



## Article

# A Distributed Algorithm for Fast Mining Frequent Patterns in Limited and Varying Network Bandwidth Environments

Chun-Cheng Lin <sup>1</sup>, Wei-Ching Li <sup>1</sup>, Ju-Chin Chen <sup>2</sup>, Wen-Yu Chung <sup>2</sup>, Sheng-Hao Chung <sup>1</sup> and Kawuu W. Lin <sup>2,\*</sup>

<sup>1</sup> Department of Industrial Engineering and Management, National Chiao Tung University, Hsinchu 30010, Taiwan; cclin321@nctu.edu.tw (C.-C.L.); f.l.frida.kahlo@gmail.com (W.-C.L.); ahoward130@gmail.com (S.-H.C.)

<sup>2</sup> Department of Computer Science and Information Engineering, National Kaohsiung University of Science and Technology, Kaohsiung 824, Taiwan; jc.chen@nkust.edu.tw (J.-C.C.); wychung@nkust.edu.tw (W.-Y.C.)

\* Correspondence: linwc@nkust.edu.tw

Received: 18 March 2019; Accepted: 29 April 2019; Published: 6 May 2019



**Abstract:** Data mining is a set of methods used to mine hidden information from data. It mainly includes frequent pattern mining, sequential pattern mining, classification, and clustering. Frequent pattern mining is used to discover the correlation among various sets of items within large databases. The rapid upward trend in data size slows the mining of frequent patterns. Numerous studies have attempted to develop algorithms that operate in distributed computing environments to accelerate the mining process. FLR-mining (Fast, Load balancing and Resource efficient mining algorithm) is one of the fastest methods of mining with efficient consideration of load balancing and resources. FLR-mining can automatically determine the appropriate number of computing nodes. However, FLR-mining and existing methods assume that the network bandwidth is constant. In practical distributed and many-task computing systems, this assumption fails because there are packet collisions caused by many mining tasks that run in a simultaneous manner. Therefore, a method that can consider the varying network bandwidth is necessary. In this study, we propose a method that can rapidly mine frequent patterns under the varying network bandwidth. The proposed method can also determine the appropriate number of computing nodes to efficiently utilize computing resources and achieve load balancing. Through empirical evaluation, the proposed method is shown to deliver excellent performance in terms of execution efficiency and load balancing.

**Keywords:** data mining; frequent pattern mining; distributed mining; parallel mining

## 1. Introduction

Today, electronic communication users generate various kinds of information at increasing rates. Data mining is a data analysis approach for identifying information of interest that is contained in data. Data mining is successfully applied in various fields, including statistics, artificial intelligence, expert systems, visualization, and machine learning. Data mining mainly includes frequent pattern mining [1], sequential mining [2], classification, and clustering. This method can discover hidden information from a dataset and can also be applied to various fields like market basket analysis, discovery of high-utility patterns [3] and so forth. A typical frequent pattern mining problem is discovering the itemsets with support greater than or equal to a specific threshold in a database. An itemset is a combination of different items, and the support is a value reflecting how frequently the itemset appears in the database. The support of  $X$  with respect to database  $D$  is defined as the proportion of a number of transactions in the database containing the itemset  $X$ . Frequent pattern mining is the core method of data mining,



which can determine that an itemset satisfies the minimum support threshold. The support threshold also influences the outcome of frequent patterns.

The algorithm for discovering frequent patterns is separated into two parts: an Apriori-like approach and a frequent-pattern (FP) tree. Apriori is the first algorithm proposed by Agrawal et al. [1] for mining association rules. First, the user-defined minimum support threshold is used to find itemsets, and then frequent patterns from these itemsets are discovered. This algorithm generates many candidate itemsets when constructing association rules; thus, it needs to scan the database several times. The execution efficiency is very low. To improve the execution time, an algorithm called FP-growth has been previously proposed, which is an extended prefix-tree approach. This algorithm first scans the database and then records a list of items in descending order of frequency. Further, it scans the database again, obtains the conditional pattern bases from the FP tree, and constructs a conditional FP tree from the conditional pattern bases. Using the FP-tree structure just only scans the database twice, which reduces the input/output time and improves the execution efficiency.

With the increasing amount of data being produced, the size of databases must also increase. Big data is a term describing datasets that are too large to handle in conventional ways. As Nate Silver, the world-famous statistician, said, “when data gets big, big problems can arise.” This means that a number of problems are associated with big data, and one of the most pressing problems is executive performance. Several research approaches use distributed or parallel systems such as grid computing [4,5] or other computing techniques to handle big data, which we review in this study. To accelerate the mining process and improve execution performance, numerous studies have attempted to develop algorithms that operate in distributed computing environments. Fast mining is the core for big data applications [6,7].

Lin et al. [8] proposed five dispatch methods based on the cloud-based association rule mining (CARM) [9] algorithm: (a) equal working set (EWS), (b) request on demand (ROD), (c) small size working set (SSWS), (d) progressive size working set (PSWS), and (e) FLR-mining [8] algorithms. EWS uses an equal allocation of computing nodes; however, the completed time of each computing node is unknown, which results in unbalanced loading. The ROD algorithm is proposed to deal with the unbalanced loading of EWS. This method involves a kernel node to assign new tasks after a computing node completes a task. Therefore, the ROD algorithm can address the problem of unbalanced loading; however, the algorithm spends much time in communication. SSWS and PSWS are proposed to solve the drawbacks of the abovementioned methods. The difference between SSWS and PSWS is the former uses the fixed parameter to assign work items, but the parameter is dynamic in the latter. The above-described methods are all based on the CARM architecture, whose algorithms require users to determine the number of computing nodes, which can lead to inefficient execution. To solve this problem, the author in [10] proposed the FLR-mining algorithm, which efficiently allocates the computing nodes. FLR-mining is the fastest algorithm among the five methods. It efficiently considers load balancing and resources, as well as determines the appropriate number of computing nodes automatically.

However, FLR-mining and the existing methods assume that the network bandwidth is constant. In practical distributed and many-task running systems, the assumption fails because there are packet collisions caused by many mining tasks running simultaneously. As a result, a method that can consider the varying network bandwidth is necessary. Therefore, we propose the fast distributed mining in changing network bandwidth (FDCNB-MINING) algorithm to rapidly mine frequent patterns under the varying network bandwidth. This algorithm can also determine the appropriate number of computing nodes to efficiently utilize the computing resources and achieve load balancing. Through empirical evaluation, the proposed method is shown to deliver varying network and bandwidth environmental issues.

The rest of this paper is structured as follows. We briefly review related work in Section 2. In Section 3, we introduce the FDCNB-MINING algorithm. In Section 4, we compare our proposed



algorithm with FLR-mining under a constant network bandwidth in various simulation conditions, as well as discuss the experimental results. In Section 5, we present the conclusions of this study.

## 2. Related Work

Apriori is the first algorithm proposed by Agrawal et al. [1] for mining association rules, and it is widely used in various fields. This algorithm is based on the concept of candidate itemsets. When the support value of a candidate itemset satisfies the minimum support threshold, then this candidate itemset is called a large itemset. Although the Apriori algorithm can discover frequent patterns effectively, its disadvantage is the long execution time for scanning databases multiple times to generate the candidate itemset.

Chen et al. [11] proposed the direct hashing and pruning (DHP) algorithm, which is based on the Apriori algorithm that can use a hash table and prunes the size of the candidate  $K+1$  itemsets generated at each step. This method can generate large itemsets more efficiently than the Apriori algorithm. The experimental results also proved that when the size of the database grows, the efficiency of the DHP algorithm is better than that of the Apriori algorithm. However, the performance of the DHP algorithm depends strongly on the hash table size. To mitigate the drawbacks of the DHP Algorithm, Ozel et al. [12] proposed the perfect hashing and pruning (PHP) algorithm, which reduces the size of the database and leverages perfect hashing for the hash table generated at each stage. Although the PHP algorithm can solve the collision problem of the DHP algorithm, it also demands more memory space to store the hash table. Therefore, Agarwal et al. [13] proposed the transaction hashing and pruning (THP) algorithm, which created a new scheme for the hash table. As a result, the THP algorithm can either overcome the disadvantage of the DHP algorithm or the problem of the PHP algorithm.

Brin et al. [14] proposed the dynamic itemset counting (DIC) algorithm, in which the database is partitioned into blocks. The DIC algorithm can reduce the number of passes and it searches the database as unit blocks to overcome the high execution time in the Apriori algorithm. However, this method should consider how to suit the block size. With increasing data, previous studies focused on the parallel-distributed structure. Yang et al. [15] proposed an Apriori algorithm based on MapReduce mode, which uses Hadoop distributed file system to handle massive datasets. In [16], the authors proposed the distributed parallel Apriori (DPA) algorithm, which is based on the Apriori algorithm. The main concept is to avoid the problems associated with multiple scans by implementing the message passing interface (MPI) method, which is a message passing system for programming parallel computers. MPI can be used to create parallel programs in C, C++, Fortran, or Python. The DPA algorithm reduces the database scanning time and also achieves a better load balance by storing transaction identifications (TIDs) in a table structure. However, this method has the same drawbacks as the Apriori algorithm; they both generate candidate itemsets during the process of mining association rules.

To overcome the problem of the Apriori algorithm, Han et al. [17] proposed the FP-growth algorithm. The FP-growth algorithm operates in two steps: (a) scan the database and then record a list of frequent items in descending order; (b) scan the database for the second time, obtain conditional pattern bases from the FP tree, and construct a conditional FP tree from the conditional pattern bases. The conditional FP tree is recursively mined until the tree contains a single item. This algorithm only scans the database twice and does not generate a candidate itemset. However, the main disadvantage of FP-growth is that the FP tree may not fit in memory. Zaiane et al. [18] proposed the multiple local frequent pattern tree algorithm based on the FP-growth algorithm, in which the FP tree is divided in chunks and the shared counters are used to cumulate the frequencies. This algorithm can efficiently improve the problem of the dependent, but it may have issues if the shared memory is locked. Another algorithm based on the FP-growth algorithm is proposed in [17]. This approach is a divide-and-conquer methodology which consists of two stages, namely, a mapper part and a reduce phase. In the mapper part, the database is converted into new databases of group-dependent transactions. In the reduce phase, the FP tree is constructed and mined recursively. During this process,



patterns are also discovered. Shang et al. [19] proposed a novel idea: The utilization of SQL-based frequent pattern mining while using the FP-growth algorithm. The algorithm uses an FP table to store the related data that comprise items, counts, and paths. In the FP-tree construction process, it decides each field path after comparing the already existing paths. If the path already exists, then the algorithm will update the FP table and increase the item's count by 1, but if not, then it will only update the FP table. To improve efficiency, the algorithm uses a temporary table to store the FP tree, and it also uses SQL queries to generate the FP tree. This method can get efficient performance and also speed up the construction of the FP tree. In order to reduce the memory consumption of FP-growth, Schlegel et al. [20] proposed the CFP-growth algorithm. It uses order of magnitude to overcome the problem of tree size.

Javed et al. [21] proposed the parallel FP-tree (PFP) algorithm, which is based on the FP-tree data structure. This method uses the special tree exchange technique to reduce the execution time problem of the FP-tree algorithm. However, the PFP algorithm continues to experience the execution time issue when the number of the processors increases. In addition to scanning the entire database, Zhou et al. [22] used a tidset to exchange information in cluster computing environments. Subsequently, the extension method, known as the balanced tidset parallel algorithm (BTP-tree) [23], was derived from the TPFP [22] algorithm. The results show the BTP-tree to have better performance than the TPFP and PFP algorithms. Several research approaches now use distributed and parallel systems to solve big data problems. One such system that has recently become popular is Hadoop, an open-source software framework that some authors [24,25] have used to improve performance.

In [9], Lin et al. proposed the CARM algorithm, which consists of high-workability distributed FP-mine (HD-mine) and fast distributed FP-mine (FD-mine). HD-mine splits data for easy mining and then merges the results of computing nodes to discover frequent patterns. The main drawback of HD-mine is its inefficiency. Therefore, FD-mine is proposed to solve this problem. Given that FD-mine occupies more memory than HD-mine, HD-mine is first used to discover frequent patterns. If HD-mine fails, we will use FD-mine in the entire mining process. The CARM algorithm also has privacy-preserved abilities.

Lin et al. proposed the following five methods: equal working set (EWS), request on demand (ROD), small-size working set (SSWS), progressive size working set (PSWS), and FLR-mining algorithms. These five algorithms are all based on the CARM algorithm (as we mentioned above). The EWS algorithm first establishes the header table and the FP tree through the database. Then the EWS algorithm assigns each computing node to deal with a  $1/N$ th part of the working set, where  $N$  is the number of computing nodes. The drawback of the EWS algorithm is that the required mining time is uncertain, it can have unbalanced workloads. As a result, the ROD algorithm was created to solve the unbalanced workload shortcoming of the EWS algorithm. The ROD method can reduce the idle time because the kernel node assigns a new task to each computing node when it finishes its previous task. However, the kernel node needs to examine idle computing nodes and also dispatch work; therefore, this process normally uses longer transmission times. To overcome the problems of the EWS and ROD algorithms, a method called SSWS was proposed. The SSWS algorithm uses a fixed parameter  $L$  to assign the number of work items in individual tasks given to each mining computing node. On account of the fact that the parameter  $L$  is fixed, it will also have some problematic issues from this assumption. For example, if the parameter  $L$  is set to one, then the time complexity of the SSWS algorithm is the same as the ROD algorithm. On the other hand, if the parameter  $L$  is too large, then it may increase the time spent in network communication. Therefore, to find an improved balance, the PSWS algorithm was proposed, which can determine a good size for the parameter  $L$  automatically. Test results show that the PSWS algorithm is one of the most efficient algorithms for finding frequent patterns.

Although the PSWS algorithm was presented as the fastest algorithm among the four algorithms by Lin et al. discussed so far, the PSWS algorithm does not define the appropriate number of computing nodes; rather, for this algorithm, this number is determined by an expert's experience. If this node number setting is not fair, then it might lead to some computing nodes being idle and the waste



of resources. Owing to the fact that each computing node starts a mining task after it receives the compressed FP tree, the PSWS algorithm might waste too much time allocating tasks to too many nodes. To overcome this problem, the DAN-mining method, proposed by [26], automatically determines the appropriate number of nodes to complete a mining task with respect to frequent patterns. This algorithm, which is also based on the CARM architecture, evaluates the required mining time by sampling  $y\%$  of header items from the header table. Although the DAN-mining algorithm can determine how many nodes are needed to mine the data, it has no criteria for determining the value of  $y$ . Therefore, the FLR-mining algorithm [10] was also proposed to determine the appropriate number of computing nodes automatically. In the FLR-mining algorithm environmental structure, it needs the kernel node, which stores the FP tree and the compressed FP tree. In the beginning of an assignment strategy, the FP tree and all tasks are sent to the first computing node. And, at the same time that the first computing node starts executing the first mining task, the second computing node also receives the FP tree. After receiving a process/task complete message from the first node, the kernel node sends a message to the first computing node to request the information about how the first node performed and how long it took, during the same time that the second computing node was receiving the FP tree. The kernel node utilizes this information to calculate the average time used to complete each task. From then on, the kernel node can calculate how many tasks can be assigned during the time the FP tree was sent to the computing node. After the kernel node finishes the calculation, it then reallocates tasks to the first and second computing nodes. If not all the tasks are finished, then the kernel node will transmit the compressed FP tree to a third computing node. After that, the kernel node will again collect the information on the finished tasks and reallocate tasks to the same number or a larger number of computing nodes as appropriate. The process is repeated until all the tasks are done.

In the following algorithm proposed by Lin et al., FLR-mining is one of the fastest algorithms for mining frequent patterns in distributed computing environments based on CARM architecture. FLR-mining also considers the factors of load balancing and resources efficiently, and it can determine the appropriate number of computing nodes automatically. But there is still a big problem. Previous studies assumed that the network bandwidth is constant, but this assumption fails in a real network environment. There are packet collisions caused by many mining tasks running simultaneously. To avoid these drawbacks and make the distributed cluster computations more suitable for a real network environment, we will make our environmental structure more robust in this paper; therefore, the FDCNB-MINING algorithm based on the CARM algorithm is proposed.

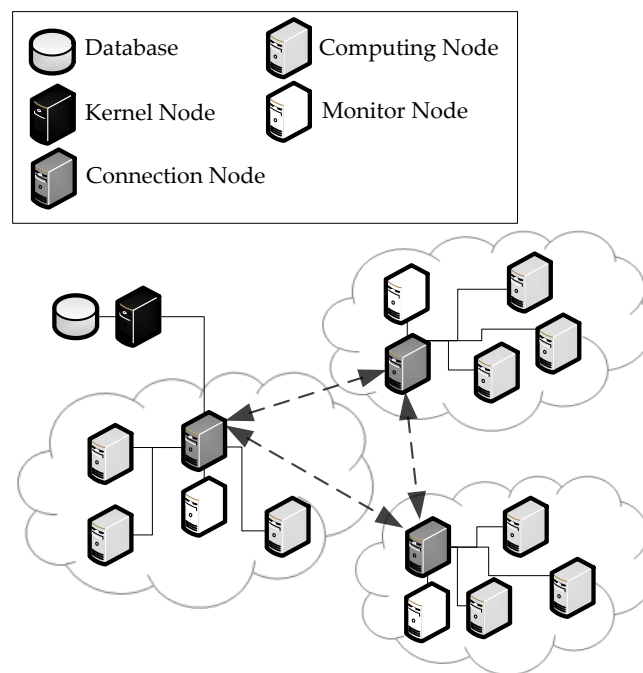
### 3. Proposed Method

This section details the proposed method. In Section 3.1, we introduce an improved environmental structure. In Section 3.2, our proposed algorithm is described in detail.

#### 3.1. System Structure

FLR-mining [10] is an efficient algorithm with good load balancing; its architecture is based on CARM [9] for improvement. To solve varying network bandwidth environmental problems, we must strengthen the environmental structure of CARM. In addition to the original structure of the database, kernel node, connection node, and computing node, a new node (monitor node) is added for strength. The structure is shown in Figure 1.





**Figure 1.** System structure of the fast distributed mining in changing network bandwidth (FDCNB-MINING) algorithm.

The kernel node is responsible for reading the database, building an FP tree, compressing the FP tree into a zip file, and assigning tasks to the computing node. The computing node is responsible for mining FP-growth from assigned tasks of the kernel node. The connection node is the intermediate communication link between the kernel node and computing node. The newly added monitor node observes the varying network bandwidth in cloud or distributed computing, as well as helps the kernel node select an optimal computing node. The monitor node sends a test package to each computing node in regular time intervals, so that the network bandwidth in cloud or distributed computing is known. The monitor node also assists the kernel node in selecting the currently preferred computing node. The monitor node informs the kernel node whether to discontinue transmission and change into a better computing node or continue transmission to avoid poor bandwidths, which can affect the overall efficiency.

### 3.2. FDCNB-MINING Algorithm

To solve the problem of varying network bandwidths, we propose the FDCNB-MINING algorithm, namely, fast distributed mining in changing network bandwidth. The algorithm is as shown in Section 3.3 and is conducted as follows:

- Step 1 Before the kernel node is activated, the monitor node must begin operation to constantly monitor the network bandwidth environment. (Line 1)
- Step 2 When the kernel node needs to select one of the computing nodes to transmit compressed files, the monitor node provides the best computing node to the kernel node. (Line 5)
- Step 3 If the monitor node finds the kernel node of using low bandwidth during transfer, it informs the kernel node to discontinue transmission and change into a better computing node to transfer information; otherwise, transmission is continued. FDCNB-MINING sorts computing nodes in descending order according to the performance index of bandwidth that reflects their real-time network bandwidth. A packet with size  $s$  is sent to each computing node periodically from the kernel node, and the computing node acknowledges the kernel node after receiving the packet. In this way, the response time between kernel node and every computing node can be obtained. The performance index of bandwidth  $B_p$  is defined as  $s/t$ , where  $s$  is the size



of transferred packet and  $t$  is the elapsed time the packet sent from the kernel node to the computing node. Paused transmitted data are saved in the computing node until the monitor node finds a recovery bandwidth. The kernel node then waits to be informed by the monitor node to mine and continue to transfer the unfinished part. (Line 7–10)

Step 4 This process is repeated until the kernel node does not need the computing node to help in the mining operation.

### 3.3. Example

In this section we describe an example of the FDCNB-MINING algorithm. Suppose the environment has one kernel node, one connection node, one monitor node, and four computing nodes (A, B, C, and D) for distributed computing. After transferring the test packet, the monitor node determines that the network bandwidth of the computing nodes is as follows: A has 6 Mbps, B has 45 Mbps, C has 93 Mbps, and D has 30 Mbps. Therefore, when the kernel node needs to select one of the computing nodes to assist the mining, the monitor node provides the best computing node, i.e., node C, to the kernel node. If the kernel node still needs to select one of the computing nodes to assist the mining again and the bandwidth has not changed, then the monitor node provides the next best node, i.e., node B, to the kernel node and so forth.

---

#### Algorithm FDCNB-MINING

Input A transaction database  $D$ , minimum support  $S$ .

Output Frequent Patterns  $FP$

---

```

1      Monitor.work ();
2      HT = getHT(D, S);
3      tree = buildFPtree(D, S);
4      WHILE (isNotEmty(HT))
5          n = Monitor.getBestNode();
6          transmitTree(tree, n);
7          IF(Monitor.findLowBandwidth(n))
8              n.stopTransmitting();
9              CONTINUE;
10         END IF
11         hi = getMiningTask(HT);
12         HT = HT – hi;
13         FLR-Mining.BeginMining(n, hi);
14     END WHILE
15     RETURN FP

```

---

## 4. Experimental Results

### 4.1. Experiment Setup

To estimate the performance of our proposed method, we use IBM's Quest synthetic data generator [27] to generate the workload. The data were based on general transaction record databases, including the number of transactions ( $D$ ), average frequent itemset length ( $I$ ), number of items ( $N$ ), and average transaction length ( $T$ ).

This experimental environment comprised 13 computers to execute an experimental analysis, one kernel node, one connection node, one monitor node, and other computing nodes for distributed computing. Each node was equipped with an Intel(R) Core(TM) i7-2600K CPU @3.40 GHz, 8 GB of memory (Intel Corporation, Santa Clara, CA, USA), 500 GB of storage executing on Windows 7 Enterprise Edition operating system (Microsoft Corporation, Redmond, WA, USA). The programs were written in Java, and the communication between nodes used Java Remote Method Invocation and socket.



To evaluate the performance of FDCNB-MINING, we compared it with FLR-mining [10] by conducting a series of experiments. FLR-mining is a known efficient method for mining frequent patterns in such environments. We used the same dataset settings as [10] for a fair evaluation, so our basic dataset for experiments is D100K I5 N100K T20 under varying support. In addition, we used two real data sets for experiments under varying levels of support. The data sets are respectively retail-market [28] and accidents [29] data, which were downloaded from frequent itemset mining implementations (FIMI) [30].

We simulated variation in network bandwidth as shown in Table 1. We simulated six different network bandwidth cases in computing nodes. In real applications, FDCNB-MINING can use the performance index of bandwidth  $B_p$  to measure the bandwidth value at each time point; for simplicity in the performance discussion, the bandwidth value from the kernel node to each computing node was as shown in Table 1, and the value was kept the same all the time in each case. Case 1 assumed that all computing nodes are in the 100 Mbps network bandwidth. Case 2–6 were randomly generated computing nodes in the 100–1 Mbps network bandwidth, and a bandwidth utilization rate of 80% was assumed.

**Table 1.** Simulation of different network bandwidths used in the experiments.

Node	Network Bandwidth (Mbps)									
	1	2	3	4	5	6	7	8	9	10
Case 1	Bandwidths are all 100									
Case 2	3	80	6	77	35	69	87	63	14	22
Case 3	38	60	88	95	100	36	80	69	68	20
Case 4	87	19	36	100	38	92	25	61	76	68
Case 5	34	15	42	41	50	82	90	57	56	93
Case 6	42	3	4	10	95	14	78	44	59	7

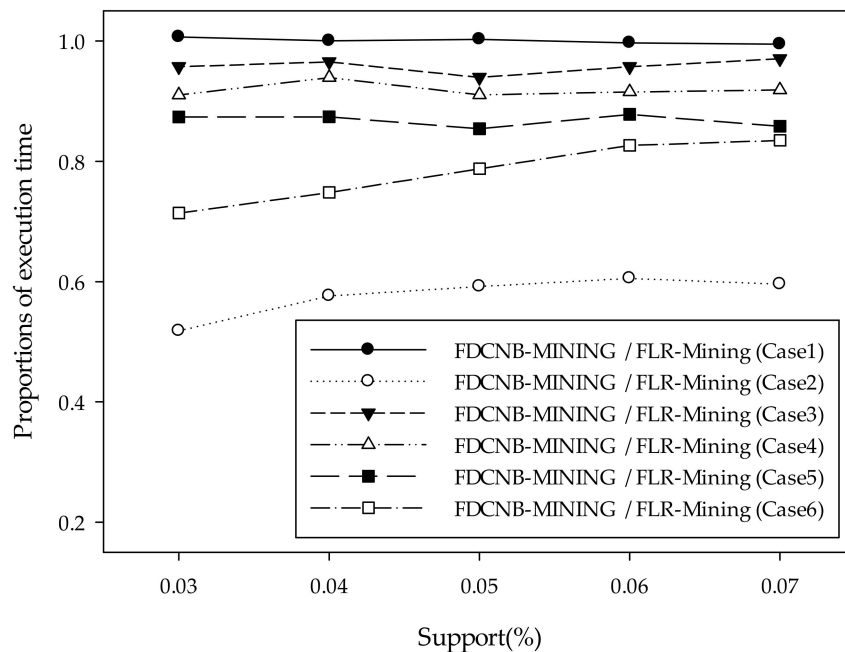
## 4.2. Experiment Analysis

### 4.2.1. Synthetic Data

In the experiments, we varied the support from 0.07% to 0.03% for FDCNB-MINING and FLR-mining in six different simulation cases of network bandwidth to study its effect on proportions of execution time. The number of transactions, average frequent itemset length, number of items, and average transaction length were 100 K, 5, 100 K, and 20, respectively. Network bandwidth had six simulation cases, which are shown in Table 1.

The experimental results shown in Figure 2 reveal that the proportion of the execution time of FDCNB-MINING/FLR-mining was close to 1 in Case 1. This finding represents the addition of a monitor node in the stable network; our proposed method exerted a minimal effect on the overall execution efficiency. In the unstable bandwidth in Case 2, FDCNB-MINING required 59.60% of the execution time of FLR-mining when the support was 0.07%. FDCNB-MINING required 51.79% of the execution time of FLR-mining when the support was 0.03%. Thus, FLR-mining had poor execution efficiency when an unequal or unstable network bandwidth was encountered. FLR-mining asks the connection node to obtain the available computing node by node order, and it does not consider the unequal or unstable bandwidth in a real network.





**Figure 2.** Effect of varying support from 0.03% to 0.07% on proportions of execution time in FDCNB-MINING and FLR-mining.

Therefore, we propose the FDCNB-MINING method to improve the problem of varying network bandwidth environments through the monitor node helping the kernel node to select the best node. For other simulation cases, the results show that the proposed method FDCNB-MINING delivered excellent performance in terms of execution efficiency and scalability compared with the FLR-mining algorithm. The detailed execution time ratio is shown in Table 2. In the experiments, we also observed that the average proportion of execution time of FDCNB-MINING to that of FLR-mining was about 85%, meaning that the proposed method can save 15% of the execution time of FLR-mining for the synthetic datasets.

**Table 2.** Proportions of execution time for FDCNB-MINING/FLR-mining while varying support from 0.03% to 0.07% and for Case 1 to Case 6.

Support (%)	0.07	0.06	0.05	0.04	0.03
Case1	0.9947	0.9970	1.0028	1.0002	1.0066
Case2	0.5960	0.6056	0.5921	0.5764	0.5179
Case3	0.9707	0.9573	0.9396	0.9656	0.9575
Case4	0.9188	0.9154	0.9107	0.9395	0.9105
Case5	0.8583	0.8781	0.8542	0.8742	0.8739
Case6	0.8350	0.8265	0.7878	0.7485	0.7140

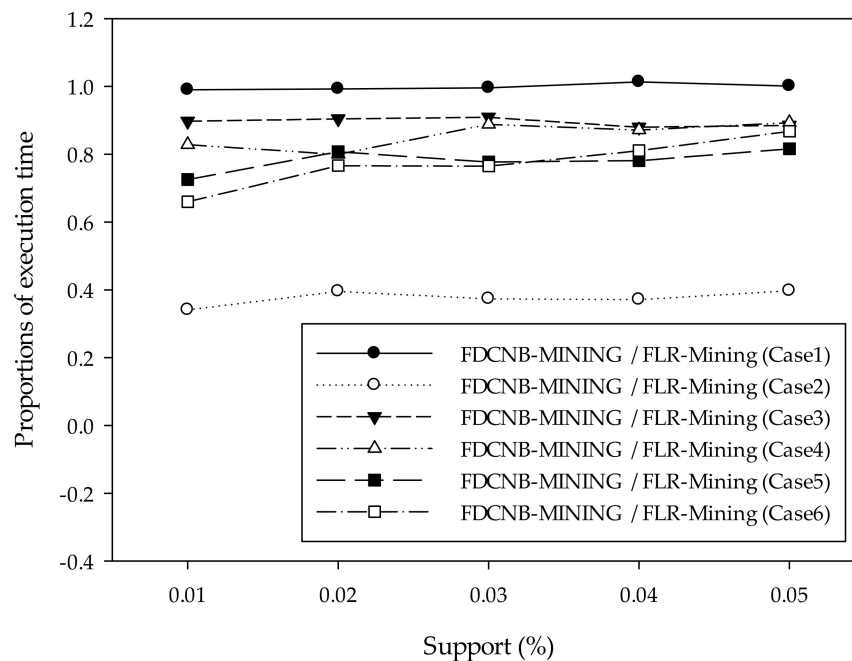
#### 4.2.2. Real Data—Retail

We varied the support level from 0.05% to 0.01% for the FDCNB-MINING and FLR-mining algorithms in six different simulation cases of the network bandwidth environment to study the environment effects on the ratios of execution times. The data [18] was retail-market data from an anonymous Belgian retail store. The number of transactions, number of items, and the average transaction length were 88,162, 16,470, and 10, respectively. Table 1 describes the six network bandwidth environment cases.

Figure 3 shows that the ratio of the execution times of the two algorithms (here denoted FDCNB-MINING/FLR-mining) was close to 1 in Case 1. In Case 2, where there were severe changes in the network bandwidth environment, the FDCNB-MINING algorithm only required 34.14% of the execution time of the FLR-mining algorithm when the support level was 0.01%. In Cases 3 to 6,



our proposed FDCNB-MINING method required about 82% of the execution time of the FLR-mining algorithm. Table 3 shows the detailed execution time ratios against support levels. In frequent pattern mining, a lower support threshold resulted in a much higher number of frequent patterns and is therefore one of the performance bottlenecks. We observed that the proportion of execution time of FDCNB-MINING to that of FLR-mining decreased with the decrease in support threshold from 0.05% to 0.01% for most of the cases, meaning that the proposed FDCNB-MINING was more scalable than FLR-mining. The average proportion of execution time of FDCNB-MINING to that of FLR-mining was about 78%.



**Figure 3.** Effect of varying support from 0.01% to 0.05% on proportions of execution time in FDCNB-MINING and FLR-mining.

**Table 3.** Proportions of execution time for FDCNB-MINING/FLR-mining while varying support from 0.01% to 0.05% and Case1 to Case 6.

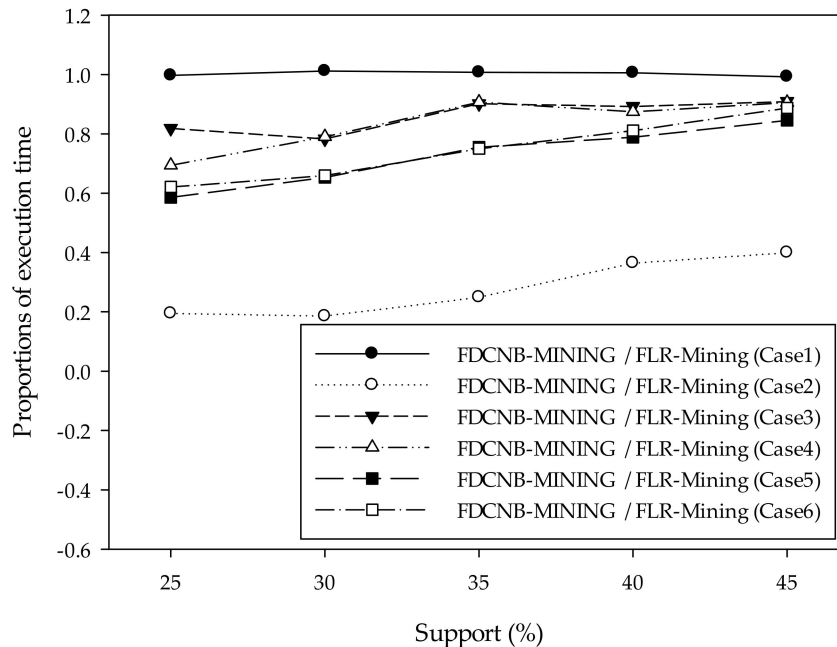
Support (%)	0.05	0.04	0.03	0.02	0.01
Case 1	1.0013	1.0138	0.9960	0.9927	0.9904
Case 2	0.3978	0.3713	0.3738	0.3957	0.3414
Case 3	0.8857	0.8801	0.9093	0.9048	0.8976
Case 4	0.8934	0.8720	0.8885	0.8001	0.8284
Case 5	0.8163	0.7814	0.7776	0.8078	0.7255
Case 6	0.8678	0.8108	0.7655	0.7668	0.6603

#### 4.2.3. Real Data—Accidents

In order to study the effect of the support level on execution time ratios, we varied the support levels from 45% to 25% for the FDCNB-MINING and FLR-mining algorithms in six different simulation cases of the network bandwidth environment. The data [19] covered traffic accidents in the Flanders region of Belgium for the period from 1991 to 2000. The number of traffic accidents, number of different attribute values, and average length of each accident were 340,183, 572, and 45, respectively. Table 1 shows the network bandwidth environment of the six simulation cases. The results are shown in Figure 4. We found that the FDCNB-MINING algorithm had a good performance in the unstable network bandwidth environment and was superior to the FLR-mining algorithm. The detailed execution time ratios are listed in Table 4. In the experiment, the proportion of execution time of



FDCNB-MINING to that of FLR-mining also decreased with the decrease in support threshold from 45% to 25% for most of the cases, demonstrating that the proposed FDCNB-MINING was more scalable than FLR-mining. The average proportion of execution time of FDCNB-MINING to that of FLR-mining was about 74%.



**Figure 4.** Effect of varying support from 25% to 45% on proportions of execution time in FDCNB-MINING and FLR-mining.

**Table 4.** Proportions of execution time for FDCNB-MINING/FLR-mining while varying support from 25% to 45% and Case 1 to Case 6.

Support (%)	45	40	35	30	25
Case1	0.9922	1.0057	1.0074	1.0117	0.9970
Case2	0.3996	0.3643	0.2496	0.1854	0.1943
Case3	0.9090	0.8926	0.9020	0.7831	0.8185
Case4	0.9054	0.8746	0.9073	0.7900	0.6940
Case5	0.8455	0.7885	0.7552	0.6538	0.5861
Case6	0.8875	0.8110	0.7500	0.6597	0.6211

## 5. Conclusions

In this paper, we have proposed the FDCNB-MINING algorithm able to determine the appropriate number of computing nodes to mine frequent patterns when the network bandwidth is varying. The FDCNB-MINING algorithm is based on the structure we proposed in [10] for distributed computing environments. Previous studies have assumed a constant network bandwidth, but this assumption fails when there are packet collisions due to many mining tasks being run simultaneously. Therefore, to more closely represent actual situations and to strengthen the environmental structure of CARM, we added a monitor node to observe variable network bandwidths in cloud or distributed computing environments as well as to help the kernel node to select the optimal computing node. Although our results showed that the addition of a monitor node leads to a more stable network, our proposed method exerts only a minimal effect on overall execution efficiency. Through empirical evaluation of six different network bandwidth simulation cases, we found that the FDCNB-MINING algorithm has a good performance in unstable network bandwidth environments and its performance



is also superior to that of the FLR-mining algorithm. The proposed method has been shown to deliver excellent performance in terms of execution efficiency and load balancing.

**Author Contributions:** Conceptualization, C.-C.L. and K.W.L.; Formal analysis, J.-C.C.; Methodology, C.-C.L., W.-C.L., J.-C.C. and W.-Y.C.; Project administration, K.W.L.; Software, J.-C.C., S.-H.C.; Writing—original draft, W.-C.L.; Writing—review & editing, K.W.L.

**Funding:** This work was supported by the Ministry of Science and Technology of Taiwan, R.O.C., under grant No. 107-2221-E-992-085.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Agrawal, R.; Srikant, R. Fast algorithms for mining association rules. In Proceedings of the 20th International Conference Very Large Data Bases, VLDB, Santiago, Chile, 12–15 September 1994; pp. 487–499.
2. Fournier-Viger, P.; Lin, J.C.-W.; Kiran, R.U.; Koh, Y.S.; Thomas, R. A survey of sequential pattern mining. *Data Sci. Pattern Recognit.* **2017**, *1*, 54–77.
3. Gan, W.; Lin, J.C.-W.; Fournier-Viger, P.; Chao, H.C.; Philip, S.Y. HUOPM: High-Utility Occupancy Pattern Mining. *IEEE Trans. Cybern.* **2019**. [CrossRef] [PubMed]
4. Luo, P.; Lu, K.; Shi, Z.; He, Q. Distributed data mining in grid computing environments. *Future Gener. Comput. Syst.* **2007**, *23*, 84–91. [CrossRef]
5. Garlasu, D.; Sandulescu, V.; Halcu, I.; Neculoiu, G.; Grigoriu, O.; Marinescu, M.; Marinescu, V. A big data implementation based on Grid computing. In Proceedings of the Roedunet International Conference (RoEduNet), Sinaia, Romania, 17–19 January 2013; pp. 1–4.
6. Lin, J.C.-W.; Wu, J.M.-T.; Fournier-Viger, P.; Djenouri, Y.; Chen, C.H.; Zhang, Y. A Sanitization Approach to Secure Shared Data in an IoT Environment. *IEEE Access* **2017**, *7*, 25359–25368. [CrossRef]
7. Lin, J.C.-W.; Yang, L.; Fournier-Viger, P.; Hong, T.-P. Mining of skyline patterns by considering both frequent and utility constraints. *Eng. Appl. Artif. Intell.* **2019**, *77*, 229–238. [CrossRef]
8. Lin, K.W.; Lo, Y.-C. A fast parallel algorithm for discovering frequent patterns. In Proceedings of the 2009 IEEE International Conference on Granular Computing, Nanchang, China, 17–19 August 2009; pp. 398–403.
9. Lin, K.W.; Lo, Y.-C. Efficient algorithms for frequent pattern mining in many-task computing environments. *Knowl. Based Syst.* **2013**, *49*, 10–21. [CrossRef]
10. Lin, K.W.; Chung, S.-H. A fast and resource efficient mining algorithm for discovering frequent patterns in distributed computing environments. *Future Gener. Comput. Syst.* **2015**, *52*, 49–58. [CrossRef]
11. Park, J.S.; Chen, M.-S.; Yu, P.S. Using a hash-based method with transaction trimming for mining association rules. Knowledge and Data Engineering. *IEEE Trans. Knowl. Data Eng.* **1997**, *9*, 813–825. [CrossRef]
12. Ozel, S.A.; Guvenir, H. An algorithm for mining association rules using perfect hashing and database pruning. Available online: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.28.6116&rep=rep1&type=pdf> (accessed on 3 May 2019).
13. Agarwal, J.; Singh, A. Frequent item set generation based on transaction hashing. In Proceedings of the 2014 5th International Conference—Confluence the Next Generation Information Technology Summit (Confluence), Noida, India, 25–26 September 2014; pp. 182–187.
14. Brin, S.; Motwani, R.; Ullman, J.D.; Tsur, S. Dynamic itemset counting and implication rules for market basket data. *ACM Sigmod Rec.* **1997**, *26*, 255–264. [CrossRef]
15. Yang, X.Y.; Liu, Z.; Fu, Y. Mapreduce as a programming model for association rules algorithm on hadoop. In Proceedings of the 2010 3rd International Conference on Information Sciences and Interaction Sciences (ICIS), Chengdu, China, 23–25 June 2010; pp. 99–102.
16. Yu, K.-M.; Zhou, J.; Hong, T.-P.; Zhou, J.-L. A load-balanced distributed parallel mining algorithm. *Expert Syst. Appl.* **2010**, *37*, 2459–2464. [CrossRef]
17. Han, J.; Pei, J.; Yin, Y. Mining frequent patterns without candidate generation. In Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, Dallas, TX, USA, 15–18 May 2000; pp. 1–12.
18. Zaïane, O.R.; El-Hajj, M.; Lu, P. Fast parallel association rule mining without candidacy generation. In Proceedings of the 2001 IEEE International Conference on Data Mining, San Jose, CA, USA, 29 November–2 December 2001; pp. 665–668.



19. Shang, X.; Sattler, K.U.; Geist, I. Sql based frequent pattern mining without candidate generation. In Proceedings of the 2004 ACM Symposium on Applied Computing, Nicosia, Cyprus, 14–17 March 2004; pp. 618–619.
20. Schlegel, B.; Gemulla, R.; Lehner, W. Memory-efficient frequent-itemset mining. In Proceedings of the 14th International Conference on Extending Database Technology, ACM, Uppsala, Sweden, 21–24 March 2011; pp. 461–472.
21. Javed, A.; Khokhar, A. Frequent Pattern Mining on Message Passing Multiprocessor Systems. *Distrib. Parallel Database* **2004**, *16*, 321–334. [[CrossRef](#)]
22. Zhou, J.; Yu, K.-M. Tidset-based parallel FP-tree algorithm for the frequent pattern mining problem on PC clusters. In *Advances in Grid and Pervasive Computing*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 18–28.
23. Zhou, J.; Yu, K.-M. Balanced tidset-based parallel FP-tree algorithm for the frequent pattern mining on grid system. In Proceedings of the 2008 Fourth International Conference on Semantics, Knowledge and Grid, Kunming, China, 25–28 May 2008; pp. 103–108.
24. Lai, Y.; ZhongZhi, S. An efficient data mining framework on Hadoop using Java persistence API. In Proceedings of the 2010 IEEE 10th International Conference on Computer and Information Technology (CIT), Bradford, UK, 29 June–1 July 2010; pp. 203–209.
25. Yang, L.; Shi, Z.; Xu, L.D.; Liang, F.; Kirsh, I. DH-TRIE frequent pattern mining on Hadoop using JPA. In Proceedings of the 2011 IEEE International Conference on Granular Computing (GrC), Kaohsiung, Taiwan, 8–10 November 2011; pp. 875–878.
26. Lin, W.-T.; Chu, C.-P. Determining the appropriate number of nodes for fast mining of frequent patterns in distributed computing environments. *Int. J. Parallel Emergent Distrib. Syst.* **2015**, *30*, 380–392. [[CrossRef](#)]
27. Agrawal, R.; Srikant, R. *Quest Synthetic Data Generator*; IBM Almaden Research Center: San Jose, CA, USA, 1994.
28. Brijs, T.; Swinnen, G.; Vanhoof, K.; Wets, G. Using association rules for product assortment decisions: A case study. In Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, 15–18 August 1999; pp. 254–260.
29. Geurts, K.; Wets, G.; Brijs, T.; Vanhoof, K. Profiling of high-frequency accident locations by use of association rules. *Transp. Res. Rec.* **2003**, *1840*, 123–130. [[CrossRef](#)]
30. Goethals, B.; Zaki, M.J. Frequent Itemset Mining Dataset Repository. 2003. Available online: <http://fimi.ua.ac.be/data/> (accessed on 3 May 2019).



© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).