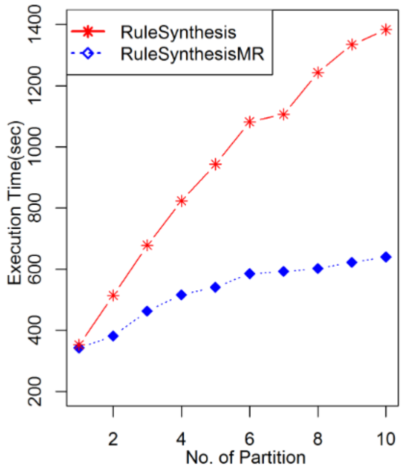

Fig. 4. Execution Time(sec)vs No. of Partition

D. *Effect of Synthesis Rule Support*

In the rule synthesis process the minimum synthesis rule support has been considered as an important parameter for the performance evaluation. Figure 5 shows the performance based upon the synthesis rule support and execution time between two algorithms. It has been shown that with an increase in synthesis rule support our proposed *RuleSynthesisMR* algorithm takes less time as compared to *RuleSynthesis* algorithm. Our proposed algorithm has shown better performance with varying synthesized rule support. Because of the parallel execution environment, multiple map tasks were distributed among different datanodes where rules have been generated and pruned independently. The high-frequency rules have been aggrigated and synthesized based upon the synthesized rule support.

E. *Effect of Number of Computing Nodes*

Figure 6 shows the performance of *RuleSynthesisMR* algorithm with 10 data nodes. It is clear that by increasing the number of data node execution time decreases. This is because of the parallel processing of the mapper and reducer.

It has been shown that by applying the *RuleSynthesizeMR* algorithm with the 2nd collection of the dataset as given in Table IV the performance has been degraded a lot as compared to *RuleSynthesizing* algorithm. This is because of many small splits prepared for analysis in HDFS. It has been shown that with larger dataset split our proposed algorithm performs better than the existing high-frequency rule synthesis algorithm.

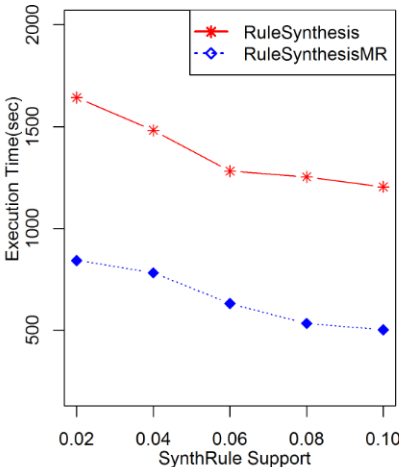

Fig. 5. Execution Time(sec) vsSynthesized Rule Support

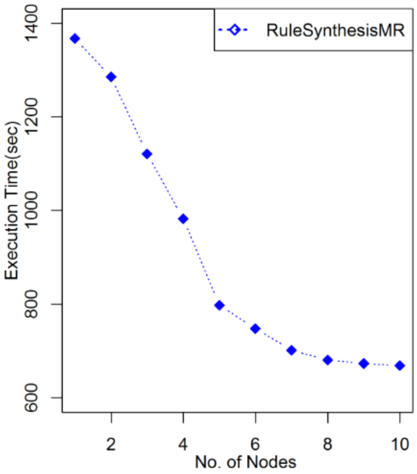

Fig. 6. Exec. Time(sec) vs No. of DataNode

F. *Effect of Transaction length*

The impact of transaction lenth measures the performance of RuleSynthesis and RuleSynthesisMR algorithm. The synthetic dataset T90-P5-I100k-C0.25-D10M having 18014 number of items with varying transaction length is used for the experimental analysis. It has been shown that longer transaction generates large number of long frequent patterns. Because of the parallel execution RuleSynthesisMR takes less time as compared to the RuleSynthesis algorithm as show in Figure 7. It has been observed that sometime the long transactions lead to memory inefficiency problem and it has been considered very carefully. For longer transaction the minimum support value has been adjusted and the split size has been controlled in the Mapper class for increasing the efficiency of the proposed approach. By considering these two-synthetic large datasets the length of transaction w.r.t execution time has been recorded. It has also been found that without transaction pruning the proposed RuleSynthesisMR algorithm takes very long time and leads to halt the total Hadoop cluster. The possible performance bottleneck of RuleSynthesisMR algorithm basically depends on the generation of high-frequent association rules from different sites and assignment of the rule weight. If the high-frequent rules are generated without any memory constraints and with a balanced parallelism then further performance can be improved considerably.

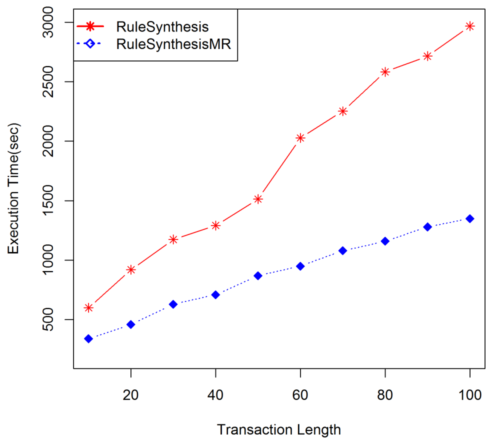

Fig. 7. Execution Time vs Transaction Length

G. Accuracy

The accuracy of our proposed rule synthesis algorithm is determined by considering different minimum synthesis rule support value for synthesizing high-frequent rules. It has been shown in Figure 8 that our proposed algorithms achieve the accuracy as compared to the Rule Synthesis algorithm proposed in [18] where collection of weighted high-frequent rules from different data sources was considered. With the varying rule synthesis support from 0.01 to 0.14 our proposed algorithm achieves better accuracy.

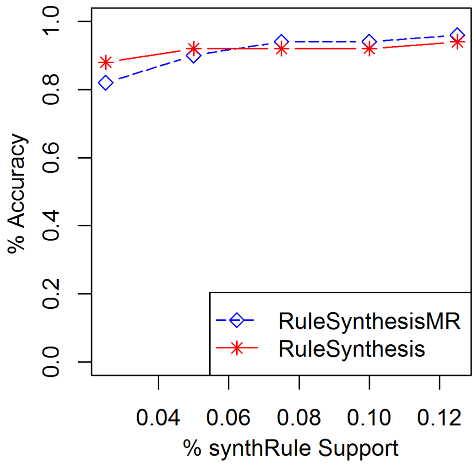

Fig. 8. Accuracy vs Synthesized Rule Support

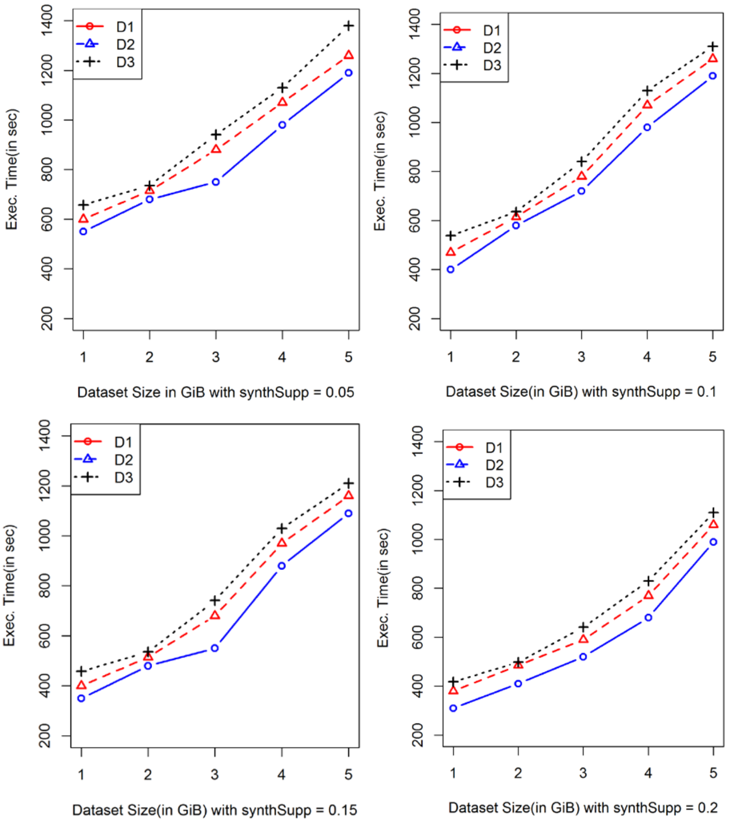

Fig. 9. Execution Time(sec) vs Synthesized Rule Support

H. Scalability

Scalability is used to determine the adaptability of our proposed method for high-frequent rule synthesis in different dataset with varying dataset size, synthesis rule support and with fixed average transaction length. The study of scalability has been performed on execution time by varying the dataset size in GiB by considering the dataset given in Table V. Figure 9 shows the execution time by varying the size of dataset from 1GB to 5GB and synthesis rule support with value 0.05, 0.1, 0.15 and 0.2. It has been shown that the proposed method maintain scalability when applied on the 10 node Hadoop cluster having 3 different datasets i.e. the execution time remains almost same for all the three-synthetic dataset generated by IBM Synthetic data generator.

TABLE V
SYNTHETIC DATASET FOR TESTING EFFICIENCY

ID	Dataset Name	NI	ANI/T	Size(GiB)
D1	T80-P5-I100k-C0.25-D10M	18012	79.9	4.7
D2	T90-P5-I100k-C0.25-D10M	18014	89.9	5.3
D3	T100-P5-I100k-C0.25-D10M	18015	99.9	5.9

V. CONCLUSION

Many organizations have multiple sites and each site should have a certain important contribution to organizational DSS. The participation of each site in the DSS is decided through transaction population so as the site weight. Our proposed model is very much efficient in collecting and storing many frequent rules from multiple database sites into a big data warehouse. To synthesize high-frequency rules in a more efficient, accurate and cost-effective manner a weighted multi-partitioned model has been proposed by using a cluster of commodity hardware. In this paper, we have successfully collected all the data from multiple databases into multi-partitioned warehouse based on HDFS. The data collection is carried out using Sqoop without loss of granularity. Sqoop pulls the data from multiple databases parallelly by using MapReduce approach. We have implemented the MapReduce base rule selection and synthesize algorithm in the parallel and distributed environment using Hadoop with YARN. All the high-frequency rules are selected based on the transaction population and partition weight. Two algorithms namely RuleWeightCountMR and RuleSelectionMR have been proposed for this purpose. These selected high-frequency rules are synthesized based on the minimum synthesized support and confidence threshold respectively. This is carried out through RuleSynthesizeMR. The Performance scales linearly with increase in the computing node. Multiple replications of each partition in the data node achieve high availability for the processing environment. It has been ensured that the proposed distributed storage and parallel processing model is much more efficient and most importantly cost effective with commodity hardware. So, our MapReduce based approach provides an effective solution for global organizational DSS for the companies having geographically multiple distributed database sites.

References

[1] R. Agrawal and J. C. Shafer, "Parallel mining of association rules," *IEEE Transactions on Knowledge and Data Engineering*, vol. 8, no. 6, pp. 962–969, 1996. [Online]. Available: https://dx.doi.org/10.1109/69.553164

[2] D. W. Cheung, V. T. Ng, A. W. Fu, and Y. Fu, "Efficient mining of association rules in distributed databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 8, no. 6, pp. 911–922, 1996. [Online]. Available: https:g//dx.doi.org/10.1109/69.553158

[3] F. Wei and B. Albert, "Mining big data: Current status, and forecast to the future," *ACM SIGKDD Explor. Newslett.*, vol. 14, no. 2, pp. 1–5, 2012. [Online]. Available: https://dx.doi.org/10.1145/2481244.2481246

[4] R. Didugu and D. A. Devarakonda, "A framework for exploring algorithms for big data mining," *Indian Journal of Science and Technology*, vol. 9, 05 2016. [Online]. Available: https://dx.doi.org/10.17485/ijst/2016/v9i17/93017

[5] ——, *Adding Big Value to Big Businesses: A Present State of the Art of Big Data, Frameworks and Algorithms*, 01 2018, pp. 171–184. [Online]. Available: https://dx.doi.org/10.1007/978-981-10-6602-3_17

[6] M. J. Zaki, *Parallel and Distributed Data Mining: An Introduction*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 1–23. [Online]. Available: https://dx.doi.org/10.1007/3-540-46502-2_1

[7] W. Tom, *Hadoop–The Definitive Guide 4th Edition*. O'Reilly, 2015.

[8] V. Vasantham and D. Haritha, "A survey on cost minimization techniques for big data processing," *Journal of Advanced Research in Dynamical and Control Systems*, vol. 10, pp. 547–551, 01 2018.

[9] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 2010, pp. 1–10. [Online]. Available: https://dx.doi.org/10.1109/MSST.2010.5496972

[10] K. Rao, S. Subramani, M. Prasad, and A. Saravanan, "Technical challenges and perspectives in batch and stream big data machine learning," *International Journal of Engineering and Technology*, vol. 7, p. 48, 12 2017. [Online]. Available: https://dx.doi.org/10.14419/ijet.v7i1.3.9225

[11] E.-H. Han, G. Karypis, and V. Kumar, "Scalable parallel data mining for association rules," *SIGMOD Rec.*, vol. 26, no. 2, pp. 277–288, 1997. [Online]. Available: https://dx.doi.org/10.1145/253262.253330

[12] R. Grossman, S. Bailey, A. Ramu, B. Malhi, and A. Turinsky, "The preliminary design of papyrus: A system for high performance, distributed data mining over clusters," *Advances in Distributed and Parallel Knowledge Discovery*, pp. 259–275, 2000.

[13] H. Liu, H. Lu, and J. Yao, "Toward multidatabase mining: identifying relevant databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 4, pp. 541–553, 2001. [Online]. Available: https://dx.doi.org/10.1109/69.940731

[14] X. Wu, C. Zhang, and S. Zhang, "Database classification for multi-database mining," *Inf. Syst.*, vol. 30, no. 1, pp. 71–88, 2005. [Online]. Available: https://dx.doi.org/10.1016/j.is.2003.10.001

[15] S. Zhang, X. Wu, and C. Zhang, "Multi-database mining," *IEEE Computational Intelligence Bulletin*, vol. 2, no. 1, pp. 5–13, 2003.

[16] Z. Chengqi, L. Meiling, N. Wenlong, and Z. Shichao, "Identifying global exceptional patterns in multi-database mining," *IEEE Intelligent Informatics Bulletin*, vol. 3, no. 1, pp. 19–24, 2004.

[17] X. Wu and S. Zhang, "Synthesizing high-frequency rules from different data sources," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 2, pp. 353–367, 2003. [Online]. Available: https://dx.doi.org/10.1109/TKDE.2003.1185839

[18] T. Ramkumar and R. Srinivasan, "Modified algorithms for synthesizing high-frequency rules from different data sources," *Knowl. Inf. Syst.*, vol. 17, no. 3, pp. 313–334, 2008. [Online]. Available: https://dx.doi.org/10.1007/s10115-008-0126-6

[19] R. Thirunavukkarasu, H. Shanmugasundaram, and S. Shanmugam, "Synthesizing global negative association rules in multi-database mining," *International Arab Journal of Information Technology*, vol. 11, no. 6, pp. 526–531, 2014.

[20] A. Adhikari, L.C.Jain, and S.Ramanna, "Analysing effect of database grouping on multi-database mining," *IEEE Intelligent Informatics Bulletin*, vol. 12, no. 1, pp. 25–32, 2011.

[21] L. Ning, Z. Li, H. Qing, and S. Zhongzhi, "Parallel implementation of apriori algorithm based on mapreduce," *International Journal of Networked and Distributed Computing*, vol. 1, no. 2, pp. 89–96, 2013. [Online]. Available: https://dx.doi.org/10.1109/SNPD.2012.31

[22] X. Yang, Z. Liu, and Y. Fu, "Mapreduce as a programming model for association rules algorithm on hadoop," in *Proceedings of the 3rd International Conference on Information Sciences and Interaction Sciences*, ser. ICIS '10. IEEE, 2010, pp. 99–102. [Online]. Available: https://dx.doi.org/10.1109/ICICIS.2010.5534718

[23] Z. Farzanyar and N. Cercone, "Efficient mining of frequent itemsets in social network data based on mapreduce framework," in *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, ser. ASONAM '13. ACM, 2013, pp. 1183–1188. [Online]. Available: https://dx.doi.org/10.1145/2492517.2500301

[24] J. Dean and S. Ghemawat, "Mapreduce: A flexible data processing tool," *Commun. ACM*, vol. 53, pp. 72–77, 2010. [Online]. Available: https://dx.doi.org/10.1145/1629175.1629198

[25] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, ser. SOCC '13, 2013, pp. 1–16. [Online]. Available: https://dx.doi.org/10.1145/2523616.2523633

[26] A. Vishwanath and R. Murugan, "Parallel processing on big data in the context of machine learning and hadoop ecosystem: A survey," *International Journal of Engineering and Technology*, vol. 7, p. 577, 03 2018. [Online]. Available: https://dx.doi.org/10.14419/ijet.v7i2.7.10885