IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 3, 2021, accepted September 17, 2021, date of publication September 24, 2021,
date of current version October 8, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3115514

A Distributed Method for Fast Mining Frequent Patterns From Big Data

PENG-YU HUANG[1], WAN-SHU CHENG[2], JU-CHIN CHEN[1], WEN-YU CHUNG[1], YOUNG-LIN CHEN[3], AND KAWUU W. LIN[1]

[1]Department of Computer Science and Information Engineering, National Kaohsiung University of Science and Technology, Kaohsiung 807618, Taiwan
[2]Department of Electrical Engineering, National Kaohsiung University of Science and Technology, Kaohsiung 807618, Taiwan
[3]Foxconn Technology Group, Taipei 114699, Taiwan

Corresponding author: Kawuu W. Lin (linwc@nkust.edu.tw)

This work was supported by the Ministry of Science and Technology of Taiwan under Grant 109-2221-E-992 -072 -MY3.

ABSTRACT In recent years, knowledge discovery in databases provides a powerful capability to discover meaningful and useful information. For numerous real-life applications, frequent pattern mining and association rule mining have been extensively studied. In traditional mining algorithms, data are centralized and memory-resident. As a result of the large amount of data, bandwidth limitation, and energy limitations when applying these methods to distributed databases, especially in this era of big data, the performance is not effective enough. Hence, data mining on distributed environments has emerged as an important research area. To improve the performance, we propose a set of algorithms based on FP growth that discover FPs that are capable of providing fast and scalable service in distributed computing environments and a brief data structure to store items and counts to minimize the data for transmission on the network. To ensure completeness and execution capability, DistEclat and BigFIM were considered for the experiment comparison. Experiments show that the proposed method has superior cost-effectiveness for processing massive datasets and good capabilities under various experiment conditions. The proposed method on average required only 33% of the execution time and 45% of the transmission cost of DistEclat. Compared to BigFIM, The proposed method on average required 23.3% of the execution time and 14.2% of the transmission cost of BigFIM.

INDEX TERMS Data mining, parallel algorithms, distributed computing.

I. INTRODUCTION

Knowledge discovery in databases provides a powerful capability to discover meaningful and useful information. It improves the efficiency of business operations and has been implemented in a variety of industrial environments. Numerous real-life applications have resulted in several data mining [1], [2] tasks, such as association rule mining, classification, clustering, and sequential pattern mining. For numerous real-life applications, frequent pattern (FP) mining and association rule mining have been extensively studied.

In recent years, artificial intelligence (AI) has made machines smart, giving them the power to make certain decisions without human intervention. AI and the Internet of Things work together to collect and analyze data and automate decision-making. With the coming era of AI of Things, data are growing much more rapidly.

The associate editor coordinating the review of this manuscript and approving it for publication was Hong-Mei Zhang.

Two well-known algorithms, namely, Apriori [3] and FP growth [4], are proposed to mine frequent itemsets and association rules on the basis of generation and test or pattern growth approaches. The Apriori approach generates a great number of candidate datasets and repetitively scans a database to verify whether a pattern appears frequently or not. Hen *et al.* proposed the FP-growth method based on FP tree for mining FPs. FP growth scans the databases only twice and the mined information could be obtained from the proposed data structure. However, finding FPs increases the execution time when the database size is large.

In traditional mining algorithms, data are centralized and memory-resident. As a result of the large amount of data, bandwidth limitation, and energy limitations when applying these methods to distributed databases, especially in this era of big data, the performance is not effective enough. Hence, data mining on distributed environments has emerged as an important research area. The use of field programmable gate array (FPGA) for implementing parallel computing was also proposed by Tehreem *et al.* [5].

Previous research also illustrated that FP tree-like algorithms performed better than Apriori-like algorithm based on distributed, parallel computing, and Hadoop techniques, as well as other algorithms based on the well-known framework Apache Spark, such as PFP (parallelize the FP-Growth) algorithm [6] based on Hadoop, PIFP-Growth (parallelized incremental FP-Growth) [7], R-PFP (recursive-PFP) [8], MR-PFP (MapReduce-based parallel frequent pattern growth) [9], PBFP-Growth (parallel block FP-Growth algorithm) [10], MISFP-growth (multiple item support frequent patterns) [11], BigFIM (big frequent itemset mining) [12], and S-FPG (Spark FP-Growth) [13]. Some research proposed approaches to mining frequent itemsets from secondary memory when the database or the data structures used in the mining are too large to fit in the main memory [2], [14]. These research provide new opportunities but pose some challenges, such as a long execution time and redundant execution, thus resulting in waste during data mining.

Two frequent itemset mining algorithms, namely, DistEclat (dist-equivalence class clustering and bottom-up lattice traversal) and BigFIM for the MapReduce platform, were proposed. DistEclat partitions the search space more evenly among different processing units through prefixes to balance the workload for each mapper. BigFIM combines the principles from Apriori to mine frequent itemsets first to support the DistEclat method when mining on large datasets. DistEclat and BigFIM have been reported to outperform PFP. However, DistEclat and BigFIM need additional parameters to complete the mining, which is difficult and impractical for users.

The DP (database projection) [2] and aggressive projection [14] algorithms were proposed, using a disk-based structure to deal with the scalability problem, thereby ensuring the mining can be completed. During the mining process, the projected database associated with each frequent item needs to be loaded into the main memory. Then, large databases are materialized on the disk in different projected databases whose dimension fits in the main memory. If the condition is not allowed, then the algorithm would activate the condensed process again. The algorithm requires additional execution performance because the efficiency for memory is much better than that for the disk.

The discussed methods focus on FP mining, which is why improvements in execution time efficiency have been proposed. Parallel and distributed computing techniques have attracted attention because of their ability to manage and compute large amounts of data. However, these studies all have the same characteristics: high amount of data transmission time, high memory cost, high scanning cost expended by the database to discover FPs, and redundant execution time cost by unadaptable nodes. Neither Hadoop MapReduce nor Apache Spark addresses these critical problems at the same time. To improve the execution time and the redundant execution cost, we propose a distributed and parallel computing method called DFP (distributed frequent pattern mining). The basic flow of DFP is as follows. FP growth is performed in the first stage and the information for the necessary memory sizes on the database is recorded at the same time. If the FPs can be extracted successfully in the first stage, then the algorithm is finished. Otherwise, the algorithm is continued to estimate the required nodes for the parallel computing according to the recorded information, and delivers workload to nodes to complete mining.

The primary contributions of this study are (1) a set of algorithms based on FP growth that discover FPs that are capable of providing fast and scalable service in distributed computing environments and (2) a brief data structure to store items and counts to minimize the data for transmission on the network.

To ensure completeness and execution capability, DistEclat and BigFIM were considered for the experiment comparison. Experiments show that the proposed method has superior cost-effectiveness for processing massive datasets and good capabilities under various experiment conditions.

This paper is organized as follows: Section II presents the background. Section III presents our proposed algorithm called DFP and explains how the DFP is used for tree-based FP mining. Section IV focuses on an analytical evaluation of the complexity of the proposed method in comparison with other similar algorithms and shows experimental results. Section V provides conclusions and future work.

II. RELATED WORK

Past studies are reviewed in two categories: A) association rule mining and B) distributed algorithms for discovery of FPs, including the most efficient algorithms, DistEclat and BigFIM, for mining FPs in distributed computing environments.

A. ASSOCIATION RULE MINING

Numerous studies have been conducted on data mining in recent years (Agrawal *et al.*, 1993, 2001; Agrawal and Srikant, 1995; Bayardo, 1997; Ester *et al.*, 1996). The problem of association rule mining is the most discussed topic for wide applications by studies on data mining techniques. The problem was proposed by Agrawal *et al.* (1993), which is defined below. Let $DB = \{T1, T2, \ldots, Tn\}$ be a database (DB) containing n transactions, and $I = \{i1, i2, \ldots, im\}$ be a set of items. For each transaction Tj *in* DB, $Tj \subseteq I$, an association rule is represented as $X => Y$, where $X \subset I$, $Y \subset I$, and $X \cap Y = \Phi$. The support of itemset X is the number of transactions in DB that contain X. The rule $X => Y$ holds with confidence C if $C\%$ of transactions in DB contain both X and Y. The FPs are the itemsets whose support values are greater than or equal to a support threshold. To efficiently discover the association rules, the first step is to discover all the FPs.

The solutions applied by past studies that discussed the problem of mining FPs can be classified into two types: Apriori-like (Agrawal *et al.*, 1993, 1996) and FP growth-like (Han *et al.*, 2000b). The Apriori-like algorithms generate the candidates and scan the database repetitively to discover

the FPs. The inefficiency of this kind of algorithm is caused by the required large memory for generating candidates and the multiple scans on the database. Han *et al.* (2000a) proposed a tree-based data structure named FP tree and the corresponding mining algorithm named FP growth for discovering FPs. The algorithm requires two scans on the database to complete the mining task. The first scan is performed to calculate the support of each item. It also creates a header table that records the item name, its corresponding support, and the first node-link linking to the first node in the FP tree carrying the same item name. The items of the header table are sorted in descending order by the support. In the second scan, for each transaction, the items with support smaller than the threshold are filtered and the remaining items are sorted in descending order by the support value. The sorted items of each transaction are inserted into the tree, namely, the FP tree. The structure of an FP tree consists of a root node labeled as null, a set of item prefix subtrees as the children of the root, and a header table. The structure of the nodes of the FP tree is $< item-name, count(support), node-link >$, in which the item name is the item name used for identification, the count is the number of transactions that reach this node by the same path from the root, and the node-link is a pointer linking to the next node in the FP tree with the same item name. To insert a transaction, P, into the FP tree, T, we check whether T has a child, n, such that n item name is identical to the item name of the first element of P. If the node exists, then the count of n is increased by 1. Otherwise, it creates a new node, m, with the same item name as n. Meanwhile, the count of m is set to 1, the parent link is set to T, and its node-link is set to the nodes with the same item name via the node-link structure. We recursively perform the insertion for each item in P until each item is inserted into the FP tree. After the FP tree is constructed, FP growth is used to discover the FPs. An item of the header table is selected to construct the conditional FP tree by inserting all the prefix paths of the item, which can be retrieved by the node-link structure in the header table. The name of the item is called the conditional pattern base. Then, the FP growth is executed recursively and the conditional pattern base is cascaded by a new one in each recursion until the conditional FP tree contains only a single path or is an empty tree. The FPs can be easily generated by the cascaded conditional pattern base and the FP tree. After each item in the header table is processed, all the FPs are obtained.

An efficient and scalable association rule mining algorithm is important as it can be the kernel of many advanced algorithms. For example, the study [15] considered the rule correlation among various tasks to discover interesting information and proposed a method, MTARM (multitask association rule miner), for mining such rules. The proposed algorithm, MTARM, then chose an association rule mining algorithm as based miners for completing the mining named MT-Apriori, MT-FP-Growth, and MT-Eclat, which stands for integration of Apriori, FP growth, and Eclat, respectively.

1) DATABASE PROJECTION ALGORITHM

Han *et al.* [2] proposed database projection to address the issue of insufficient memory when mining in big data, because mining in a big database would make the FP tree unable to finish construction. Database projection utilizes the disk to save mining information on the basis of FP growth. When the memory is insufficient, the algorithm would start to condense the database for the subsequent stage of FP growth. The process of database condensing would be maintained until the memory is sufficient.

Algorithm 1 Database Projection

Input: Database D and Memory M.
Output: The complete set of frequent patterns.
Procedure Database Projection (D, M) {
  $T =$ bulid_FP-tree(D);
  IF $T \leq M$ then {
    return FP-growth (T);
  } Else {
    get frequent items $\boldsymbol{i_1}, \boldsymbol{i_2}, \ldots, \boldsymbol{i_n}$ of D;
    decompose D into $\boldsymbol{D_{i_1}}, \boldsymbol{D_{i2}}, \ldots, \boldsymbol{D_{in}}$;
    return Database Projection ($\boldsymbol{D_{i1}}$, M) $\bigcup$
      Database Projection ($\boldsymbol{D_{i2}}$, M) $\bigcup$
      ...
      Database Projection ($\boldsymbol{D_{i_n}}$, M)
  }
}

B. DISTRIBUTED ALGORITHMS FOR DISCOVERY OF FREQUENT PATTERNS

Several algorithms based on distributed or parallel computing have been proposed to improve execution performance. For example, PFP tree [16] utilizes a multiprocessor system. However, the execution time increases due to the tree structure and high cost of data transmission. The performance deteriorates notably when the database size increases or the given support decreases. QFP (QFP-growth) [17] applies the multiprocessor system as well, but it has limited performance in FP tree construction by each core processor. If multiple transactions share an identical frequent itemset, then they could be merged with the number of occurrences, which is registered as count. The QFP method does not have a good computing ability from other available processors, thereby resulting in a highly redundant execution time. TPFP-tree (TIDset-based parallel FP-tree) [18] is a proposed parallel-distributed mining algorithm based on FP tree. To exchange transactions efficiently, a transaction identification set (TIDset) was used to directly choose transactions without scanning databases. However, the method would take up a large amount of memory space, especially when mining big data, which would easily result in insufficient memory.

A grid system is a heterogeneous computing environment, where the processor's capability and the memory space are different. Equal partition will increase the make span and cause some computing nodes to become idle. Hence, the BTP-tree (balanced TIDset parallel FP-tree) [19] was proposed to reduce the communication and tree insertion cost for

decreasing the execution time. The BTP tree partitions the target mining items according to the performance index (PI) of each processor to balance the workload, similar to the header table. Each processor processes a small set of transactions by using the FP tree creation process to evaluate the PI. However, the execution time for data transmission increases for each processor when the nodes request the required data from each other to finish the mining task. The task assignment is decided by the PI mechanism. The required different transaction loading also increases the execution time, thereby wasting some of the preprocessed time for PI. Moreover, the execution time and the required memory increase considerably as the database size increases because of the transaction selection mechanism, namely, TIDset. FD-Mine [20] utilizes the nodes to discover FPs in cloud computing environments. The data structure is based on FP growth, which stores the frequent items in a compressed form to reduce the transmission time. To make the computing loading balance better than BTP tree, batch-run FP growth for each conditional item in the subset was used to mine the FPs. However, this strategy would also cause the data transmission cost to increase with the growth of the tree. Also, during the mining process, the unadaptable nodes would sometimes decrease the execution efficiency.

Apache Spark inherited Hadoop and was proposed in 2010, then it replaced Hadoop MapReduce in 2014. S-FPG [13] was first proposed using the in-memory parallel computing framework by using the scalable parallel P-growth implementation. The method improved the efficiency of processing large datasets. Then, caching-based parallel FP growth was proposed [21], consisting of integer-based sorting and resilient distributed datasets (RDD)-caching strategy to improve the efficiency. The proposed CPFPGrowth-MD (caching-based parallel FP-Growth with memory-and-disk) and CPFPGrowth-D (caching-based parallel FP-Growth with disk) decreased the execution times more than the original PFP growth did and performed better when executing on data centers without enough memory. The communication cost causes the problem of reduced efficiency. A parallel FP-growth algorithm called IFPS [22] based on Spark was improved by matrix technology. In this algorithm, a dataset was compressed into an information matrix to reduce memory consumption. However, IFPS also has problems when the amount of data is large, such as insufficient memory and high time consumption. PS-MP-FP-Growth (Spark platform merging pruning FP-Growth) [23] designed parallel Apriori and FP-growth algorithm by using the Spark platform. The algorithm was developed for the optimization and improvement of the balanced grouping strategy and the non-frequent item merging pruning strategy. However, the problem of insufficient memory still exists.

Apache Spark utilizes RAM without being tied to the two-stage paradigm of Hadoop and is thus potentially 100 times faster than Hadoop MapReduce [24]. However, this feature comes with the limitation of computing cost. Moreover, Apache Spark works well for smaller datasets only. When the strategy is more prefer to the cost-effective processing massive data set. Then, the critical problem would revert to Hadoop MapReduce and related algorithms.

The PFP [6] algorithm, based on the Hadoop framework, proposed a MapReduce approach of parallel FP- growth algorithm, the shared DB was preprocessed through the generated G-list. FP trees were constructed by a G-list by distributing TIDs to the related groups. Once the tree has been constructed, the subsequent pattern mining could be performed. However, the mapper needs to communicate the collected corresponding group-dependent transactions before mining, thereby resulting in a high execution cost. According to the advantages of PFP, several kinds of algorithm have been proposed. For example, PIFP growth [7] successfully solves the incremental issue caused by the dynamic threshold value and database, thereby avoiding repeated computation. R-PFP [8] extends the level of parallelization that can be achieved by PFP using the parent–child MapReduce feature of the IBM Platform Symphony [25]. However, the bottleneck on the communication cost remains when exchanging data. The same drawbacks are found with MR-PFP [9]. To address the issue of the communication cost, PBFP growth [10] was combined with the Apriori and FP-growth algorithms to reduce the scanning frequency. For MISFP growth [11], the concept of classification of item was proposed, which involves classifying items of higher homogeneity to reduce the execution time on parallel architectures. However, the execution time improvement is limited, especially when mining on large databases.

Mining algorithms based on the Hadoop system have the least efficient execution time. The Hadoop system has limited efficiency because the subsequent mining result is reported to Hadoop distributed file system (HDFS) for every single phase, despite some results were the input for the next turn could cause extra data mining time. BigFIM [12] proposed by S. Moens was implicated the associating mining based on the Hadoop system. The algorithm has two dedicated mechanisms: DistEclat and BigFIM. DistEclat focuses on efficiency and ensures that each mapper finishes mining tasks with the adaptive parameter p to balance the computing loading on a distributed system. BigFIM focuses on big database and first uses the Apriori algorithm for mining until the frequent itemsets of size p are achieved. From the increment parameter p, the BigFIM mechanism would switch to DistEclat to improve the insufficient memory caused by TID-List. The performance improved because BigFIM would mining the frequent item first before p-FIs to reduce the size of the TID-List through the Apriori algorithm. The two mechanisms exhibited better performance than FP growth utilizing PFP with the Hadoop system. However, the parameter p needs to be set first, which would cause difficulty for beginners.

The conventional association rule mining can be further divided into two subtopics, positive association rule mining and negative association rule mining. The positive association rule mining discovers sets of items, i.e., frequent patterns, associated with other items in the database, whereas the negative association rule mining is used to discover the relation

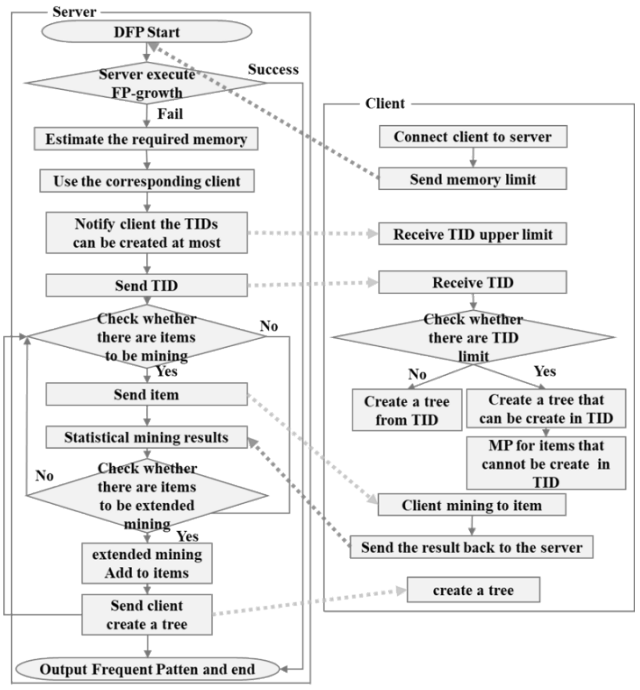

FIGURE 1. The flow chart of the proposed DFP algorithm.

between item-sets in which support values are negatively correlated. Because the data size and complexity increase rapidly, an efficient method is required for mining such rules. Most of the past studies focused on positive association rule mining; therefore, Bagui and Dhar [26] proposed a method to efficiently discover both positive and negative association rules, simultaneously, from big data, using Hadoop's MapReduce architecture.

III. PROPOSED METHOD

Section III-A introduces the problem and Section III-B details the proposed method and presents the features of the DFP algorithm.

A. PROBLEM DEFINITION

The FP-growth algorithm is not efficient when processing a big database. Hence, numerous algorithms related to distributed or parallel computing have emerged. However, these methods have high communication cost, insufficient memory, low computing efficiency, and redundant computing waste. Therefore, DFP was proposed based on FP growth. If the first stage failed during mining, then the proposed method would estimate the necessary memory and nodes for further distributed computing steps. The excellent performance of DFP is presented in the Experiment Results section. If the memory of the nodes is less than the estimated necessary memory, then the extended database projection method called mixing projection (MP) will be activated. MP is used to store compressed crucial information and develop an efficient FP tree-based mining method. MP performs better than the

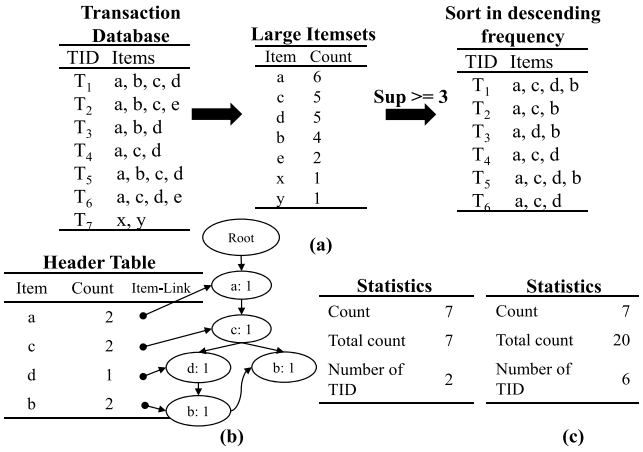

FIGURE 2. An example of memory space estimation.

DP algorithm especially when the process is unable to mine any further. The process would compress a large database into a condensed, smaller data structure repeatedly until the dataset can fit in the main memory. However, the process also increases the input/output (I/O) cost. With the proposed MP algorithm and the main memory space estimation method, the repeated procedure for estimating memory space availability becomes unnecessary. If the database needs to be condensed, then specific items instead of all items would be condensed by the proposed method. With the use of the extended MP method, the memory capacity is evaluated for the database without repeated checking, thereby improving the transmission cost between the server and the nodes.

B. DFP MINING ALGORITHM

In this section, we present an efficient DFP algorithm based on the FP tree data structure. The algorithm steps include the procedure for the nodes of the server and the client.

1) SERVER SIDE

The DFP algorithm has four stages in the server: a) memory space evaluation, b) client node activation, c) TID distribution, and d) waiting for mining item distribution.

a: MEMORY SPACE EVALUATION

As in Fig. 1, in the first stage, the server would execute the FP-growth algorithm first for two purposes: to avoid unnecessary transmission time cost by using the distribution mode instead of mining in its own server and to record essential information, including count, total counts, number of TIDs, and estimating the necessary memory space in the database when the mining procedure has failed. In the algorithm, full garbage collection (GC) would be used to determine if the mining has failed. Before the TID tree is constructed, the count of full GC is counted.

If the full GC is equal to 1, then the mining has failed. Then, the remaining TIDs would be recorded as essential information. For example, in Fig. 2(a), the transaction database

Server Information

Server memory	1000M
Count	7
Total count	20
Number of TID	6

(a)

Client Information

Client ID	Memory
Client_1	1500M
Client_2	1000M
Client_3	500M

Client TID processing limitation

Client ID	TID upper limit
Client_1	3
Client_2	2
Client_3	1

(b)

Client TID received

Client ID	TID received
Client_1	T_1 、T_4 、T_6
Client_2	T_2 、T_5
Client_3	T_3

(c)

FIGURE 3. An example for distributing TIDs to the assigned nodes.

Item b mining results

Client_1	a: 1 、c: 1 、d: 1
Client_2	a: 2 、c: 2 、d: 1
Client_3	a: 1 、d: 1

(a)

Statistics

Item	Count
a, b	4
c, b	3
d, b	3

(b)

Item to be mining

a
c
d
a, b
c, b
d, b

(c)

FIGURE 4. An example of waiting mining items distribution.

would retrieve a support value greater than 3, then the $I - FIs\{a : 6, c : 5, d : 5, b : 4\}$ would be obtained in accordance with the filter and order these TIDs for constructing the tree. The algorithm would check the count of full GC. In our example, as shown in Fig. 2(b), the constructed T_1 and T_2 include 7 items, the count and the total are equal to 7, and $TID = 2$. Assuming that the full GC is equal to 1 before the TID of T_3 is scanned, the method would scan the remaining TIDs and stop the counting, as shown in Fig. 2(c). Then, the evaluation of the cost of the main memory space for finishing the database mining would be followed by the formula $m \times (Counts/(\sum_{i=1}^{n} T_i))$, where m is the memory space and $|T_i|$ is the length of the transaction. If the main memory in the server is 1000M, then the main cost needs at least $1000 \times 20/7 = 2857$M for the assigned database mining.

b: CLIENT NODES ACTIVATION

The necessary nodes must be estimated prior to distributed mining. Our experiment results indicate that an increase in nodes would not increase the efficiency. The proposed algorithm would take the limited nodes to achieve the best performance. The following are examples for three different conditions in clients. The mechanism would optimize the best choice. If 2857M is needed to connect to 10 clients, with 1000M for each client, then 3 nodes would be activated. If the set for the three clients is 1000M, 1500M, and 2000M, respectively, then the activated clients would be ordered for 2000M and 1500M. Even if there are only two activated clients for 1000M, the proposed DFP algorithm would drive the MP mechanism and proceed with the efficient mining process.

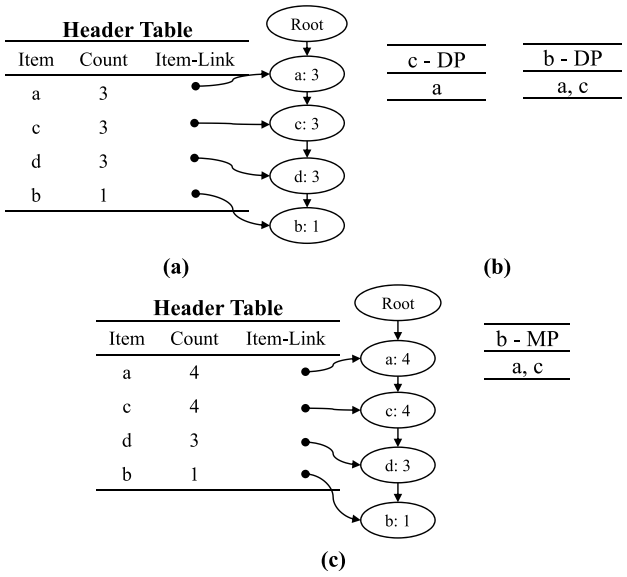

FIGURE 5. An example of tree construction.

c: TID DISTRIBUTION

In this procedure, the server evaluates the activated nodes with regard to the processing limitation for the TIDs and waits for the activated MP mechanism to condense the database. For example, if the server has a 1000M memory space, then the memories for nodes are 500M, 1000M, and 1500M, respectively. The procedure is shown as follows:

$$avgT = \frac{TIDs}{number\ of\ nodes} \tag{1}$$

$$the\ client\ limitation\ for\ TID = \frac{memory\ space\ in\ nodes}{memory\ space\ in\ server} \times avgT \tag{2}$$

For the example shown in Fig. 3(a), if the memory for *client_1* is 1500M and the memory for server is 1000M, then the TID processing limitation for client is 3 (the second formula was used, and the result is shown in Fig. 3(b)). The results are the basis for the process to start distributing TIDs to the assigned nodes (shown in Fig. 3(c)). Thus, 3, 2, and 1 TIDs would be distributed to *client_1*, *client_2*, and *client_3*, respectively. As shown in Fig. 3(c), the experiment result for the example did not exceed the limitation of TIDs. Thus, the extended procedure for the MP mechanism would not be executed.

d: WAITING MINING ITEMS DISTRIBUTION

Once the server is activated, the FP-growth algorithm is performed and result in the frequent itemset of *1-FIs* as the waiting mining items. As shown in Fig. 4, the server would distribute items of b to the client for further mining and wait for the results (shown in Section III-B-2-c). The server then starts to mine the frequent itemset from the r result shown in Fig. 4(a) and then outputs the result and the transactions, as shown in Fig. 4(b). The results for *2-FIs* re $\{a, b : 4; c, b : 3; d, b : 3\}$. The *2-FIs* have further mining potential,

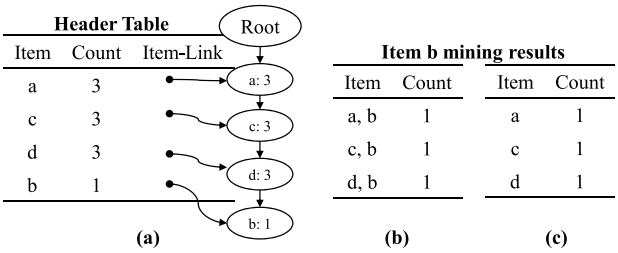

FIGURE 6. An example for mining the frequent items.

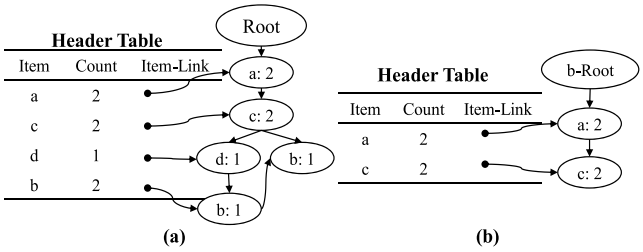

FIGURE 7. An example for constructing Conditional FP-tree.

and the server would further combine these as *3-FIs*, index them as potential mining items (Fig. 4[c]), and prompt the client to construct a conditional FP tree for the frequent items of b. In contrast, if the response result from the client is null, then the algorithm would not proceed with tree construction. The procedure would be repeated until all waiting items are finished.

2) CLIENT SIDE

This DFP algorithm has five stages in the client: a) memory space reporting to server, b) receiving the limitation for TIDs, c) tree construction according to TIDs, d) mining frequent items, and e) construction of the condition FP tree.

a: MEMORY SPACE REPORTING TO SERVER

The memory space varies with every client. To estimate the distributing TIDs for transfer to the client from the server, each client would transfer the message to the server once they connect to the server.

b: RECEIVING THE LIMITATION FOR TIDS

As mentioned, the server would estimate the limitation for TIDs according to the estimated memory from the client. If the number of TIDs exceeds the limitation, then the MP mechanism would be activated to execute the remaining overflowed TIDs to avoid the out-of-memory (OOM) condition in the client.

c: TREE CONSTRUCTION ACCORDING TO TIDS

If all clients reached their limit, then the server would transfer the remaining TIDs to each client. If the client received the $TID - \{a, b, c\}$ (Fig. 5[a]), then the algorithm would start to condense the database to avoid OOM. On the basis of the DP algorithm, the condensed database is shown in Fig. 5(b). Hence, if the client wants to perform mining, the item of c or b should be obtained from the root and the file of DP should be read from the disk. The MP mechanism could reduce the I/O cost through the root tree in the client memory by incrementing the count according to nodes. Then, the remaining items would be executed by the MP mechanism. For example, in Fig. 5(c), the $TID - \{a, c, b\}$, was received from the server, and the path is $Root - > a - > c$. Then, a and c would increment the count, and b would execute the MP mechanism.

d: MINING FREQUENT ITEMS

The mining process would start from the message to expect mining items from the server, because the received TIDs would differ from client to client. For example, a constructed tree of the client is shown in Fig. 6(a), and the item of b is the expecting mining item assigned from server. On the basis of FP growth, the item of b would be mined. If there were files of $b - MP$, then mining should be completed, as shown in Fig. 6(b). In our proposed algorithm, the mining result would be extracted as shown in Fig. 6(c). Obviously, the mining task assigned from the server would not need the redundant message-record item of b (as shown in Fig. 6[b]), which would also reduce the transmission cost. Finally, the table would be transferred to the server.

e: CONSTRUCTION OF THE CONDITION FP TREE

As mentioned, if the mechanism received the result with the potential extended mining items of Fig. 6(c), then the server would start to connect to the client to construct a conditional FP tree and extract the frequent itemset to prevent the client from adding information that does not belong to frequent items into the conditional FP tree. For example, if the client would like to build the conditional FP tree for item b and the server would index $\{a, c\}$ as a frequent item, then the client would extract the item d, as shown in Fig. 7(b). In the meantime, the file of b-MP should be read for further construction.

IV. EXPERIMENTAL EVALUATION AND PERFORMANCE STUDY

In the section, we present a performance comparison of DFP with the classical FP mining algorithm.

Quest Synthetic Data Generator [27] from IBM was utilized to generate data with different parameters. Real data were downloaded from the frequent itemset mining dataset (FIMD) repository [28] to evaluate the performance of the proposed method and compare the experiment results of the proposed method with those of DistEclat and BigFIM. Then, the modified mechanisms of MP and DP in the DFP algorithm were compared. Finally, the experiments with insufficient main memory in the client and different memory spaces in clients were compared in further sections. The experiment environment, datasets, and parameters are introduced in Section IV-A. Experiment results with different parameters are analyzed in Section IV-B. The experiments

Algorithm 2 DFP Server

Input: Database DB and a Support ξ.
Output: The complete set of frequent patterns. FP

```
result ← FP-growth(DB, ξ)
IF(result.FP ≠ ∅){
    RETURN result.FP // FP-growth succeed
} END {
    TIDCount ← result.getTIDCount()
    requiredMemory ← result.getEstimateMemory()
    C ← selectClient(requiredMemory)
   FOR i = 1 TO C.count{
        t ← calculateClinetMaxTID(C_i, TIDCount)
        tellClientMaxTID(C_i, t)
    }
    FOR i = 1 TO TIDCount{
        sendTID(C_{i mod C.Count}, DB_i)
    }
    pendingMining ← result.1-FIs
    WHILE(pendingMining ≠ ∅){
        miningTask ← pendingMining.removeLast()
        sendMiningTask (C, miningTask)
        miningResult ← allClientMiningRusult()
        result.FP ← result.FP ∪ miningResult.FP
        IF(miningResult.pendingMining ≠ ∅){
            buildSubTree(C, miningResult.subTreeInfo)
            pendingMining ← pendingMining ∪
            miningResult.pendingMining
        }
    }
    RETURN result.FP
}
```

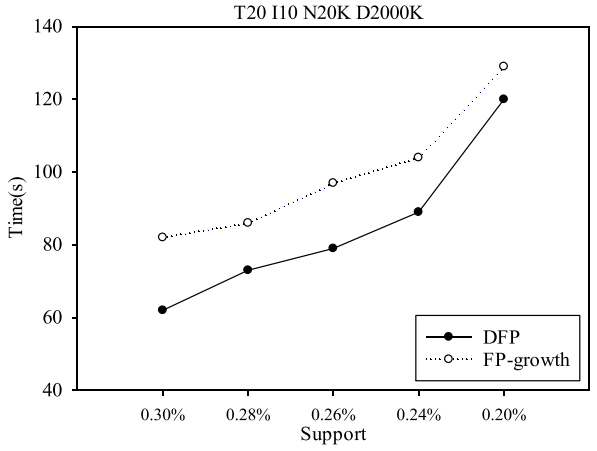

FIGURE 8. Execution performance of DFP and FP-growth with support threshold varied from 0.3 to 0.2% on IBM dataset.

Algorithm 3 DFP Client

Input: NULL
Output: NULL

```
S ← connectToServer()
sendMemory(S, getMemory() )
WHILE(TRUE){
    S.command ← waitingForServerSendCommand()
    IF S.command = MaxTID {
        MaxTID ← S.getMaxTID()
    }
    IF S.command = TID {
        TID ← S.TID
        IF BuildTID < MaxTID {
            BuildTID(TID)
        } ELSE{
            BuildTIDAndDoMP(TID)
        }
    }
    IF S.command = MiningTask {
        miningResult ← FPGrowth(S.miningTask)
        sendMiningResult(S, miningResult)
    }
    IF S.command = BuildSubTree {
        BuildSubTree(S.subTreeInfo)
    }
}
```

TABLE 1. An example for constructing Conditional FP-tree.

parameter	Statement
T	Average transaction length
I	Average frequent pattern length
N	Number of items
D	Number of transactions
Support	The minimal support threshold

TABLE 2. Execution time with various support thresholds (0.3–0.2%) for DFP and FP-growth on IBM dataset.

T20 I10 N20K D2000K				
Support(%)	Estimated memory	DFP clients	DFP	FP-growth [execution time]
		Time(s)		
0.30%	1852M	2	62	82
0.28%	2179M	3	73	86
0.26%	2531M	3	79	97
0.24%	2985M	3	89	104
0.20%	4073M	5	120	129

[Note] The execution time for FP-growth: the execution time for the OOM with 1G plus the time for finishing the mining process.

are summarized at the end of this section (IV-C). These analyses are well supported by the experiments reported in this section.

A. EXPERIMENTAL SETUP

1) ENVIRONMENTS OF EXPERIMENTS

All the experiments are performed on an Intel(R) Core^{TM} i7-4770 CPU@3.40 GHz personal computer (PC) machine with 8 GB main memory and 500 GB of storage, running on Microsoft Windows 7 Enterprise Edition operation system. All the programs are written in Java. We utilized Ubuntu (14.04.4 version) with 3 Gb main memory through Virtual Box to compare the experiment result with BigFIM. The memory limitation was set to 1 GB when the experiment was in the process of mining.

2) EXPERIMENTAL DATA SETS AND PARAMETERS

Part of the experimental dataset was generated by Quest Synthetic Data Generator of IBM [27], and the data are based on general transaction record databases, including the average

TABLE 3. Execution time with various settings of T(35-25), I(16-7), N(15-5K), and D(3500-1500K) for DFP and FP-growth on IBM dataset.

T__ I10 N20K D2000K				
T__	Estimated memory	DFP Clients	DFP	FP-growth [execution time]
		Time(s)		
25	4383M	5	117	177
30	7360M	8	216	Insufficient memory
35	9945M	10	488	Insufficient memory
T20 I__ N20K D2000K				
I__	Estimated memory	DFP Clients	DFP	FP-growth [execution time]
		Time(s)		
7	2388M	3	76	113
13	1397M	2	58	72
16	1123M	2	47	61
T20 I10 N__K D2000K				
N__K	Estimated memory	DFP Clients	DFP	FP-growth [execution time]
		Time(s)		
5	6712M	7	138	Insufficient time
10	4727M	5	104	183
15	2990M	3	80	88
T20 I10 N20K D____K				
D____K	Estimated memory	DFP Clients	DFP	FP-growth [execution time]
		Time(s)		
1500	1364M	2	46	62
2500	2176M	3	82	100
3000	2625M	3	95	124
3500	3186M	4	101	152

[Note] insufficient memory: some of the parameters were unable to be executed, since the available memory is 6000M under capability for the Ubuntu experimental environment.

transaction length(T), average frequent itemset length(I), number of items(N), and number of transactions(D).

In our experiment, the memory was limited to 1 GB for the mining process, and T20 I10 N20K D2000K were set as our basic experiment datasets. First, in the DFP algorithm, we utilized these basic experiment dataset to find the support threshold that the FP growth was unable to finish the mining in the first stage. To compare the performance evaluated by the proposed method and DistEclat and BigFIM, the real data that were generated from the FIMD Repository was utilized for the experiments. The proposed DFP algorithm could mine with high efficiency through the dataset generated from our daily life.

B. EXPERIMENTAL RESULTS

1) EFFECT OF VARYING THE SUPPORT AND PARAMETERS T, I, N, AND D OF DFP AND FP-GROWTH FOR A IBM DATASET ON EXECUTION TIME

If the memory space estimated by the DFP algorithm was assigned for continuing the mining process when the mining process was failed by the FP-growth algorithm with 1 GB RAM. The results in Table 2 show that the mechanism is feasible without including the OOM issue. Fig. 8 shows that the proposed method outperforms the FP-growth algorithm

TABLE 4. Execution time with various support thresholds (0.6%–0.2%) for DFP, DistEclat, and BigFIM on IBM dataset.

T20 I10 N20K D2000K					
Support(%)	Estimated memory	Clients	DFP	DistEclat	BigFIM
			Time(s)		
0.60%	-	-	19	-	-
0.50%	-	-	22	-	-
0.40%	-	-	29	-	-
0.30%	1852M	2	62	334	159
0.28%	2179M	3	73	299	302
0.26%	2531M	3	79	323	334
0.24%	2985M	3	89	358	366
0.20%	4073M	5	120	344	OOM

[Note] -: FP-growth mechanism were finished the mining.
[Note] OOM: The mining process was failed due to the insufficient memory.

because the mechanism of the FP-growth algorithm does not estimate the requirement for memory space, thereby causing OOM. The performance was worse than that of DFP because the FP tree needs to spend more time scanning and exploring to construct the FP tree on a single computer. In contrast, the DFP algorithm distributes the databases with optimized transferred datasets to clients. Although the process increased the data transmission cost, the performance of DFP is highly efficient, as shown in Fig. 8.

The experiment result with different parameters T, I, N, and D, and a fixed support value of 0.3% was utilized for further examination, as shown in Table 3 and Fig. 9. The memory is insufficient with the parameter T set to 30 and 35, and with N set to 5 because in the Ubuntu experiment environment, only 6000M RAM could be accessed. However, the rest of the experiment parameters showed that the memory space is correct and feasible, as estimated by the proposed DFP algorithm. Moreover, the proposed DFP algorithm outperforms the FP growth algorithm.

2) EFFECT OF VARYING THE SUPPORT THRESHOLD ON EXECUTION TIME AND AMOUNT OF TRANSMITTED DATA FOR DFP, DISTECLAT, AND BIGFIM WITH DIFFERENT NUMBER OF CLIENTS

The experiment used the basic experiment dataset T20 I10 N20K D2000K to obtain the support threshold of 0.2%–0.6%. The threshold means that the first stage of FP growth would not fail in the mining process in the server. Then, the execution time and numbers of transactions were utilized to compare the performances of DFP, DistEclat, and BigFIM.

Table 4 and Fig. 10 show that the FP growth failed when the support threshold was 0.3%. Then, the mechanism was activated to estimate the memory for the distributed mining process. The experiment showed that the proposed method outperformed DistEclat and BigFIM. The number of transactions for the proposed method was also better than that of DistEclat and BigFIM (Table 5 and Fig. 11). Overall, the performance of DFP is better than that of DistEclat and BigFIM by about 4 times, and the number of transactions is lower

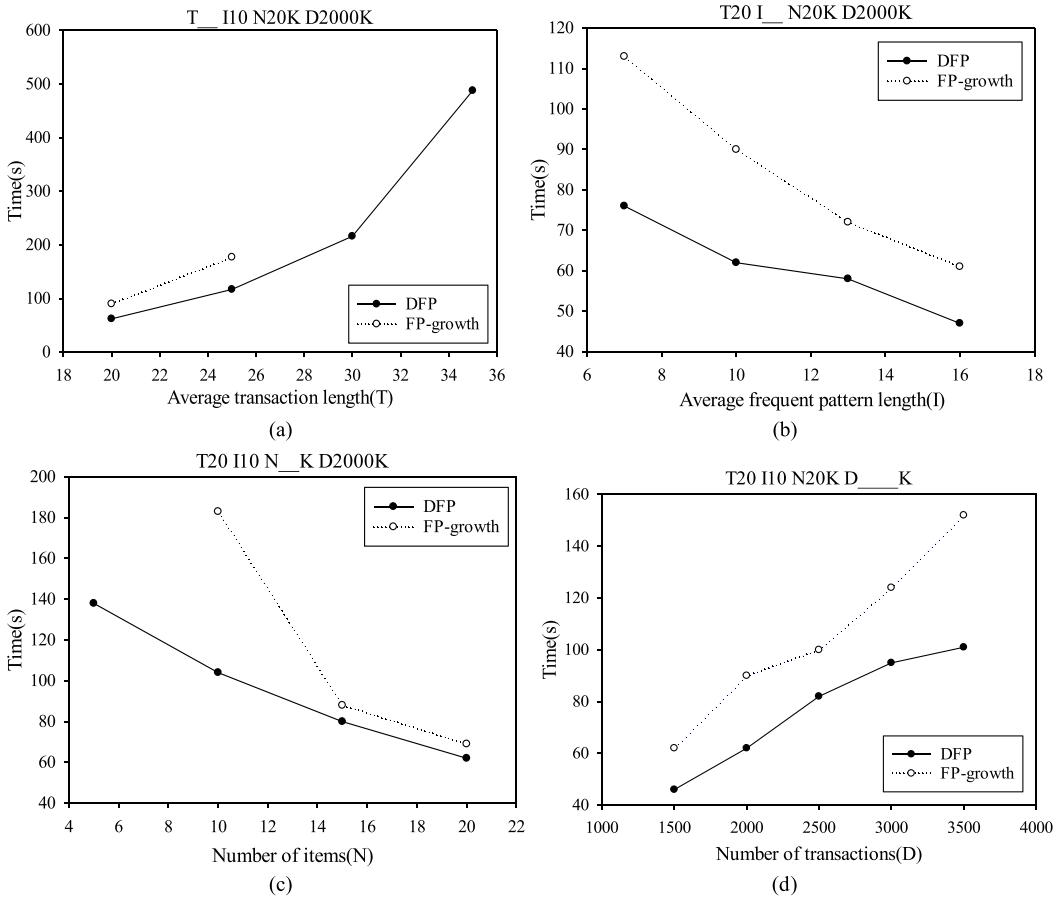

FIGURE 9. Execution performance of DFP and FP-growth with T(35-25), I(16-7), N(15-5K), and D(3500-1500K) varied on IBM dataset.

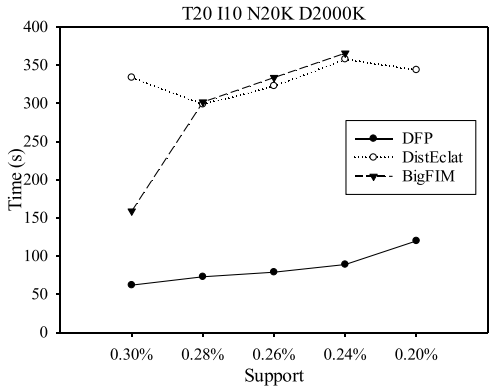

FIGURE 10. Execution performance of DFP, DistEclat, and BigFIM with support threshold varied from 0.3% to 0.2% on IBM dataset.

TABLE 5. Amount of transmitted data with various support thresholds (0.3%–0.2%) for DFP, DistEclat, and BigFIM on IBM dataset.

T20 I10 N20K D2000K				
Support(%)	Clients	DFP	DistEclat	BigFIM
		Transmitted data(K)		
0.30%	2	74,391	413,910	522,819
0.28%	3	98,270	419,927	1,045,015
0.26%	3	116,557	429,701	1,055,981
0.24%	3	141,973	436,960	1,063,841
0.20%	5	274,637	454,440	OOM

[Note] OOM: The mining process was failed due to the insufficient memory.

than that of DistEclat and BigFIM by about 4 and 9 times, respectively.

The numbers of candidates for the next stage overflowed and caused insufficient memory, thereby causing BigFIM to fail in the experiment when the support threshold was set to 0.2%. When the support threshold set was to 0.3%, DistEclat performed worse than BigFIM did because the mined frequent items did not satisfy the parameter p, also known as *3-FIs*. The finished preprocessing mining in the DistEclat mechanism to balance the computing loading was the cause of lack of efficiency.

When 10 clients were included in the DFP algorithm, DistEclat, and BigFIM, the execution time exhibited a decreasing trend because extra waiting time was spent waiting for communication and responding to clients (Table 6 and Fig. 12). In BigFIM, the sources of reduced files were increased, thereby also causing the decreasing trend in the stage of reduce. However, the DistEclat with same 10-client

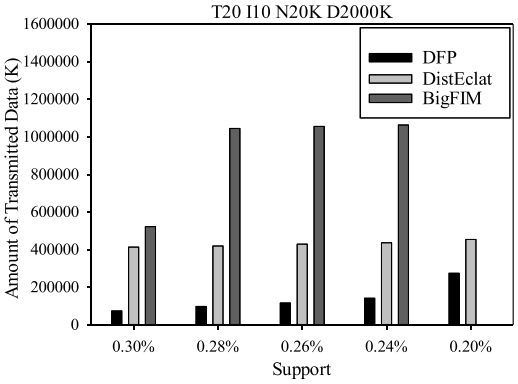

FIGURE 11. Amount of transmitted data of DFP, DistEclat, and BigFIM with support threshold varied from 0.3% to 0.2% on IBM dataset.

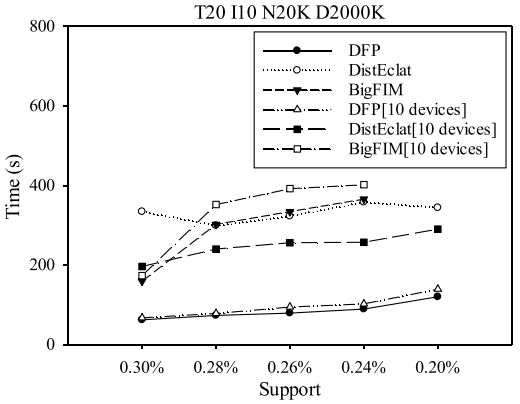

FIGURE 12. Execution performance of DFP, DistEclat, and BigFIM with support threshold varied from 0.3% to 0.2% on IBM dataset using estimated and 10 clients.

TABLE 6. Execution time with various support thresholds (0.3%–0.2%) for DFP, DistEclat, and BigFIM on IBM dataset using estimated and 10 clients.

T20 I10 N20K D2000K					
Support(%)	Estimated memory	Clients	DFP	DistEclat	BigFIM
			Time(s)		
0.30%	1852M	2	62	334	159
0.28%	2179M	3	73	299	302
0.26%	2531M	3	79	323	334
0.24%	2985M	3	89	358	366
0.20%	4073M	5	120	344	OOM
T20 I10 N20K D2000K [10 Clients]					
Support(%)	Estimated memory	Clients	DFP	DistEclat	BigFIM
			Time(s)		
0.30%	1852M	2	67	196	173
0.28%	2179M	3	78	240	352
0.26%	2531M	3	94	256	392
0.24%	2985M	3	102	257	402
0.20%	4073M	5	139	290	OOM

[Note] OOM: The mining process was failed due to the insufficient memory.

TABLE 7. Amount of transmitted data with various support thresholds (0.3%–0.2%) for DFP, DistEclat, and BigFIM on IBM dataset using estimated and 10 clients.

T20 I10 N20K D2000K				
Support(%)	Clients	DFP	DistEclat	BigFIM
		Transmitted data(K)		
0.30%	2	74,391	413,910	522,819
0.28%	3	98,270	419,927	1,045,015
0.26%	3	116,557	429,701	1,055,981
0.24%	3	141,973	436,960	1,063,841
0.20%	5	274,637	454,440	OOM
T20 I10 N20K D2000K [10 Clients]				
Support(%)	Clients	DFP	DistEclat	BigFIM
		Transmitted data(K)		
0.30%	2	130,650	414,116	544,109
0.28%	3	160,752	420,063	1,071,161
0.26%	3	196,436	429,903	1,091,490
0.24%	3	245,520	437,164	1,105,524
0.20%	5	375,699	454,641	OOM

[Note] OOM: The mining process was failed due to the insufficient memory.

conditions exhibited increased performance because DistEclat stores TID-Lists in the memory and mines the items in *I-FIs* to *p-FIs*. The increased numbers of mappers caused the decreased *1-FIs* items in each mapper. Overall, DFP still performed better in terms of efficiency and data transmission than DistEclat and BigFIM (Table 7 and Fig. 13). The experiment results also indicate that having more clients would not increase the efficiency.

Table 8 and Fig. 14 show that when the number of clients was set to 15 for DistEclat, the performance did not improve than when the number of clients increased to 15. The experiment on 15 clients with a support threshold of 0.2% showed that the performance of DFP decreased dramatically, becoming even worse than that of DistEclat. The execution time cost for DistEclat with 10 or 15 clients was still worse than that estimated for DFP. Table 9 and Fig. 15 show that the proposed method with various parameters still performed better than DistEclat and BigFIM. In other words, the DFP can utilize minimal clients but still has the best efficiency and data transmission cost.

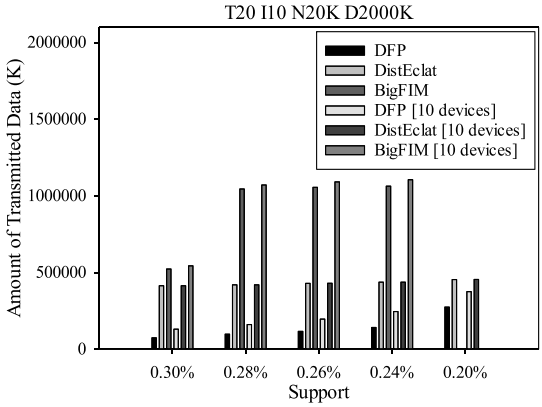

FIGURE 13. Amount of transmitted data of DFP, DistEclat, and BigFIM with support threshold varied from 0.3% to 0.2% on IBM dataset using estimated and 10 clients.

3) EFFECT OF VARYING THE SUPPORT OF DFP, DISTECLAT, AND BigFIM FOR A REAL DATASET ON EXECUTION TIME, AND AMOUNT OF TRANSMITTED DATA

The proposed DFP performed highly efficiently with the data generated by IBM's Quest Synthetic Data Generator. We then

TABLE 8. Execution time with various support thresholds (0.3%–0.2%) for DFP, DistEclat, and BigFIM on IBM dataset using estimated, 10, and 15 clients.

T20 I10 N20K D2000K											
Support (%)	Estimated memory	Clients	DFP	DistEclat	BigFIM	DFP	DistEclat	BigFIM	DFP	DistEclat	BigFIM
						[10 Clients]			[15 Clients]		
			Time(s)								
0.30%	1852M	2	62	334	159	67	196	173	98	219	191
0.28%	2179M	3	73	299	302	78	240	352	133	232	366
0.26%	2531M	3	79	323	334	94	256	392	186	258	393
0.24%	2985M	3	89	358	366	102	257	402	230	263	429
0.20%	4073M	5	120	344	OOM	139	290	OOM	380	269	OOM

[Note] OOM: The mining process was failed due to the insufficient memory.

TABLE 9. Amount of transmitted data with various support thresholds (0.3%–0.2%) for DFP, DistEclat, and BigFIM on IBM dataset using estimated, 10, and 15 clients.

T20 I10 N20K D2000K										
Support (%)	Clients	DFP	DistEclat	BigFIM	DFP	DistEclat	BigFIM	DFP	DistEclat	BigFIM
					[10 Clients]			[15 Clients]		
		Transmitted data (K)								
0.30%	2	74,391	413,910	522,819	130,650	414,116	544,109	152,480	414,358	556,467
0.28%	3	98,270	419,927	1,045,015	160,752	420,063	1,071,161	188,011	420,311	1,082,608
0.26%	3	116,557	429,701	1,055,981	196,436	429,903	1,091,490	229,873	430,161	1,102,537
0.24%	3	141,973	436,960	1,063,841	245,520	437,164	1,105,524	286,727	437,432	1,122,342
0.20%	5	274,637	454,440	OOM	375,699	454,641	OOM	434,978	454,930	OOM

[Note] OOM: The mining process was failed due to the insufficient memory.

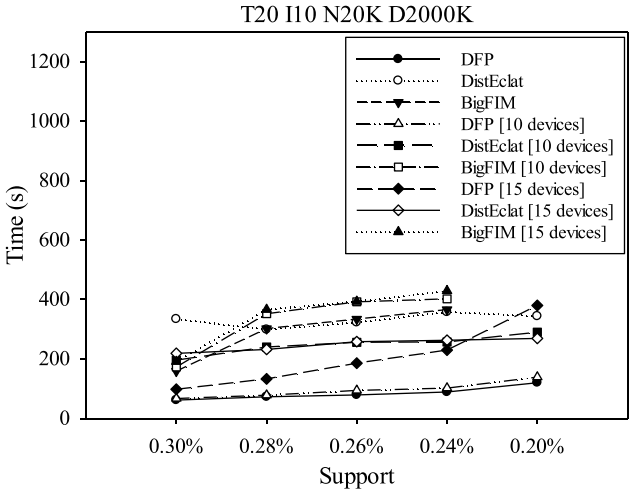

FIGURE 14. Execution performance of DFP, DistEclat, and BigFIM with support threshold varied from 0.3% to 0.2% on IBM dataset using estimated, 10, and 15 clients.

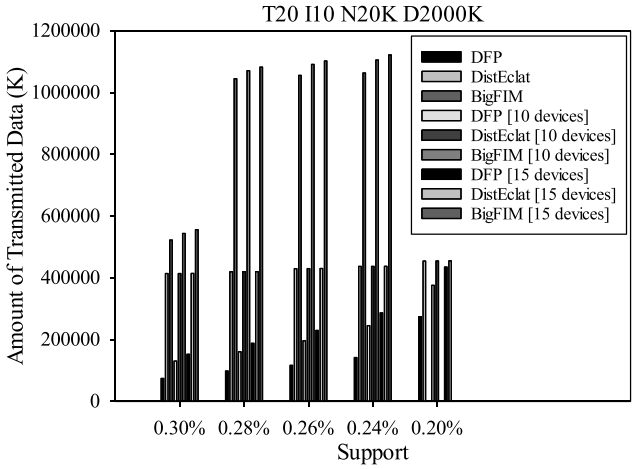

FIGURE 15. Amount of transmitted data of DFP, DistEclat, and BigFIM with support threshold varied from 0.3% to 0.2% on IBM dataset using estimated, 10, and 15 clients.

apply DFP on the real database of Webdocs downloaded from the FIMD Repository to examine the mining performance. The real data are much larger than the data used in previously analyzed experiments with 1.37 GB, 1,692,082 TID, 5,267,656 items, and average TID length of 177.

The experiment result shows that an overloaded TID-List would result in serious OOM during the analysis of a much bigger database. In contrast, the proposed DFP performed at least 6.7 to 8.6 times better than BigFIM (Table 10 and Fig. 16). The experiment value for DFP in terms of data transmission efficiency is much better than that of BigFIM (Table 11 and Fig. 17).

4) EFFECT OF VARYING THE SUPPORT THRESHOLD ON EXECUTION TIME AND AMOUNT OF TRANSMITTED DATA FOR DFP, DISTECLAT, AND BIGFIM WITH VARIOUS MEMORY SPACE CLIENTS

The proposed DFP considered whether efficiency would be affected by different memory space of the client. The experiment finished the examination through the simulation for the assigned clients with different memory space of 500M, 800M, 1000M, 1500M, and 2000M.

The support threshold from 0.2% to 0.3% was utilized for the varying parameters. Table 12 and Fig. 18 show that the DFP still finished the mining even though the client had

TABLE 10. Execution time with various support thresholds (24–16%) for DFP, DistEclat, and BigFIM on real dataset.

Webdocs					
Support(%)	Estimated Memory	Clients	DFP	DistEclat	BigFIM
			Time(s)		
24%	-	-	92	-	-
22%	1741M	2	267	OOM	2298
20%	3157M	4	364	OOM	2567
18%	4783M	5	525	OOM	3901
16%	7063M	8	827	OOM	5505

[Note] -: FP-growth mechanism were finished the mining.
[Note] OOM: The mining process was failed due to the insufficient memory.

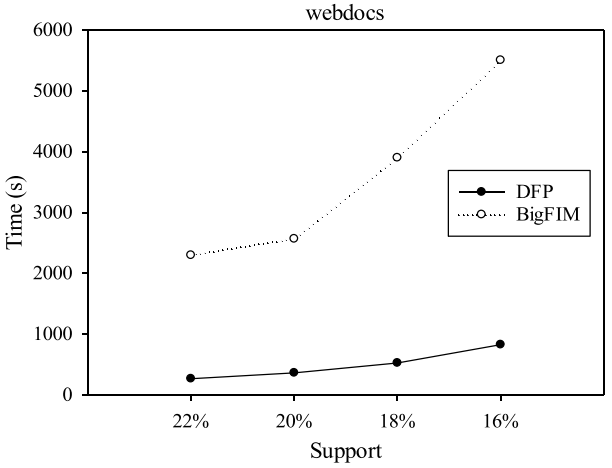

FIGURE 16. Execution performance of DFP and BigFIM with support threshold varied from 22% to 16% on real dataset.

TABLE 11. Amount of transmitted data with various support thresholds (22%–16%) for DFP, DistEclat, and BigFIM on real dataset.

Webdocs				
Support(%)	Clients	DFP	DistEclat	BigFIM
		Transmitted data(K)		
22%	2	110,121	OOM	5,718,142
20%	4	130,401	OOM	8,246,228
18%	5	151,240	OOM	10,180,207
16%	8	186,495	OOM	16,992,236

[Note] OOM: The mining process was failed due to the insufficient memory.

a memory space of 500M. However, BigFIM with a support threshold of 0.2% failed because of the overloading candidate. As for data transmission, Table 13 and Fig. 19 show that the proposed method still performed at least 3 to 5.1 times better than DistEclat and BigFIM.

5) EFFECT OF VARYING THE SUPPORT OF MIXING PROJECTION DFP, DATABASE PROJECTION DFP, DISTECLAT, AND BIGFIM FOR A IBM DATASET ON EXECUTION TIME WITH INSUFFICIENT MEMORY CLIENTS

In this experiment, one condition was considered: If the total estimated memory is less than required, then the transactions would proceed to condense for the extended method—the MP and database projection algorithms by the proposed DFP.

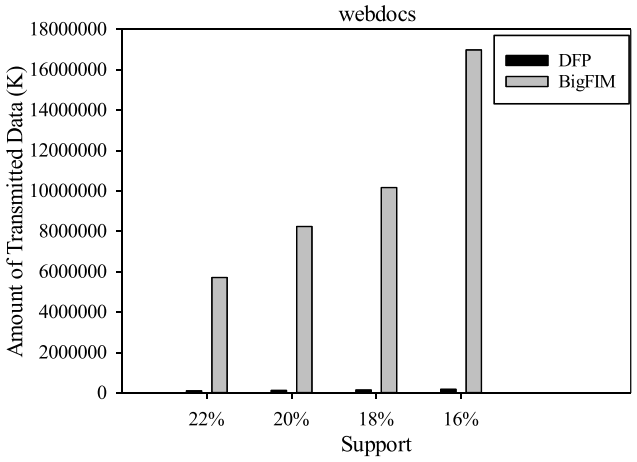

FIGURE 17. Amount of transmitted data of DFP and BigFIM with support threshold varied from 22% to 16% on real dataset.

TABLE 12. Execution time with various support thresholds (0.3–0.2%) for DFP, DistEclat, and BigFIM on IBM dataset using various memory space clients.

T20 I10 N20K D2000K				
Support (%)	Estimated Memory	DFP [5 clients]	DistEclat [5 Clients]	BigFIM [5 Clients]
		Time(s)		
0.30%	1852M	54	261	164
0.28%	2179M	61	255	310
0.26%	2531M	76	270	352
0.24%	2985M	89	293	371
0.20%	4073M	115	356	OOM

[Note] OOM: The mining process was failed due to the insufficient memory.

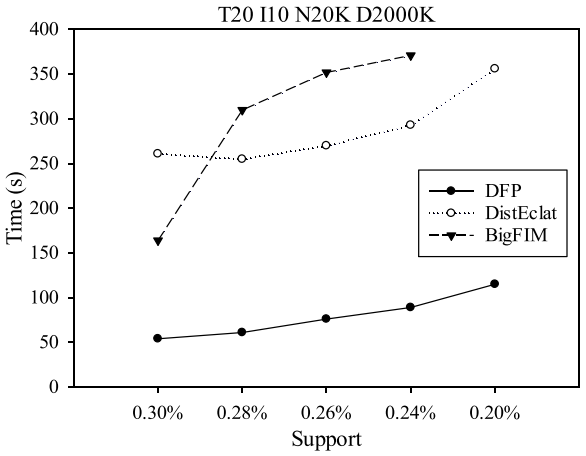

FIGURE 18. Execution performance of DFP and BigFIM with support threshold varied from 0.3 to 0.2% on IBM dataset using various memory space clients.

The support threshold of 0.20% to 0.28% was utilized for further experiments as the necessary memory exceeded the estimated memory of 2000M.

Table 14 and Fig. 20 show that the memories for the two nodes were insufficient, the transactions would be condensed, and the proposed DFP would extend to database projection

TABLE 13. Amount of transmitted data with various support thresholds (0.3–0.2%) for DFP, DistEclat, and BigFIM on IBM dataset using various memory space clients.

T20 I10 N20K D2000K			
Support (%)	DFP [5 Clients]	DistEclat [5 Clients]	BigFIM [5 Clients]
	Transmitted data (K)		
0.30%	98,901	414,002	533,623
0.28%	119,681	419,943	1,054,071
0.26%	144,602	429,740	1,067,578
0.24%	179,314	436,993	1,078,986
0.20%	269,246	454,440	OOM

[Note] OOM: The mining process was failed due to the insufficient memory.

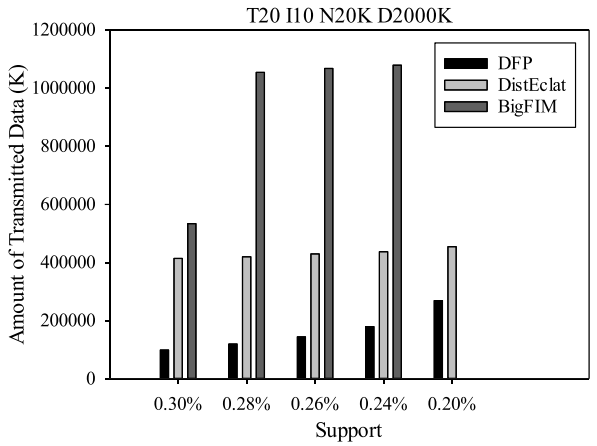

FIGURE 19. Amount of transmitted data of DFP and BigFIM with support threshold varied from 0.3 to 0.2% on IBM dataset using various memory space clients.

TABLE 14. Execution time with various support thresholds (0.28–0.2%) for Mixing Projection DFP, Database Projection DFP, DistEclat, and BigFIM on IBM dataset using insufficient memory clients.

T20 I10 N20K D2000K					
Support (%)	Estimated Memory	Mixing Projection DFP	Database Projection DFP	Dist Eclat	Big Fim
		Time(s)			
0.28%	2179M	74	76	428	314
0.26%	2531M	222	230	441	346
0.24%	2985M	317	334	491	392
0.20%	4073M	355	378	629	OOM

[Note] OOM: The mining process was failed due to the insufficient memory.

and MP for further mining. Even though the I/O cost for fetching nodes on disk increased, the performance was still better than that of DistEclat and BigFIM. The experiment result showed that the decrease of the support threshold and the mining efficiency achieved by the DFP of MP were better than those of the extended DFP of database projection. The higher performance was obtained because a smaller support threshold would produce more files for MP (Table 15).

C. SUMMARY

The above experiments showed that DFP is also much more scalable than DiscEclat and BigFIM. As to execution performance, DFP on average required only 33% of the

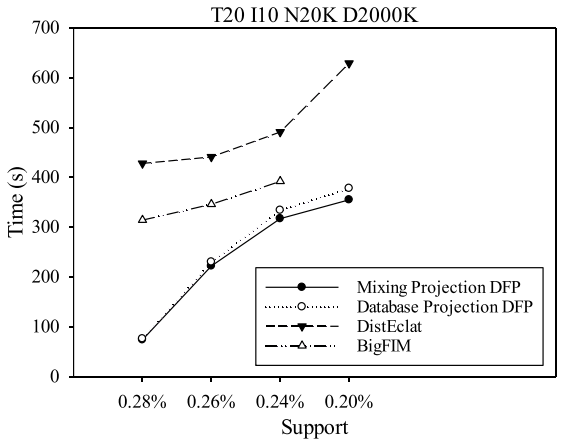

FIGURE 20. Execution performance of Mixing Projection DFP, Database Projection DFP, DistEclat, and BigFIM with support threshold varied from 0.28 to 0.2% on IBM dataset using insufficient memory clients.

TABLE 15. Produced file size of Mixing Projection DFP, and Database Projection DFP with support threshold varied from 0.28 to 0.2% on IBM dataset using insufficient memory clients.

T20 I10 N20K D2000K			
Support (%)	Mixing Projection DFP	Database Projection DFP	MP/DP (%)
	The produced file size (M)		
0.28%	89	114	78.07%
0.26%	108	136	79.41%
0.24%	134	164	81.70%
0.20%	362	422	85.78%

execution time and 45% of the transmission cost of DistEclat. Compared to BigFIM, DFP on average required 23.3% of the execution time and 14.2% of the transmission cost of BigFIM. In addition to the execution time and transmission cost, we explored the size of temporary data generated and found that our proposed method can on average reduce 15% of the DP temporary data size. Although the temporarily generated data size reduced by our method, the ratio is still high and should be improved in the future; otherwise, the execution time can significantly increase in case of large databases with large temporarily generated data sizes.

V. CONCLUSION

In this study, the DFP algorithm was proposed to improve the mining efficiency for association rule by distributed mechanism. In reported articles, the distributed mechanism cannot be applied easily for processing databases. Hence, the transmission cost would increase easily as communication among clients increases. Our experiments show that the proposed DFP algorithm successfully overcame this problem by transferring the mining result from the executing client to the server because the integrated information would go through the server instead of being communicated among clients. Our experiment results also showed through a comparison of the transmission costs that the proposed method had much better computing efficiency than DistEclat and BigFIM, required

only 45% and 14.2% of the transmission cost of DistEclat and BigFIM, respectively. The proposed method, through the estimated necessary memory and clients, reduces the computing efficiency as well. The DFP algorithm overcomes insufficient memory and repeated scans by estimating the limitation of TIDs for each client, and the remaining TIDs are processed by an extended mechanism called MP. Our experiments also showed that the modified mechanism performed better than DP. In other words, the proposed algorithm exhibits good performance during big data mining. In addition, the DFP algorithm can determine a suitable number of clients automatically according to various datasets.

The proposed DFP algorithm shows potential for future applications. For example, the mining efficiency can be improved when distributed to the same clients with highly related TIDs to reduce the branches of the FP tree. Moreover, efficiently balancing the data transmission times and workload of the FP tree in each client would benefit mining efficiency, especially when processing big databases. As the size of temporarily generated data inherited from DP is still high, we will also focus on improving the algorithm for better scalability.

REFERENCES

[1] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," presented at the 20th Int. Conf. Very Large Data Bases, Sep. 1994.

[2] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining frequent patterns without candidate generation: A frequent-pattern tree approach," *Data Mining Knowl. Discovery*, vol. 8, no. 1, pp. 53–87, 2004.

[3] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, 1993, pp. 207–216.

[4] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," *ACM SIGMOD Rec.*, vol. 29, no. 2, pp. 1–12, 2000.

[5] A. Tehreem, S. G. Khawaja, M. U. Akram, S. A. Khan, and M. Ali, "Parallel architecture for implementation of frequent itemset mining using FP-growth," in *Proc. Int. Conf. Signals Syst. (ICSigSys)*, May 2017, pp. 92–98.

[6] H. Li, Y. Wang, D. Zhang, M. Zhang, and E. Y. Chang, "Pfp: Parallel fp-growth for query recommendation," in *Proc. ACM Conf. Recommender Syst. (RecSys)*, 2008, pp. 107–114.

[7] X. Wei, Y. Ma, F. Zhang, M. Liu, and W. Shen, "Incremental FP-growth mining strategy for dynamic threshold value and database based on MapReduce," in *Proc. IEEE 18th Int. Conf. Comput. Supported Cooperat. Work Design (CSCWD)*, May 2014, pp. 271–276.

[8] A. Makanju, Z. Farzanyar, A. An, N. Cercone, Z. Z. Hu, and Y. Hu, "Deep parallelization of parallel FP-growth using parent-child MapReduce," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2016, pp. 1422–1431.

[9] D. Xia, X. Lu, H. Li, W. Wang, Y. Li, and Z. J. C. Zhang, "A MapReduce-based parallel frequent pattern growth algorithm for spatiotemporal association analysis of mobile trajectory big data," *Complexity*, vol. 2018, Jan. 2018, Art. no. 2818251.

[10] H.-Y. Chang, Y.-J. Tzang, J.-C. Lin, Z.-H. Hong, T.-Y. Chi, and C.-Y. Huang, "A hybrid algorithm for frequent pattern mining using MapReduce framework," in *Proc. 1st Int. Conf. Comput. Intell. Theory, Syst. Appl. (CCITSA)*, Dec. 2015, pp. 19–22.

[11] C.-S. Wang and J.-Y. Chang, "MISFP-growth: Hadoop-based frequent pattern mining with multiple item support," *Appl. Sci.*, vol. 9, no. 10, p. 2075, May 2019.

[12] S. Moens, E. Aksehirli, and B. Goethals, "Frequent itemset mining for big data," in *Proc. IEEE Int. Conf. Big Data*, Oct. 2013, pp. 111–118.

[13] A. D. D. Gassama, F. Camara, and S. Ndiaye, "S-FPG: A parallel version of FP-growth algorithm under Apache Spark," in *Proc. IEEE 2nd Int. Conf. Cloud Comput. Big Data Anal. (ICCCBDA)*, Apr. 2017, pp. 98–101.

[14] G. Grahne and J. Zhu, "Mining frequent itemsets from secondary memory," in *Proc. 4th IEEE Int. Conf. Data Mining (ICDM04)*, Nov. 2004, pp. 91–98.

[15] P. Y. Taser, K. U. Birant, and D. Birant, "Multitask-based association rule mining," *Turkish J. Elect. Eng. Comput. Sci.*, vol. 28, no. 2, pp. 933–955, 2020.

[16] A. Javed and A. Khokhar, "Frequent pattern mining on message passing multiprocessor systems," *Distrib. Parallel Databases*, vol. 16, no. 3, pp. 321–334, Nov. 2004.

[17] Y. Qiu, Y.-J. Lan, and Q.-S. Xie, "An improved algorithm of mining from FP-tree," in *Proc. Int. Conf. Mach. Learn. Cybern.*, Aug. 2004, pp. 1665–1670.

[18] J. Zhou and K.-M. Yu, "Tidset-based parallel FP-tree algorithm for the frequent pattern mining problem on PC clusters," in *Proc. Int. Conf. Grid Pervas. Comput.*, May 2008, pp. 18–28.

[19] J. Zhou and K.-M. Yu, "Balanced tidset-based parallel FP-tree algorithm for the frequent pattern mining on grid system," in *Proc. 4th Int. Conf. Semantics, Knowl. Grid*, Dec. 2008, pp. 103–108.

[20] K. W. Lin and Y.-C. Luo, "A fast parallel algorithm for discovering frequent patterns," in *Proc. IEEE Int. Conf. Granular Comput.*, Aug. 2009, pp. 398–403.

[21] Z. Cai, X. Zhu, Y. Zheng, D. Liu, and L. Xu, "A caching-based parallel FP-growth in Apache Spark," in *Proc. Int. Conf. Algorithms Archit. Parallel Process.*, Nov. 2018, pp. 519–533.

[22] Y. Miao, J. Lin, and N. Xu, "An improved parallel FP-growth algorithm based on spark and its application," in *Proc. Chin. Control Conf. (CCC)*, Jul. 2019, pp. 3793–3797.

[23] Y. Zhang and L. Wang, "A optimization algorithm for association rule based on spark platform," in *Proc. Int. Conf. Comput. Netw., Electron. Autom. (ICCNEA)*, Sep. 2020, pp. 82–86.

[24] (Mar. 4, 2014). *Apache Spark—Unified Analytics Engine for Big Data*. Apache Software Foundation. Accessed: Jul. 21, 2021. [Online]. Available: https://spark.apache.org/

[25] D. Quintero, *IBM Platform Computing Solutions*. Redwood Shores, CA, USA: IBM Redbooks, 2012.

[26] S. Bagui and P. C. Dhar, "Positive and negative association rule mining in Hadoop's MapReduce environment," *J. Big Data*, vol. 6, no. 1, pp. 1–16, Dec. 2019.

[27] M. J. Zaki. (Oct. 6, 2016). *IBM Generator: IBM Synthetic Data Generator for Itemsets and Sequences*. Github. Accessed: Jul. 21, 2021. [Online]. Available: https://github.com/zakimjz/IBMGenerator

[28] B. Goethals, "Frequent itemset mining dataset repository," in *Proc. FIMI Workshop Committee*, Jan. 2019. Accessed: Jul. 21, 2021. [Online]. Available: http://fimi.uantwerpen.be/data/

PENG-YU HUANG received the B.S. degree from the Department of Communication Engineering, I-Shou University (ISU), Taiwan, in 2013, and the M.S. degree from the Department of Computer Science and Information Engineering, National Kaohsiung University of Applied Sciences, Taiwan, in 2015. He is currently pursuing the Ph.D. degree with the Department of Electronic Engineering, National Kaohsiung University of Science and Technology, Taiwan. His research interests include data mining and distributed computing.

WAN-SHU CHENG received the Ph.D. degree from the Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan, in 2014. Since August 2019, she has been an Assistant Professor with the Department of Electrical Engineering, National Kaohsiung University of Science and Technology. Her current research interests include data mining and its applications, sensor technologies, computer vision, big data analysis, and precision medicine.

JU-CHIN CHEN received the B.S., M.S., and Ph.D. degrees in computer science and information engineering from the National Cheng Kung University, Tainan, Taiwan, in 2002, 2004, and 2010, respectively. She is currently an Associate Professor with the Department of Computer Science and Information Engineering, National Kaohsiung University of Science and Technology, Taiwan. Her research interests include machine learning, computer vision, and pattern recognition.

WEN-YU CHUNG received the B.S. and M.S. degrees in computer science from the National Tsing Hua University, Hsinchu, Taiwan, in 2000 and 2002, respectively, and the Ph.D. degree in computer science and engineering from Pennsylvania State University, PA, USA, in 2009. She is currently an Assistant Professor with the Department of Computer Science and Information Engineering, National Kaohsiung University of Science and Technology, Kaohsiung, Taiwan.

YOUNG-LIN CHEN received the M.S. degree from the Department of Computer Science and Information Engineering, National Kaohsiung University of Applied Sciences, Taiwan, in 2016. He is currently an Engineer at Foxconn Technology Group.

KAWUU W. LIN received the B.S. degree from the Department of Computer Science and Information Engineering, National Taiwan University (NTU), Taiwan, in 1999, and the Ph.D. degree from the Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan, in 2006. Since August 2007, he has been an Assistant Professor with the Department of Computer Science and Information Engineering, National Kaohsiung University of Applied Sciences, and became an Associate Professor and a Professor, in August 2012 and August 2015, respectively. He is currently a Professor with the Department of Computer Science and Information Engineering, National Kaohsiung University of Science and Technology, Kaohsiung, Taiwan. His research interests include data mining and its applications, image recognition, and parallel and distributed computing. He served or is serving on several workshop chairs and technical program committees for many international conferences.