applied sciences

MDPI

Article

Parallel Frequent Subtrees Mining Method by an Effective Edge Division Strategy

Jing Wang [1,*] and Xiongfei Li [2]

[1] School of Media Science, Northeast Normal University, Changchun 130024, China
[2] Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, Changchun 130012, China; xiongfei@jlu.edu.cn
* Correspondence: wangj755@nenu.edu.cn

Abstract: Most data with a complicated structure can be represented by a tree structure. Parallel processing is essential to mining frequent subtrees from massive data in a timely manner. However, only a few algorithms could be transplanted to a parallel framework. A new parallel algorithm is proposed to mine frequent subtrees by grouping strategy (GS) and edge division strategy (EDS). The main idea of GS is dividing edges according to different intervals and then dividing subtrees consisting of the edges in different intervals to their corresponding groups. Besides, the compression stage in mining is optimized by avoiding all candidate subtrees of a compression tree, which reduces the mining time on the nodes. Load balancing can improve the performance of parallel computing. An effective EDS is proposed to achieve load balancing. EDS divides the edges with different frequencies into different intervals reasonably, which directly affects the task amount in each computing node. Experiments demonstrate that the proposed algorithm can implement parallel mining, and it outperforms other compared methods on load balancing and speedup.

Keywords: frequent subtree; parallel algorithms; data partitioning; load balancing

Citation: Wang, J.; Li, X. Parallel Frequent Subtrees Mining Method by an Effective Edge Division Strategy. *Appl. Sci.* **2022**, *12*, 4778. https://doi.org/10.3390/app12094778

Academic Editor: Federico Divina

Received: 13 April 2022
Accepted: 6 May 2022
Published: 9 May 2022

1. Introduction

The era of big data has arrived with the advent of massive data. Semi-structured data [1,2] plays a crucial role in massive data with the non-strict structure feature. Most data with a complicated structure, including semi-structured data, can be represented by a tree structure. Data mining methods are used to find hidden relationships among massive data [3,4]. Frequent subtree mining has become an important field of data mining research [5–7]. It is the process of mining a subtree set from a given data set that satisfies user attention (support or frequent degree). Frequent subtree mining can be applied in many fields. For example, RNA molecule structure can be represented by a tree structure where, in order to obtain information about a new RNA molecule, the new one must be compared to the known RNA structures. The function information of new RNA can be obtained by looking for the same topology [8].

CFMIS (compressed frequent maximal induced subtrees) [9] is an efficient method for the frequent subtree mining we proposed earlier. The CFMIS algorithm can find all frequent induced subtrees without throwing solutions in less time. Parallel frequent subtree mining processing is essential for mining massive volumes of data in a timely manner. MapReduce is an ideal software framework to support distributed computing on large data sets on clusters of computers [10,11]. However, not all algorithms could be transplanted to the MapReduce framework, in fact, only a few algorithms could. Assigning data into appropriate blocks is crucial for paralleling algorithms in MapReduce [12]. In this paper, three parallel CFMIS (PCFMIS) algorithms, PCFMIS1, PCFMIS2 and PCFMIS3, are proposed. PCFMIS1 parallels CFMIS, transplanting CFMIS to MapReduce framework by GS. Furthermore, PCFMIS2 is proposed by optimizing the compression to reduce the

running time on each slave node. Based on PCFMIS1 and PCFMIS2, PCFMIS3 is proposed to achieve load balancing by using an effective EDS.

In summary, the contributions of our work are as follows:

1. A grouping strategy is proposed to achieve effective data partitioning in order to parallel the frequent subtrees mining method.
2. The compression is optimized by avoiding all candidate subtrees to reduce the mining time on nodes.
3. An effective edge division strategy is proposed to achieve load balancing.
4. The proposed algorithm PCFMIS3 is outstanding on load balancing and running time.

The rest of the paper is organized as follows: in Section 2, related work is reviewed; the proposed PCFMIS1, PCFMIS2 and PCFMIS3 are presented in Section 3; in Section 4, experimental results are displayed and discussed; conclusions are made in Section 5.

2. Related Work

With the extensive application of semi-structured data, the research priority of frequent pattern mining has expanded from frequent item set mining [13,14] to frequent subtree mining [15]. L. Wang et al. proposed a novel framework for mining temporal association rules, which mainly represent the temporal relation among numerical attributes [16]. A new structure called a frequent itemsets tree is proposed to avoid generating candidate item sets in mining rules. Building the tree and mining the temporal relation between the frequent itemset proceed simultaneously. V. Huynh et al. proposed an improved version for IPPC tree, called IPPC+, to increase the performance of the tree construction [17]. IPPC+ improves the poor performance of IPPC tree in the case of datasets comprising a large number of distinguishing items but just a small percentage of frequent items. W. Pascal et al. proposed an algorithm mining probabilistic frequent subtrees with polynomial delay, but by replacing each graph with a forest formed by an exponentially large implicit subset of its spanning trees [18]. The algorithm overcomes the drawback that the number of sampled spanning trees must be bounded by a polynomial of the size of the transaction graphs, resulting in less impressive recall even for slightly more complex structures beyond molecular graphs. J. Wang et al. proposed a compression tree sequence (CTS) to construct a compression tree model and saved the information of the original tree in the compression tree. CFMIS [9] was proposed based on CTS to mine frequent maximal induced subtrees. For each iteration, compression could reduce the size of the data set, thus, the traversal speed was faster than that of other algorithms.

Out of memory and computing resources lead massive data mining to difficulties. Parallel data mining can be an effective solution to this problem [19–21]. In recent years, researchers have made some achievements in frequent item mining. S. Shaik et al. presented a scalable parallel algorithm for big data frequent pattern mining [22]. Three key challenges are identified to parallel algorithmic design: load balancing, work partitioning and memory scalability. D. Yan et al. proposed a general-purpose framework PrefixFPM for FPM that is able to fully utilize the CPU cores in a multicore machine [23]. PrefixFPM follows the idea of prefix projection to partition the workloads of PFM into independent tasks by divide and conquer. The state-of-the-art serial algorithms are adapted for mining frequent patterns including subsequences, subtrees, and subgraphs on top of PrefixFPM. D. Yan et al. extend PrefixFPM to provide the complete parallel algorithms by adopting four new algorithms so that a richer set of pattern types are covered, including closed patterns, ordered and unordered tree patterns, and a mixture of path, free tree, and graph patterns [24]. PrefixFPM exposes a unified programming interface to users who can readily customize it to mine their desired patterns. Xun. Y. et al. proposed FiDoop to achieve compressed storage and avoid building conditional pattern bases [25]. Hong T. P. et al. proposed a parallel genetic-fuzzy mining algorithm [26] based on the master–slave architecture to extract both association rules and membership functions from quantitative transactions. C. FB et al. proposed a frequent itemset mining method using sliding windows capable of extracting tendencies from continuous data flows [27]. They develop this method using Big Data technologies, in

particular, using the Spark Streaming framework enabling distribution of the computation along several clusters and thus improving the algorithm speed. Sicard. N. et al. proposed a parallel fuzzy tree mining method (PaFuTM) [28]. In some of their approaches, the level-wise architecture is preserved but tasks are parallelized within each level. All candidates from each frequent subtree are assigned to specific tasks where they are generated and tested against the database.

3. PCFMIS Algorithm

In this section, we provide the definitions for some concepts that will be used in the remainder of the paper. The proposed parallel algorithm will also be explained in detail.

3.1. Prepared Knowledge

3.1.1. Definitions for Concepts

The CFMIS algorithm deals with the tree in which sibling nodes are unordered with labels and the sibling nodes of the same parent node have no repeats. The 'tree' mentioned below is the same tree as the CFMIS processes, assuming that there is no repeat label in a same tree. CFMIS focuses on frequent maximal induced subtree mining.

Definition 1. *Induced subtree. A tree $T' = (V', E', r')$ is an induced tree of $T = (V, E, r)$, denoted as $T' \subset T$ if $V' \subset V$, $E' \subset E$, where V is the set of nodes; E is the set of edges in which $(x, y) \in E$ represents that x is the parent of y; r is the root node.*

Definition 2. *Frequent subtree. $D = \{T_1, T_2, \ldots, T_n\}$ is a tree set, ε is the frequency threshold, $Occ(T_i, T')$ represents whether T' occurs in T_i, if $T' \subset T_i$, then $Occ(T_i, T') = 1$, else $Occ(T_i, T') = 0$. $Frq(T') = \sum_{i=1}^{n} Occ(T_i, T')$. T' is frequent subtree if $Frq(T') \geqslant \varepsilon$.*

Definition 3. *Maximal subtree. F is the frequent subtree set of D. T' and T'' are two frequent subtrees in F, T' maximize T'' (denoted as $T'' \xrightarrow{m} T'$) if and only if $T'' \subset T'$, there is no T''' in F to make $T'' \xrightarrow{m} T'''$happen. T' is called maximal subtree.*

Definition 4. *Compression tree sequence(CTS). CTS is an ordered sequence composed by $node^{pd}$, $CTS = (node_0^{pd_0}, node_1^{pd_1}, \ldots, node_n^{pd_n})$, pd is the index pointing to the parent node for each node except the root node. Let the parent of $node_i^{pd_i}$ be $node_j^{pd_j}$, and then $pd_i = |i - j|$, where $node_0^{pd_0}$is the root node.*

For example, the CTS of the example tree in Figure 1 can be $a^0b^1c^2d^1e^2f^3$. a^0 is the root element. a is the parent element of b, due to the difference between the positions of a and b in the CTS is 1 while the index superscript of b is 1. c is the parent element of f, due to the difference between the positions of c and f in the CTS being 3 while the index superscript of f is 3. The same rule applies to other nodes, so the information about tree structure and all the edges in the tree can be obtained from CTS.

Definition 5. *Tree list length. The number of $node^{pd}$ in CTS is the length of CTS.*

3.1.2. CFMIS Algorithm

A simple review of the CFMIS algorithm is described below (more details in [9]), and it is primarily performed via four stages:

Stage 1. Construct the compression tree model: the original data set is constructed as a compression tree model using CTS;

Stage 2. Cutting edge: this stage is divided into two subprocesses, trim edges and clean-up edges. First, trim the edges for which the edge frequent degree is less than threshold. Then, delete the single node.

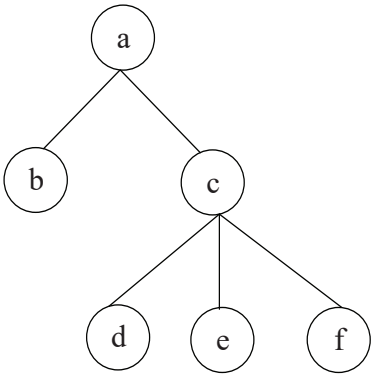


Figure 1. An example tree.

Stage 3. Find frequent subtrees: Compress them according to the descending edge frequent degree to obtain CTSs, and sort CTSs according to the tree list length from shortest to longest. Match the CTS_i with the CTSs following it; if matched (the CTS_i is obtained in another CTS_j), then the frequent degree of the T' represented by CTS_i is incremented by 1.

Stage 4. Maximal Stage: Run the frequent subtree sets maximal processing. The frequent maximal induced subtree set of the original data set are obtained.

3.2. Grouping Strategy

Data partitioning is the premise of the parallel algorithm. An effective data partitioning method can greatly reduce the data communication between different slave nodes, thereby reducing the parallel computing time. Based on the above findings, the grouping strategy (GS) of PCFMIS used to divide all the trees in original data set D into different groups is described below:

(1) If given m slave nodes, we should divide all the trees in D into m groups, and the number of the group is denoted as $Gnum$. For the edge (x,y) in D, divide (x,y) into the set $A = \{A_k\}, k \in [1, m]$.

(2) For $T_i \in D$, all the edges in T_i may belong to several A_k according to (1). Take the minimum of these k as the $Gnum$ of T_i, $Gnum$ = min k, so T_i is put into the group which $Gnum$ = min k. Then, for T_i, cut the edges which belong to A_{mink}. Some new trees $T_i^{'}$ will appear after cutting edges.

(3) For these new trees $T_i^{'}$, repeat (2) until no new trees are produced.

(4) For $T_j \in D$, repeat (2) and (3).

(5) All the trees in original data set D are divided into different groups, and the different groups will be put in different slave nodes.

Definition 6. *Related Tree. Given a tree $T = (V, E, r)$, for $\forall(x, y) \in E$, there is $(x, y) \in A_k$. min k is the minimum of k. When $(x, y) \in A_{mink}$, T is a related tree of (x, y), denoted as $(x, y) \to T$. The set of $t(x, y) \to T$ is denoted as $\Gamma(x, y)$.*

Property 1. *According to grouping strategy, a tree in original data set only belongs to one group.*

Property 2. *According to grouping strategy, if $(x, y) \in A_k$, all $\Gamma(x, y)$ will be put into the group which $Gnum = k$.*

According to Property 1, the trees in the original data set are grouped into different groups, which can reduce the scale of the data each slave node needs to process. Although in (2), new trees may be generated during the grouping process, the new trees are trimmed and less complex than the original tree.

We can conclude from Property 2, for $(x, y) \in A_k$, all the induced subtrees of $\Gamma(x, y)$ containing (x, y) can be found in the group in which $Gnum = k$. So frequent induced

subtrees of $\Gamma(x,y)$ containing (x,y) can be found only in group k instead of the whole original data set. This avoids communication between groups and realizes real parallelism.

Definition 7. *Frequent Edge Degree.* $D = \{T_1, T_2, ..., T_n\}$, *the frequent edge degree of* (x,y) *is the number of times* (x,y) *appears in* D, *denoted as* $EFrq(x,y)$.

How to divide the edge (x,y) into set $\{A_k\}$ in (1) directly affects the $Gnum$ of $\Gamma(x,y)$. One feasible method is described below:

Given an evenly divided interval $A_{div} = (a_1, ..., a_i, ..., a_m)$, $a_1 > ... > a_i > ... > a_m$, $a_1 = EFrq_{min}(x,y)$, $a_m = EFrq_{max}(x,y)$. If $a_{i+1} > EFrq(x,y) \geq a_i$, the edge (x,y) is divided into A_i, then $\Gamma(x,y)$ will be put into the group in which $Gnum = k$. For example, in Figure 2, suppose that the number of slave nodes is 3. Divide the trees T_0, T_{00} into 3 groups (G_1, G_2, G_3). Given dividing interval $A_{div} = (a_1, a_2, a_3)$, $a_1 > a_2 > a_3$. $EFrq(a,b)$, $EFrq(c,d)$, $EFrq(d,f) \geq a_1$, $a_1 > EFeq(h,d), EFrq(d,e) \geq a_2, a2 > EFrq(a,c) \geq a3$, so $(a,b)(c,d)(d,f)$ are divided into A_1; $(h,d)(d,e)$ are divided into A_2; (a,c) is divided into A_3. Take T_0 in Figure 2 as an example. For A_3, $\Gamma(a,c)$ (in Figure 2) is divided into $G3$. Then delete (a,c) from A_3 and cut (a,c). For A_2, $\Gamma(a,c)$ (Ti in Figure 2) are divided into $G2$. Then delete (d,e) from A_2 and cut (d,e). For A_1, $\Gamma(ab)$ (T_3 in Figure 2), $\Gamma(cd)$ ($T4$ in Figure 2) and $\Gamma(df)$ (T_4 in Figure 2) are divided into G_1. T_{01}, T_{02} are obtained after applying GS on T_{00}.

By using the above grouping strategy on the original data set and applying CFMIS algorithm in each group, the CFMIS algorithm is implemented in parallel. The new parallel algorithm is called PCFMIS1. However, the experiment results shown in Section 4.2 indicate that the parallel computing time and speedup did not achieve the desired results. Further improvements on PCFMIS1 will be discussed below.

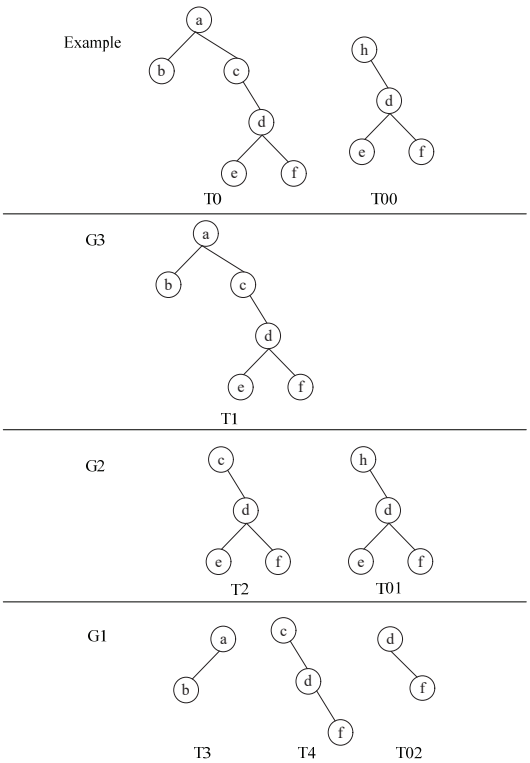


Figure 2. An example of weight tree dividing.

3.3. Improvements on Parallel Algorithm

3.3.1. Optimized Compression in CFMIS

Stage 3 is the central step of the CFMIS algorithm, and it also takes most of the time in CFMIS. The subsequence processing of Stage 3 must find all subtrees of a compression tree to determine whether each of them is frequent. In fact, it consumes up to 70% of the

execution time in this stage. Optimizing Stage 3 can greatly reduce time consumption of the algorithm, particularly when the data size is large. In this section, CFMIS is refined by optimizing Stage 3.

According to Property 2, for any edges $(x, y) \in A_k$, all the induced subtrees of $\gamma(x, y)$ containing (x, y) can be found in the group k, so frequent induced subtrees of $\gamma(x, y)$ can also be found. For other edges $(p, q) \notin A_k, (p, q) \in A_f$, all the induced subtrees of $\gamma(p, q)$ containing (p, q) can be found in the group f. For the reasons above, we only need to find all the frequent induced subtrees of $\gamma(x, y)$ which contain (x, y) instead of all the frequent induced subtrees in group k. Other frequent induced subtrees could be found in other groups. If a CTS in group k does not contain (x, y), this CTS does not have to match with the CTSs following it. That is, this improvement in Stage 3 avoids finding all subtrees of a compression tree to determine whether each of them is frequent. The improvement in compression in groups makes the running time shorter. The improved parallel algorithm is called PCFMIS2. Although the time has been reduced, the load on slave nodes is not balanced.

3.3.2. Load Balancing

An effective edge division strategy (EDS) is proposed in this section. If $(x, y) \epsilon A_k$, $\Gamma(x, y)$ will be put into the group k. The division of edges in the original data set affects the load of the slave nodes. Take the frequent edge degree as the basis for the edge division strategy. Abstract edge division strategy as a math problem, it can be described as below: Suppose that there are w different edges in the original data set, and their frequent edge degree is denoted as $EFrq(i)$, $1 \leqslant i \leqslant w$. Record $EFrq(i)$ in the array $x[i]$. Divide different edges from the original data set into the set $\{A_k\}$, ensuring that the sum of the frequent edge degree in each A_k is approximately equal. The improved parallel algorithm is called PCFMIS3.

The steps of EDS are described below:

1. Get the mean of array X, $u = \frac{\sum X[i]}{k}$.
2. Traverse the array X, if $X[i] > u$, then it is assigned separately to a group. Suppose there are s groups like that.
3. Now, the problem is translated into such a problem: divide $(w-s)$ different edges into $(k-s)$ sets, ensuring that the sum of the frequent edge degree in each group is approximately equal.
4. For the rest $X[j]$, $1 \leqslant j \leqslant w-s$, get the mean of array X, $u' = \frac{\sum X[j]}{k-s}$.
5. Translate the step (3) and (4) problem to 0–1 Knapsack Problem in order to solve it.

An example of the EDS method is given here to show how it works: suppose that the frequent edge degree of the original data set is 50, 60, 80, 100, 150, 200, 400, 1000, and $k = 3$. The mean value $u = \frac{50+60+80+100+150+200+400+1000}{3} = 680$. As $1000 > 680$, 1000 is assigned separately to A_1. Now, the problem is translated into such a problem: Divide the rest of the edges into two sets, ensuring that the sum of the frequent edge degree in each set is approximately equal. Get the mean value $u = \frac{50+60+80+100+150+200+400}{2} = 520$. Translate the problem to 0–1 Knapsack Problem in order to solve it. The weights of these items are 50, 60, 80, 100, 150, 200, 400 and the backpack capacity is 520. The answer is that 50, 60, 400 make the total value biggest in the backpack. 50, 60, 400 are assigned to A_2, and the rest are assigned to A_3.

PCFMIS3 solved the problem of load balancing and introduced the optimized compression. The flowchart of PCFMIS3 is shown in Figure 3. Three steps including two Map-reduce operations are completed during parallel computation.

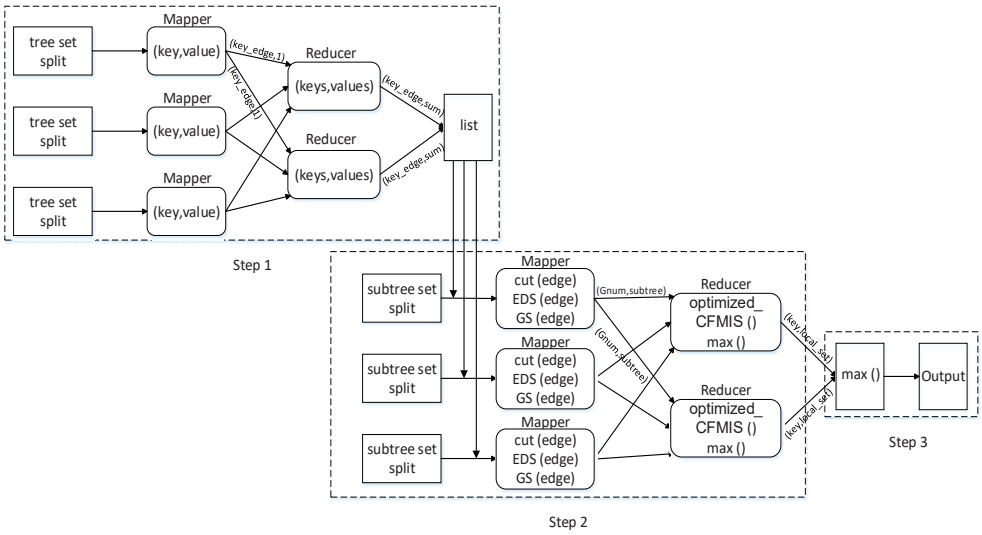


Figure 3. Parallel algorithm PCFMIS3 flowchart.

Step1: Calculate edge frequent degree of each edge in original tree set. In Map 1 (Algorithm 1), record each occurrence of each edge in a tree set split, which is the input of Reduce 1 (Algorithm 2). In Reduce 1, the edge frequent degree of each edge is counted out.

Step 2: Cut edge and find frequent subtree. In Map 2 (Algorithm 3), trim the edges for which the edge frequent degree is less than according to the list; divide the edges into different sets according to EDS; divide the subtrees into different groups according to GS. Then which group the subtree is divided to is the input of Reduce (Algorithm 4). In Reduce 2, find frequent subtrees in each group. The maximal frequent subtrees of each group could be obtained.

Step 3: Find the maximal subtrees of original tree set. In this step, run the frequent subtree sets maximal processing to obtain the final maximal frequent subtrees.

Algorithm 1: Map 1 $(key, value)$

Input: $//key$: document name
$//value$: subtree set
Output: $(key\_edge, 1)$ // the key_edge is the edge in the subtree set.

```
for each edge in subtree set do
    Emit (key_edge,1)
end
```

Algorithm 2: Reduce 1 $(keys, values)$

Input: $//key$: the set of the same edge
$//values$: the list of the edge value
Output: (key_edge, sum) // sum is the frequency of the edge

```
sum=0;
for each edge value in values do
    sum=sum + value; Emit (key_edge, sum)
end
```

Algorithm 3: Map 2 $(key, value)$

Input: $//split$: subtree set split
$//list$: the list of the edges
Output: key // key is the $Gnum$
$value$ // value is the subtree in different Gnum

```
for each edge in list do
    subtree_group = cut(edge)
    // cut edges that do not meet the threshold requirements, group is the
     remaining subtree set
end
for each edge in subtree_group do
    A(edge)=EDS(edge)
    // EDS is the edge division strategy; A(edge) is the division set
end
for each subtree in subtree_group do
    Gnum(subtree)=GS(subtree)// GS is the grouping strategy
    value=subtree
    key=Gnum
    Emit(key, value)
end
```

Algorithm 4: Reduce 2 $(key, local\_set)$

Input: // $Gnum$: the group number
// $Gnum\_subtrees$: all the subtrees which group number is $Gnum$
Output: key // null
$local\_set$ // the frequent subtree set which belongs to this group

```
frequent_list=get_edges(Gnum)
// get_edges(Gnum) is used to get the list of the frequent edges of this group
for the edges in frequent_list do
    v=optimized_CFMIS(Gnum_subtrees)
    // optimized_CFMIS is used to find the frequent subtrees in this group
    local_set=max(v)
    // max(v) is used to maximize the frequent subtrees
    Emit (key, local_set)
end
```

4. Experiments and Results

Paralleling the CFMIS algorithm only affects execution time, and it has no effect on the accuracy of the calculation results. The corresponding validity of the algorithm could consult to [9]. The number of groups in the GS is the number of the slave nodes in this paper. In order to prove that EDS is an effective method to solve load balancing, the load balancing tests are compared between PCFMIS1 and PCFMIS3. The experiments on computational time and speedup are done among PCFMIS1, PCFMIS2, PCFMIS3 and PaFuTM [28].

4.1. Experimental Environment

All of the experiments were conducted on a PC cluster connected with 100M Ethernet. Each PC was equipped with a 3.20 GHz Intel Core i5 and 4 GB main memory, running the Centos6.6 operating system. The version of the platform is Hadoop 2.6.2. The system configuration is shown in Figure 4. The synthetic data set used in this paper was generated by tree generator using the method in [29]. Parameters of the synthetic dataset are set as follows: f = 10 (f represents fan-out), d = 10 (d represents the depth of the tree), n = 100 (n represents the number of labels), m = 100 (m represents the number of tree nodes), and t

(t represents the number of trees). While t = 50,000, the data set is denoted as D5; t = 100,000, denoted as D10; t = 200,000, denoted as D20; t = 500,000, denoted as D50; t = 1,000,000, denoted as D100; t = 2,000,000, denoted as D200. The real data set was obtained from CSLOGS data set, which is from a month-log of the data of the Rensselaer Polytechnic Institute's web site. CSLOGS10 contains 100,000 trees and CSLOGS100 contains 1,000,000 trees. The support thresholds for D5, D10, D20 and CSLOGS10 are 0.01 and 0.05. For D50, D100, D200 and CSLOGS100, the support thresholds are 0.01 and 0.001.

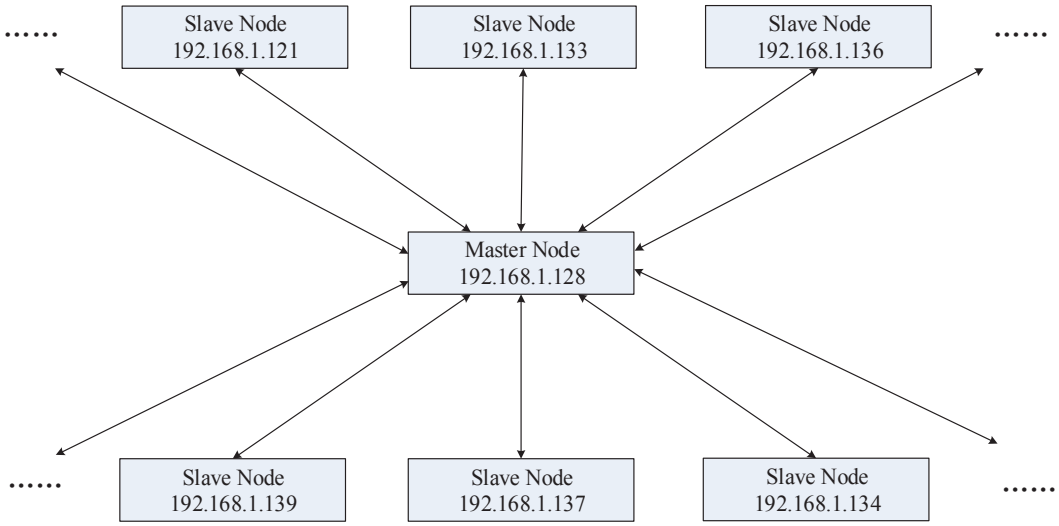


Figure 4. System configuration diagram.

4.2. Experiments and Analysis

During the experiments, it was found that for the data sets with large data volume, time cost increases dramatically when the number of nodes is too small. The reason is that the computation of the tree structure needs to constantly operate the stack. If the JVM stack is not enough, it can only keep replacing the stack. In this paper, experiments have been done on a small number of nodes for small data sets and on a large number of nodes for large data sets.

4.2.1. Comparison of Load Balancing

We performed the load balancing evaluation on both real data set CSLOGS10 and synthetic data set D50. For CSLOGS10, the number of nodes is set to 1–5; for D50, the number is set to 1–8.

Table 1 shows the amount of subtrees and computing time in different groups of CSLOGS10 in Reduce 2 while the total number of nodes are 3, 4, 5 and the support threshold is 0.01. The subtree amounts in each group divided by PCFMIS1 and PCFMIS3 are shown in Table 1. For PCFMIS1, when the number of nodes is 3, the amount of subtrees in the most loaded group (*Gnum* = 3) is 3.66 times that of the least loaded group (*Gnum* = 1); when the number of nodes is 4 and 5, the ratio is 4.54 and 6.96. For PCFMIS3, when the number of nodes is 3, the amount of subtrees in the most loaded group (*Gnum* = 1) is 1.32 times that of the least loaded group (*Gnum* = 2); when the number of nodes is 4 and 5, the ratio is 1.30 and 1.31. For the computing time, when the number of nodes is 3 in PCFMIS1, the longest computing time (*Gnum* = 3) is 3.31 times that of the shortest computing time (*Gnum* = 1). The corresponding amount ratio and time ratio are shown in Table 2. The closer the ratio is to 1, the better the effect of the load balancing. The same load balancing test is done on D50 (the support threshold is 0.01) which is shown in Table 3, and the corresponding amount ratio and time ratio are shown in Table 4. The experiments indicate that the efficient division of edges in the original data set affects the load of the slave nodes. The proposed EDS can achieve load balancing.

Table 1. The amount of subtrees and computing time in different *Gnum* groups of CSLOGS10.

Number of Nodes	*Gnum*	PCFMIS1		PCFMIS3	
		Amount	Time (s)	Amount	Time (s)
3	1	18,395	22.73	53,466	40.01
	2	41,369	41.65	40,564	38.37
	3	67,365	75.34	50,084	37.72
4	1	15,463	18.39	49,826	28.58
	2	52,369	27.76	49,819	29.49
	3	67,428	55.56	38,462	25.28
	4	70,254	42.84	38,352	25.91
5	1	10,630	13.27	34,657	21.46
	2	32,745	21.36	38,724	23.11
	3	62,156	41.2	45,396	22.31
	4	47,563	32.58	42,683	22.12
	5	73,947	37.25	40,567	21.89

Table 2. The amount ratio and time ratio in different *Gnum* groups of CSLOGS10.

Number of Nodes	PCFMIS1		PCFMIS3	
	Amount	Time (s)	Amount	Time (s)
3	3.66	3.31	1.32	1.06
4	4.54	3.02	1.3	1.17
5	6.96	3.1	1.31	1.08

Table 3. The amount of subtrees and computing time in different *Gnum* groups of D50.

Number of Nodes	*Gnum*	PCFMIS1		PCFMIS3	
		Amount	Time (s)	Amount	Time (s)
4	1	96,386	82.36	134,526	118.63
	2	113,695	113.67	101,637	113.6
	3	155,692	136.98	116,985	123.64
	4	277,258	187.63	114,329	117.25
6	1	75,264	52.69	96,452	86.66
	2	65,897	53.76	93,658	84.32
	3	193,648	122.25	107,612	83.59
	4	85,638	85.96	84,369	83.12
	5	84,567	93.21	83,695	83.72
	6	156,942	110.67	92,418	84.87
8	1	87,526	67.77	54,960	63.89
	2	40,236	45.63	76,147	62.14
	3	73,658	62.37	58,426	63.45
	4	54,269	43.73	66,545	64.77
	5	86,352	58.23	65,471	62.95
	6	125,621	87.12	64,774	64.01
	7	97,634	81.69	70,425	63.52
	8	177,563	93.28	69,998	62.59

Table 4. The amount ratio and time ratio in different *Gnum* groups of D50.

Number of Nodes	PCFMIS1		PCFMIS3	
	Amount	Time (s)	Amount	Time (s)
3	2.88	2.29	1.32	1.09
4	2.94	2.32	1.29	1.04
5	4.41	2.13	1.39	1.04

4.2.2. Comparison of Running Time and Speedup

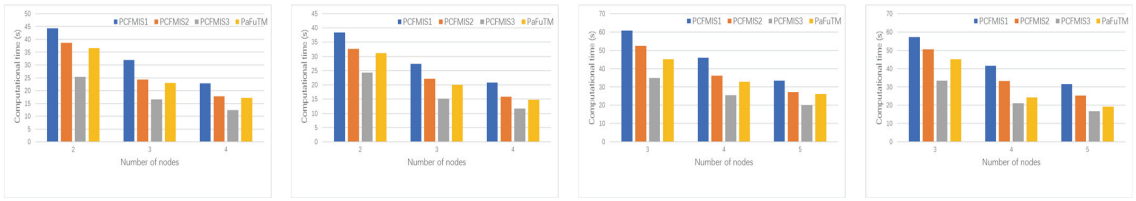
The speedup is used to measure the parallelization performance of parallel systems or programs. It is the ratio of the time that the same task runs in uniprocessor and parallel processor systems. The speedup is defined by the following formula [30]:

$$S_p = \frac{T_1}{T_p} \tag{1}$$

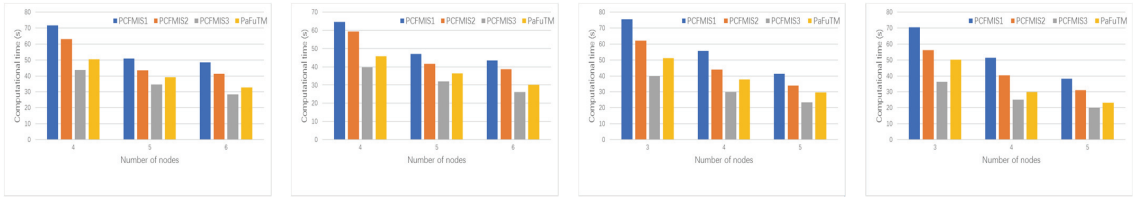
where p is the number of nodes, T_1 is the execution time on a single node, and T_p is the execution time on p nodes.

When $S_p = p$, the speedup is a linear speedup. It is the ideal speedup that the parallel algorithm tries to achieve. However, the linear speedup is difficult to achieve because the communication cost increases with the increasing number of cores.

Figure 5 shows the computational time of four algorithms for four data sets on a small number of nodes. Figure 6 shows the computational time of four algorithms for four data sets on large number of nodes. As the number of the nodes increases, the computational time of the four parallel methods becomes shorter. Due to the use of optimized compression, PCFMIS2 has less computational time than PCFMIS1. PCFMIS3 performs best, saving half the time compared to PCFMIS1. It indicates that the improvements on optimized compression and load balancing are effective. In particular, the EDS strategy proposed by PCFMIS3 divides the edges more rationally to provide a basis for grouping. As PaFuTM has to find all the candidate subtrees and duplicate redundant subtrees exist among nodes, PaFuTM has longer computational time than PCFMIS3.

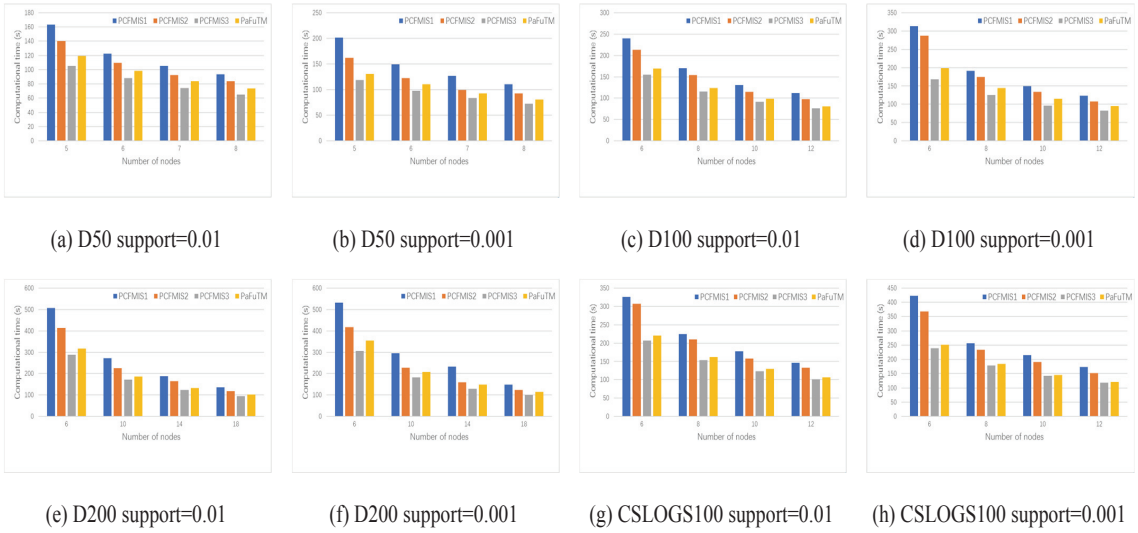


(a) D5 support=0.01 (b) D5 support=0.05 (c) D10 support=0.01 (d) D10 support=0.05



(e) D20 support=0.01 (f) D20 support=0.05 (g) CSLOGS10 support=0.01 (h) CSLOGS10 support=0.05

Figure 5. Computational time of four algorithms for four data sets on a small number of nodes.

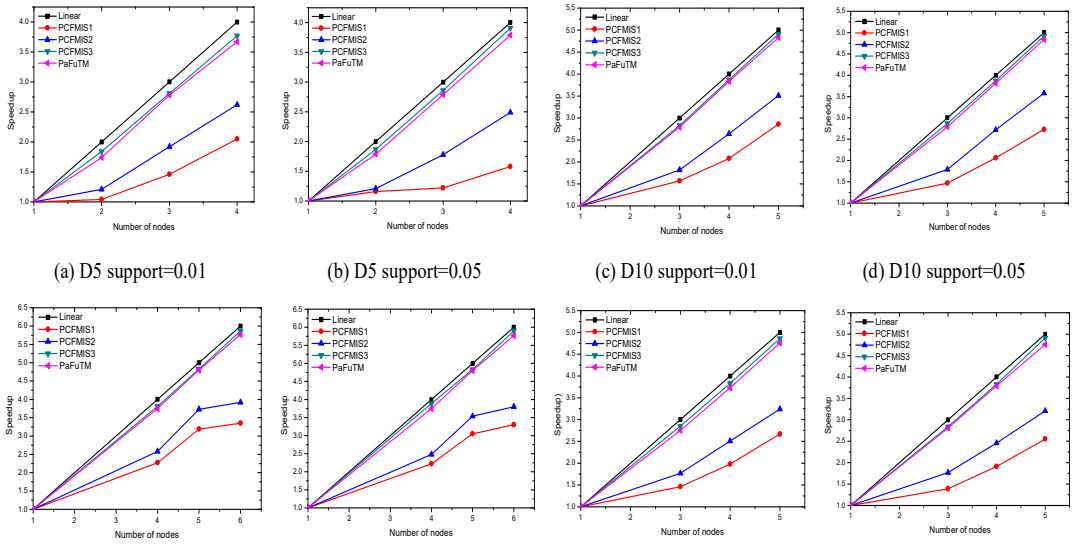


(a) D50 support=0.01 (b) D50 support=0.001 (c) D100 support=0.01 (d) D100 support=0.001

(e) D200 support=0.01 (f) D200 support=0.001 (g) CSLOGS100 support=0.01 (h) CSLOGS100 support=0.001

Figure 6. Computational time of four algorithms for four data sets on a large number of nodes.

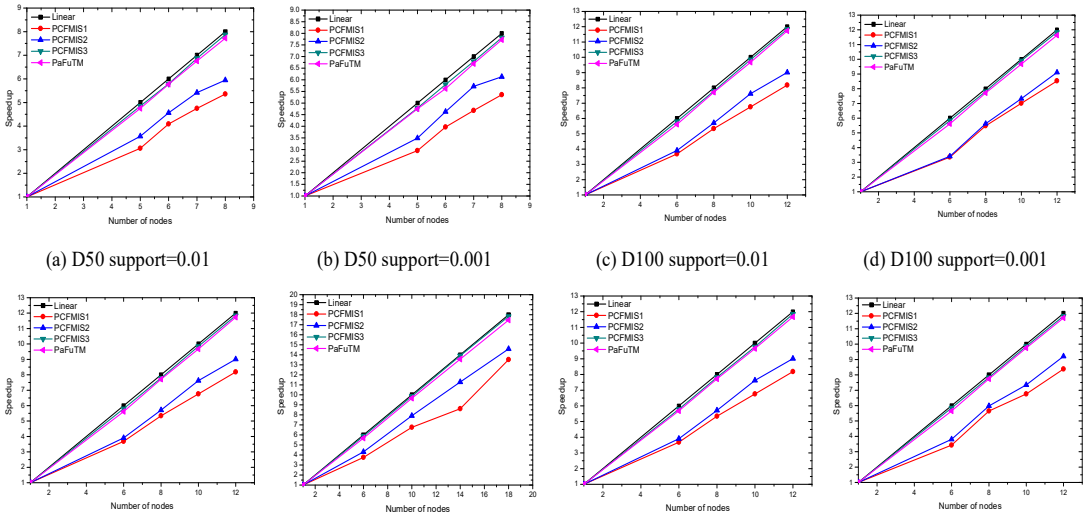
Figure 7 shows the speedups on four data sets with a small number of nodes. Figure 8 shows the speedups on another four data sets with a large number of nodes. As the results show, the speedup of PCFMIS1 performs worst on all data sets. PCFMIS2 performs better than PCFMIS1. The speedups of PCFMIS3 and PaFuTM are closest to the linear one, but PCFMIS3 performs better. As the number of nodes increases, the speedup keeps growing. However, in Figure 7e,f, when the nodes increase from 5 to 6, the speedup of PCFMIS1 and PCFMIS2 grow slowly. There is a clear salient point. This also appears in Figure 8a,b,f. PCFMIS3 maintains a steady growth.



(a) D5 support=0.01 (b) D5 support=0.05 (c) D10 support=0.01 (d) D10 support=0.05

(e) D20 support=0.01 (f) D20 support=0.05 (g) CSLOGS10 support=0.01 (h) CSLOGS10 support=0.05

Figure 7. The speedups of four algorithms for four data sets on a small number of nodes.



(a) D50 support=0.01 (b) D50 support=0.001 (c) D100 support=0.01 (d) D100 support=0.001

(e) D200 support=0.01 (f) D200 support=0.001 (g) CSLOGS100 support=0.01 (h) CSLOGS100 support=0.001

Figure 8. The speedups of four algorithms for four data sets on a large number of nodes.

5. Conclusions

A paralleling algorithm PCFMIS3 with GS and EDS is proposed in MapReduce framework to parallel CFMIS to mine frequent subtrees efficiently. The GS method divides the subtrees into different groups, which solves the data division problem in parallel computing. It avoids inter-group communication while mining frequent subtrees in each group to reduce parallel computing time. In PCFMIS3, load balancing is achieved by using EDS. Additionally, the compression stage in mining frequent subtrees is optimized by avoiding all candidate subtrees of a compression tree, which reduces the calculation time of nodes. Experiments demonstrate that the PCFMIS3 algorithm performs best on the comparison of load balancing and running time on both the real data set and synthetic data set. The maximum load is 1.3 times the minimum load, while it is up to 7 times without EDS. The time ratio of PCFMIS3 is only about 1.1, while the time ratio exceeds 2.0 without EDS. PCFMIS3 also performs best on different support values, saving half the computational time compared to PCFMIS1. The PCFMIS3 achieves the optimal speedup which is closest to the linear one on both small and large number of nodes.

Serial computing technology is difficult to meet the needs of massive data processing. Parallel computing can take advantage of multi-node computing resources to reduce problem resolution time. The proposed GS and EDS methods solve the two important issues of data partitioning and load balancing in parallel computing. To apply GS and EDS method in the mining of other frequent item sets is our future work.

Author Contributions: Conceptualization, J.W. and X.L.; methodology, J.W. and X.L.; software, J.W.; validation, J.W.; formal analysis, X.L.; writing—original draft preparation, J.W.; visualization, J.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Technology Development Plan of Jilin Province (Jing Wang 2022) and the Fundamental Research Funds for the Central Universities.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Tekli, G. A survey on semi-structured web data manipulations by non-expert users. *Comput. Sci. Rev.* **2021**, *40*, 100367. [CrossRef]
2. Du, Y. Massive Semi-structured Data Platform Based on Elasticsearch and MongoDB. In *Signal and Information Processing, Networking and Computers*; Springer: Singapore, 2021; pp. 877–884.
3. Hong, T.P.; Lin, C.Y.; Huang, W.M.; Li, S.M.; Wang, S.L.; Lin, J.C. A One-Phase Tree-Structure Method to Mine High Temporal Fuzzy Utility Itemsets. *Appl. Sci.* **2022**, *12*, 2821. [CrossRef]
4. Lee, C.; Baek, Y.; Lin, J.C.; Truong, T.; Yun, U. Advanced uncertainty based approach for discovering erasable product patterns. *Knowl.-Based Syst.* **2022**, *24*, 108134. [CrossRef]
5. Black, F.; Drellich, E.; Tymoczko, J. Valid Plane Trees: Combinatorial Models for RNA Secondary Structures with Watson–Crick Base Pairs. *SIAM J. Discret. Math.* **2017**, *31*, 2586–2602. [CrossRef]
6. Welke, P. *Efficient Frequent Subtree Mining Beyond Forests*; IOS Press: Amsterdam, The Netherlands, 2020.
7. Li, H.; Lee, J.; Mi, H.; Yin, M. Finding good subtrees for constraint optimization problems using frequent pattern mining. *Proc. AAAI Conf. Artif. Intell.* **2020**, *34*, 1577–1584. [CrossRef]
8. Banchhor, C.; Srinivasu, N. Integrating Cuckoo search-Grey wolf optimization and Correlative Naive Bayes classifier with Map Reduce model for big data classification. *Data Knowl. Eng.* **2020**, *127*, 101788. [CrossRef]
9. Wang, J.; Liu, Z.; Li, W.; Li, X. Research on a frequent maximal induced subtrees mining method based on the compression tree sequence. *Expert Syst. Appl.* **2015**, *42*, 94–100. [CrossRef]
10. Neshatpour, K.; Malik, M.; Sasan, A.; Rafatirad, S.; Mohsenin, T.; Ghasemzadeh, H.; Homayoun, H. Energy-efficient acceleration of MapReduce applications using FPGAs. *J. Parallel Distrib. Comput.* **2018**, *119*, 1–17. [CrossRef]
11. Es-Sabery, F.; Hair, A. Big data solutions proposed for cluster computing systems challenges: A survey. In Proceedings of the 3rd International Conference on Networking, Information Systems & Security, Marrakech, Morocco, 31 March–2 April 2020; pp. 1–7.
12. Chen, F.; Deng, P.; Wan, J.; Zhang, D.; Vasilakos, A.V.; Rong, X. Data mining for the internet of things: Literature review and challenges. *Int. J. Distrib. Sens. Netw.* **2015**, *11*, 431047. [CrossRef]
13. Vo B.; Le T.; Coenen, F.; Hong, T.P. Mining frequent itemsets using the N-list and subsume concepts. *Int. J. Mach. Learn. Cybern.* **2016**, *7*, 253–265. [CrossRef]
14. Mao, Y.; Geng, J.; Deborah, S.M.; Yaser, A.N.; Zhang, C.; Deng, X.; Chen, Z. PFIMD: A parallel MapReduce-based algorithm for frequent itemset mining. *Multimed. Syst.* **2021**, *27*, 709–722.
15. Welke, P.; Seiffarth, F.; Kamp, M.; Wrobel, S. HOPS: Probabilistic subtree mining for small and large graphs. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Virtual Event, CA, USA, 23–27 August 2020; pp. 1275–1284.
16. Wang, L.; Meng, J.; Xu, P.; Peng, K. Mining temporal association rules with frequent itemsets tree. *Appl. Soft Comput.* **2018**, *62*, 817–829. [CrossRef]
17. Huynh, V.; Küng, J. Higher Performance IPPC+ Tree for Parallel Incremental Frequent Itemsets Mining. In Proceedings of the 5th International Conference FDSE, Ho Chi Minh City, Vietnam, 28–30 November 2018.
18. Pascal, W.; Tamas, H.; Stefan, W. Probabilistic and exact frequent subtree mining in graphs beyond forests. *Mach. Learn.* **2019**, *108*, 1137–1164.
19. Wang, C.S.; Chang, J.Y. MISFP-Growth: Hadoop-Based Frequent Pattern Mining with Multiple Item Support. *Appl. Sci.* **2019**, *9*, 2075. [CrossRef]
20. Upadhyay, N.M.; Singh, R.S.; Dwivedi, S.P. Prediction of multicore CPU performance through parallel data mining on public datasets. *Displays* **2022**, *71*, 102112. [CrossRef]
21. Hashem, I.A.; Yaqoob, I.; Anuar, N.B.; Mokhtar, S.; Gani, A.; Khan, S.U. The rise of big data on cloud computing: Review and open research issues. *Inf. Syst.* **2015**, *47*, 98–115. [CrossRef]
22. Shaik, S.; Subhani, S.; Devarakonda, N.; Nagamani, C. Parallel Computing Algorithms for Big data frequent pattern mining. In Proceedings of the International Conference on Computational Intelligence & Data Engineering ICCIDE, Vijayawada, India, 14–15 July 2017.
23. Yan, D.; Qu, W.; Guo, G.; Wang, X. PrefixFPM: A Parallel Framework for General-Purpose Frequent Pattern Mining. In Proceedings of the IEEE 36th International Conference on Data Engineering (ICDE), Dallas, TX, USA, 20–24 April 2020.
24. Yan, D.; Qu, W.; Guo, G.; Wang, X.; Zhou, Y. PrefixFPM: A parallel framework for general-purpose mining of frequent and closed patterns. *VLDB J.* **2021**, *31*, 253–286. [CrossRef]
25. Xun, Y.; Zhang, J.; Qin, X. Fidoop: Parallel mining of frequent itemsets using mapreduce. *IEEE Trans. Syst. Man Cybern. Syst.* **2016**, *46*, 313–325. [CrossRef]
26. Hong, T.P.; Lee, Y.C.; Wu, M.T. An effective parallel approach for genetic-fuzzy data mining. *Expert Syst. Appl.* **2014**, *41*, 655–662. [CrossRef]
27. Fern, ez-Basso, C.; Francisco-Agra, A.J.; Martin-Bautista, M.J.; Ruiz, M.D. Finding tendencies in streaming data using Big Data frequent itemset mining. *Knowl.-Based Syst.* **2019**, *163*, 666–674.
28. Sicard, N.; Laurent, A.; López, F.D.; Flores, P.M. Towards multi-core parallel fuzzy tree mining. In Proceedings of the 2010 IEEE International Conference on Fuzzy Systems (FUZZ), Barcelona, Spain, 18–23 July 2010; pp. 1–7.

29. Zaki, M.J. Efficiently Mining Frequent Trees in a Forest: Algorithms and Applications. *IEEE Trans. Knowl. Data Eng.* **2005**, *17*, 1021–1035. [CrossRef]
30. Zhang, J.; Wong, J.S.; Li, T.; Pan, Y. A comparison of parallel large-scale knowledge acquisition using rough set theory on different MapReduce runtime systems. *Int. J. Approx. Reason.* **2014**, *55*, 896–907. [CrossRef]