Parallel Eclat for Opportunistic Mining of Frequent Itemsets

Junqiang Liu[1(✉)], Yongsheng Wu[1], Qingfeng Zhou[1], Benjamin C.M. Fung[2], Fanghui Chen[1], and Binxiao Yu[1]

[1] School of Information and Electronic Engineering, Zhejiang Gongshang University, Hangzhou 310018, China
jjliu@alumni.sfu.ca
[2] School of Information Studies, McGill University, Montreal, QC, Canada
ben.fung@mcgill.ca

Abstract. Mining frequent itemsets is an essential data mining problem. As the big data era comes, the size of databases is becoming so large that traditional algorithms will not scale well. An approach to the issue is to parallelize the mining algorithm, which however is a challenge that has not been well addressed yet. In this paper, we propose a MapReduce-based algorithm, Peclat, that parallelizes the vertical mining algorithm, Eclat, with three improvements. First, Peclat proposes a hybrid vertical data format to represent the data, which saves both space and time in the mining process. Second, Peclat adopts the pruning technique from the Apriori algorithm to improve efficiency of breadth-first search. Third, Peclat employs an ordering of itemsets that helps balancing the workloads. Extensive experiments demonstrate that Peclat outperforms the existing MapReduce-based algorithms significantly.

Keywords: Big data · Data mining · Frequent itemsets · Vertical format · MapReduce · Parallel algorithms

1 Introduction

Discovering frequent itemsets is always an essential problem in data mining research and database applications. This problem is formulated as follows: Given a transaction database, find all frequent itemsets where a frequent itemset is one that occurs in at least a user-specified number or percentage of transactions, that is, its support is no less than the threshold.

The best-known breadth-first algorithm is Apriori [2] by Agrawal et al. Apriori finds frequent itemsets in a level-wise manner. At each level k, it scans the database once to compute the supports of the k-itemsets whose subsets at level $k-1$ are all frequent. Therefore, Apriori scans the database as many passes as the largest size of frequent itemsets, which incurs high I/O overhead. The FP-growth algorithm [9] by Han et al. is the best-known depth-first algorithm. It uses a novel frequent pattern tree (FP-tree) held in memory to represent the

Q. Chen et al. (Eds.): DEXA 2015, Part I, LNCS 9261, pp. 401–415, 2015.
DOI: 10.1007/978-3-319-22849-5_27

database in a compressed form, and mines all frequent itemsets by recursively projecting the FP-trees. FP-growth as well as any depth-first algorithm does not work as efficiently as supposed when faced with huge and sparse databases.

While most previous works inspired by Apriori [2] and FP-growth [9] employ a horizontal data format, a number of algorithms [6,15,16,19,20] use vertical data format. In a vertical data format, each item is associated with its tidset, the set of identifiers of transactions that contain the item. The best-known vertical mining algorithm is Eclat [19] by Zaki et al., which uses the tidset format. But, when the database is large and dense, Eclat suffers from huge intermediate storage cost. Thus, Zaki proposed the diffset format [20] for dense databases. However, large databases cannot be simply categorized as sparse or dense, with which neither single vertical format can scale well.

From the above analysis, we can observe that sequential mining algorithms do not scale well when databases become huge. Thus, some works [1,3,17,21] proposed parallel algorithms that make use of computing paradigms in the form of multi-processors. But, such parallelization paradigms have issues in balancing workloads and in recovering from hardware or communication failures.

Therefore, Google proposed the MapReduce paradigm [5]. Since then, a few parallel algorithms [4,7,8,10–14,18] based on MapReduce have been proposed for mining frequent itemsets, among which the DPC [13] algorithm is the best derived from Apriori, and PFP [10] is the best derived from FP-growth.

DPC [13] targets the problem of high I/O overhead with Apriori. It generates candidate itemsets and counts supports for as many levels as possible in a MapReduce job, and therefore may reduce the number of database scans. However, DPC still suffers for dense databases as it can only mine one level in one database scan in such a case.

PFP [10] targets a problem with FP-growth, that is, the FP-tree may be too large to be held in memory. It solves the problem by breaking the FP-tree into small subtrees that can be held in memory. But, PFP suffers from data redundancy as the subtrees overlap, which results in a huge amount of redundant data to be sent across the cluster. To alleviate the issue, the PFP implementation by Apache Mahout [22] only outputs the top-k frequent closed itemsets.

In summary, while the best sequential mining algorithms such as Apriori, FP-growth, and Eclat do not work for processing huge databases, the existing parallel algorithms either follow a parallelization paradigm that is not fault-tolerant or do not adequately address the issues with sequential algorithms.

This paper proposes the first algorithm that parallelizes the Eclat algorithm [19] for addressing the shortcomings of the existing MapReduce-based parallel algorithms. We take full advantage of parallelization provided by MapReduce [5], and propose an opportunistic vertical mining approach that improves the Eclat algorithm. Concretely, our contributions are as follows:

– We propose to employ hybrid vertical data formats in the mining process. The novelty is to make a choice on a per itemset basis in selecting a vertical data format that results in a smaller storage footprint. In particular, an itemset can be derived from two itemsets that use different data formats in representing

their transactions. Our approach improves not only scalability but also efficiency as the intermediate storage space, I/O overhead, and computation time all decrease in the mining process.
- We improve the breadth-first search by adopting the pruning technique from the Apriori algorithm [2]. That is, we avoid unnecessary set-intersection operations for computing supports of itmesets whose subsets are infrequent.
- We dynamically rearrange itemsets in the ascending order of supports, which results in more groups of itemsets grouped by prefixes and smaller group sizes. Such an ordering helps breaking the computation workloads into smaller pieces that can be processed in parallel, i.e., balancing the workloads.

The rest of the paper is organized as follows. Section 2 defines the mining problem, Sect. 3 proposes our opportunistic vertical mining approach, Sect. 4 presents our algorithm, Sect. 5 evaluates our work, and Sect. 6 concludes the paper.

2 Problem Statement and Preliminaries

This section first defines the mining problem, and then analyzes mining strategies and data formats that will be adapted and improved by our algorithm.

2.1 Problem Statement

The frequent itemset mining problem is stated as follows [2].

Definition 1 (Itemset and Subset). *Let I be a set of items. A set of items from I is also called an itemset. An itemset consisting of k items is called a k-itemset. An itemset S is called a subset of an itemset X if all items in S are also in X, denoted as $S \subseteq X$.*

Definition 2 (Frequent Itemset). *Let D be a transaction database over a set of items I, where each transaction t in D contains a set of items from I and has a unique identifier called tid, denoted as $t.tid$. The support of an itemset X, denoted as $\sigma(X)$, is the number of transactions that have X as a subset. The itemset X is frequent or large if its support, $\sigma(X)$, is no less than a user-specified minimum support threshold, denoted as minsup.*

The problem is to find the set of all frequent or large itemsets, i.e., $L = \{X \subseteq I \mid \sigma(X) \geq minsup\}$. For example, for the transaction database D in Table 1 and $minsup = 3$, the set L of all frequent itemsets consists of {a}, {b}, {c}, {f}, {m}, {p}, {a,c}, {a,f}, {a,m}, {c,f}, {c,m}, {c,p}, {f,m}, {a,c,f}, {a,c,m},{a,f,m}, {c,f,m}, and {a,c,f,m}.

2.2 Strategies for Mining Frequent Itemsets

Conceptually, all itemsets form a lattice based on a partial order, i.e., the subset relation on the set of itemsets. For example, Fig. 1 is a lattice on the powerset of {a,b,c,f,m,p}. To find all frequent itemsets, all algorithms adopt a bottom-up strategy to search a lattice of itemsets starting with the empty itemset, and employ the downward closure property partially or fully.

Table 1. Transaction database D

Tid	Items
1	{a, c, d, f, g, i, m, p}
2	{a, b, c, f, l, m, o}
3	{b, f, h, j, o}
4	{b, c, k, p, s}
5	{a, c, e, f, l, m, n, p}

Lemma 1 (Downward Closure Property). *All subsets of frequent itemset must be frequent. An itemset is infrequent if one of its subsets is infrequent.*

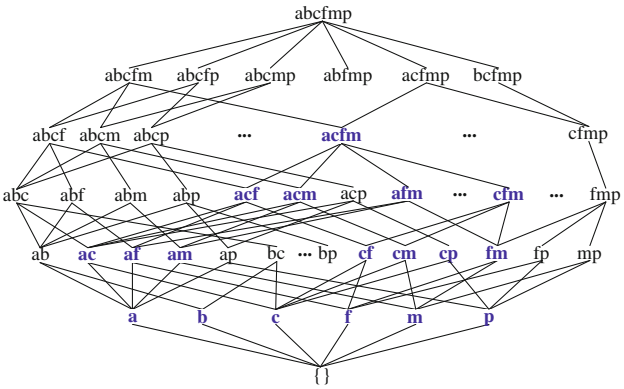

Fig. 1. Lattice of itemsets in the lexicographic order

The Apriori algorithm [2] fully utilizes the downward closure property in its bottom-up, breadth-first search of the lattice. In particular, it generates the set C_k of candidates, i.e., k-itemsets to be search at level k, from the set L_{k-1} of frequent (k-1)-itemsets by two functions, join and pruning.

- The join function forms a k-itemset, i.e., a candidate, by joining two (k-1)-itemsets that share a (k-2)-prefix.
- The pruning function prunes a candidate that has an infrequent subset of size k-1 (which may not be a prefix).

Definition 3 (Prefix). *Let X be a k-itemset, the h-prefix of X is the set of the first h items in X by a given ordering Ω of items, which is denoted as $prefix(X, h, \Omega)$ with $h \leq k$ and abbreviated as $prefix(X, h)$ when the ordering is obvious.*

The FP-growth algorithm [9] partially utilizes the downward closure property in its bottom-up, depth-first search of the lattice. In particular, for each frequent itemset X, it recursively searches the union of X and an item locally frequent in

Table 2. The tidsets and diffsets of frequent items

Frequent item	Tidset	Diffset
a	{1, 2, 5}	{3,4}
b	{2, 3, 4}	{1,5}
c	{1, 2, 4, 5}	{3}
f	{1, 2, 3, 5}	{4}
m	{1, 2, 5}	{3,4}
p	{1, 4, 5}	{2,3}

the conditional database of X, which is similar to the working of the Apriori's join function. The Eclat algorithm [19] also partially utilizes the downward closure property in its bottom-up search of the lattice. In particular, it also employs a join function similar to Apriori's in determining itemsets to be searched.

2.3 Support Counting and Data Formats

When bottom-up searching the lattice of itemsets, algorithms will count the support of each itemset currently being examined, which depends on the data format for representing the database. There are two categories of data formats, horizontal and vertical.

The Apriori algorithm [2] determines the supports of itemsets by directly scanning the database in the horizontal format. The FP-growth algorithm [9] reads off the supports from the FP-tree that compresses the horizontal database.

The Eclat algorithm [19] turns the database into the tidset format as in Definition 4, and calculates the support of an itemset by intersecting the tidsets of the subsets of the itemset. Another vertical format related to tidset, is the diffset format [20].

Definition 4 (Tidset). *Let D be a transaction database over a set of items I. The tidset of an itemset X is denoted by $tidset(X) = \{t.tid \mid t \in D, X \subseteq t\}$. Thus, $\sigma(X) = |tidset(X)|$.*

Definition 5 (Diffset). *The diffset of X is the difference between the tidset of $prefix(X, k\text{-}1)$ and the tidset of X, that is, $diffset(X) = tidset(prefix(X, |X| - 1)) - tidset(X)$. Thus, $\sigma(X) = \sigma(prefix(X, |X| - 1)) - |diffset(X)|$.*

For example, Table 2 shows both tidsets and diffsets of frequent items for D in Table 1 and $minsup = 3$. Moreover, the tidset of itemset {a,c} is the intersection of the tidsets of {a} and {c}, that is, $\{1, 2, 5\} \cap \{1, 2, 4, 5\} = \{1, 2, 5\}$. Similarly, $tidset(\{c,p\}) = tidset(\{c\}) \cap tidset(\{p\}) = \{1,4,5\}$, and $tidset(\{a,c,p\}) = \{1, 2, 5\} \cap \{1, 4, 5\} = \{1,5\}$. Given the lexicographic order, $diffset(\{a,c\}) = \{\}$ and $diffset(\{a,c,p\}) = \{2\}$, and $\sigma(\{a,c,p\}) = \sigma(\{a,c\})$ - $|diffset(\{a,c,p\})| = 2$.

3 Opportunistic Vertical Mining Approach

We propose an opportunistic vertical mining approach that improves the Eclat algorithm [19] by employing a hybrid vertical data format per itemset, by pushing additional pruning into breadth-first search, and by facilitating balanced parallelization of depth-first search.

3.1 Our Hybrid Vertical Format

Traditional vertical mining algorithms [6,15,16,19,20] utilize only one vertical format, either tidset or diffset. When databases are sparse, the tidset is the preferred choice. When databases are dense, the diffset may perform better. But when databases get huge, neither single vertical format scales well. A good approach is to independently choose the vertical format for each individual itemset by considering the resulting storage footprint.

For example, the tidset is good for {a,b} as $tidset(\{a,b\}) = \{2\}$ and $diffset(\{a,b\}) = \{1,5\}$ while the diffset is good for {a,c,p} as $tidset(\{a,c,p\}) = \{1,5\}$ and $diffset(\{a,c,p\}) = \{2\}$, given the lexicographic ordering of items.

To accommodate such a good approach, we present a hybrid vertical format, mixset, as in Definition 6.

Definition 6 (Mixset). *The mixset of an itemset X, denoted as $mixset(X)$ is defined as*

$$mixset(X) = \begin{cases} tidset(X) & \text{if } |tidset(X)| \leq |diffset(X)| \\ diffset(X) & \text{otherwise} \end{cases}$$

For example, $mixset(\{a,b\}) = tidset(\{a,b\}) = \{2\}$, and $mixset(\{a,c,p\}) = diffset(\{a,c,p\}) = \{2\}$.

Notice that mixset is an instrument for facilitating our approach, that is, to automatically select the tidset format or the diffset format on a per itemset basis, which is made possible both in breadth-first search and in depth-first search. Our approach is greatly different from approaches that use only the tidset format [19] or the diffset format for all itemsets, or manually determine at which level to switch from one format to another [20].

3.2 Enabling Our Opportunistic Vertical Mining

To enable our opportunistic vertical mining approach, we need to compute the mixset for any itemset that is a union of two k-itemsets X and Y that share a (k-1)-prefix, which we denote as $mixset(X \cup Y) = mixset(X) \cap mixset(Y)$.

Notice that there is no prior work that computes the tidset and diffset of the union of two itemsets when only given the tidset of one itemset and the diffset of the other. Therefore, we investigate in six cases by Theorem 1 how to compute $mixset(X) \cap mixset(Y)$.

Theorem 1. *Given an ordering Ω where itemset X is listed before itemset Y, and $|X| = |Y| = k$ and $prefix(X, k\text{-}1) = prefix(Y, k\text{-}1)$, we have:*

$$\begin{aligned} tidset(X \cup Y) &= tidset(X) \cap tidset(Y) && (1)\\ &= tidset(X) - diffset(Y) && (2)\\ &= tidset(Y) - diffset(X) && (3)\\ diffset(X \cup Y) &= tidset(X) - tidset(Y) && (4)\\ &= tidset(X) \cap diffset(Y) && (5)\\ &= diffset(Y) - diffset(X) && (6) \end{aligned}$$

For example, for D in Table 1 with the lexicographic ordering, we have $tidset(\{a\}) = \{1, 2, 5\}$, $tidset(\{c\}) = \{1,2,4,5\}$, and $tidset(\{p\}) = \{1,4,5\}$ in Table 2. According to Theorem 1, we get $tidset(\{a,c\}) = \{1,2,5\}$ and $tidset(\{a,p\}) = \{1,5\}$ by (1), and $diffset(\{a,c\}) = \{\}$ and $diffset(\{a,p\}) = \{2\}$ by (4). Furthermore, we can get $tidset(\{a,c,p\}) = \{1,5\}$ by (1), by (2), or by (3), and $diffset(\{a,c,p\}) = \{2\}$ by (4), by (5), or by (6).

3.3 Our Search Strategy

To avoid repetitively enumerating itemsets in bottom-up searching the lattice, all algorithms assume an ordering of itemsets. Thus, all algorithms can also be thought of as top-down searching an itemset enumeration tree, and fall into three categories, breadth-first, depth-first, or the hybrid.

First, we push additional pruning into the breadth-first search. Notice that Eclat [19] determines the support of any k-itemset by directly intersecting the tidsets of any two of its size (k-1) subsets. In other words, Eclat executes the itemset-union (Apriori's join) and tidset-intersection (support-counting) without Apriori's pruning. We propose to push the Apriori's pruning to avoid unnecessary tidset-intersection operations since all frequent (k-1)-itemsets are available when enumerating k-itemsets in a breadth-first search.

Second, we facilitate parallelization of depth-first search by dynamically arranging sibling itemsets with the same parent itemset from left to right in the ascending order of supports. With such an ordering, more frequent itemsets will root smaller subtrees to be further searched, which results in a smaller and more balanced itemset enumeration tree. Clearly, such an ordering helps balance the workload when depth-first searching the subtrees in parallel, and ultimately improves the efficiency.

4 New Parallel Algorithm Based on MapReduce

This section presents our new parallel algorithm for mining frequent itemsets based on MapReduce [5]. We call our algorithm Parallel Eclat or Peclat for short to note the connection between our algorithm and Eclat [19].

Our algorithm significantly differs from Eclat as discussed in Sect. 3, and parallelizing our approach based on the MapReduce paradigm as presented in this section is nontrivial.

4.1 Our Peclat Algorithm

The pseudo code of our Peclat algorithm is listed in Algorithm 1, which works as follows. First, Peclat invokes a MapReduce job, mrCountingItems, to find all frequent items and to output a list of frequent items together with the mixset for each frequent item (at line 1).

Algorithm 1. Peclat
Input: D, $minsup$
Output: all frequent itemsets
1: $L_1 \leftarrow$ mrCountingItems$(D, minsup)$;
2: **for** ($k = 2$; $k \leq m \wedge L_{k-1} \neq \{\}$; k++) **do**
3: $L_k \leftarrow$ mrLargeK$(L_{k-1}, minsup)$;
4: **end for**
5: $L \leftarrow$ mrMiningSubtrees$(L_m, minsup)$;

Second, Peclat calls a MapReduce job, mrLargeK, in a maximum m rounds to search the itemset enumeration tree in a breadth-first manner (at lines 2 – 4). The breadth-first search will also terminate if all frequent itemsets are found.

Finally, Peclat switches to a depth-first search by invoking a MapReduce job, mrMiningSubtrees that mines subtrees of the itemset enumeration in parallel (at line 5). The output L_k of the MapReduce jobs is a list of ⟨key = itemset, value = mixset⟩ pairs.

4.2 The mrCountingItems Job

This MapReduce Job first produces the tidset for each item in its map phase, and then outputs each frequent item with its mixset in its reduce phase.

Concretely, in the map phase, the map function is invoked as many times as the number of transactions. Each time the map function reads one transaction and outputs an ⟨item, tid⟩ pair for every item in the transaction.

For example, given D in Table 1 and $minsup = 3$, one invocation of the map function will read the first transaction (tid =1) and will output the following ⟨item, tid⟩ pairs: ⟨a, 1⟩, ⟨c, 1⟩, ⟨d, 1⟩, ⟨f, 1⟩, ⟨g,1⟩, ⟨i, 1⟩, ⟨m, 1⟩, and ⟨p, 1⟩, which will be passed to the reduce phase.

In the reduce phase, the reduce function is invoked as many times as the number of distinct items. Each time the reduce function gets a distinct item and a list of tids of the transactions that contain the item. If the item is frequent, it outputs the item with its mixset. For example, the pairs ⟨a, 1⟩, ⟨a, 2⟩, and ⟨a, 5⟩ produced in the map phase will be fed into one invocation of the reduce function, which will first get $tidset(\{a\}) = \{1, 2, 5\}$ and output {a} with $mixset(\{a\}) = diffset(\{a\}) = \{3, 4\}$ as diffset is smaller than tidset.

MapReduce Job: mrCountingItems
Input: D, $minsup$
Output: L_1
Map (key = null, value = transaction T_i)
1: **for each** $item \in T_i$ **do**
2: output $\langle item, T_i.tid \rangle$;
3: **end for**
Reduce (key = $item$, value = $list\text{-}of\text{-}tids$)
1: $tidset(item) \leftarrow \{\}$;
2: **for each** value $t \in list\text{-}of\text{-}tids$ **do**
3: $tidset(item) \leftarrow tidset(item) \cup \{t\}$;
4: **end for**
5: **if** $\sigma(item) \geq minsup$ **then**
6: output $\langle item, mixset(item) \rangle$;
7: **end if**

The reduce phase for the previous example eventually outputs the following ⟨1-itemset, mixset⟩ pairs: ⟨{a}, $\{3,4\}_{diffset}$⟩, ⟨{b}, $\{1,5\}_{diffset}$⟩, ⟨{c}, $\{3\}_{diffset}$⟩, ⟨{f}, $\{4\}_{diffset}$⟩, ⟨{m}, $\{3,4\}_{diffset}$⟩, and ⟨{p}, $\{2,3\}_{diffset}$⟩.

In examples, additional information for each ⟨itemset, mixset⟩ pair expressed as labels, e.g., ⟨{a}, $\{3,4\}_{diffset}$⟩ indicates that {a} is with its diffset {3,4}.

4.3 The mrLargeK Job

This MapReduce job mines all frequent k-itemsets for a given k. First of all, the map phase is initialized by reading the frequent (k-1)-itemsets L_{k-1} and calling the join and pruning functions in Sect. 2.2 to generate the candidate k-itemsets C_k. For example, given D in Table 1 and $minsup = 3$, when calling the job with $k = 3$, we get C'_3 = {{a,c,f}, {{a,c,m}, {a,f,m}, {c,f,m}, {c,f,p}, {c,m,p}} after join, and C_3 = {{a,c,f}, {{a,c,m}, {a,f,m}, {c,f,m}} after pruning.

After initialization, each invocation of the map function will reads a $\langle X, mixset(X) \rangle$ pair and match the itemset X with each candidate c in C_k, and will produce a $\langle$c, $mixset(X)\rangle$ pair if X is a subset of c and shares a common (k-2)-prefix with c, which will be passed to the reduce phase for computing $mixset(c)$.

For example, when calling the job with $k = 3$, one invocation of the reduce function inputs ⟨{a,c}, $\{\}_{diffset}$⟩, and outputs ⟨{a,c,f}, ⟨{a,c}, $\{\}_{diffset}$⟩⟩ and ⟨{a,c,m}, ⟨{a,c}, $\{\}_{diffset}$⟩⟩. Another invocation inputs ⟨{a,f}, $\{\}_{diffset}$⟩, and outputs ⟨{a,c,f}, ⟨{a,f}, $\{\}_{diffset}$⟩⟩ and ⟨{a,f,m}, ⟨{a,f}, $\{\}_{diffset}$⟩⟩.

In the reduce phase, each invocation of the reduce function will get a candidate itemset c and the mixsets of two itemsets X_1 and X_2 that share a common prefix with c, and will compute $mixset(c)$ as in Sect. 3.2. If the candidate is frequent, it outputs a $\langle c, mixset(c) \rangle$ pair.

MapReduce Job: mrLargeK
Input: L_{k-1}, $minsup$
Output: L_k
Initialization of Map Phase:
1: read L_{k-1};
2: $C'_k \leftarrow$ join(L_{k-1});
3: $C_k \leftarrow$ pruning(C'_k, L_{k-1});
Map (key = itemset X, value = $mixset(X)$)
1: load C_k;
2: **for each** candidate $c \in C_k$ **do**
3: **if** $X \subset c \wedge prefix(c, k\text{-}2) = prefix(X, k\text{-}2)$
4: **then** output $\langle c, mixset(X)\rangle$;
5: **end for**
Reduce (key = candidate c, value = $list\text{-}of\text{-}mixsets$)
1: $mixset(X_1) \leftarrow$ the first in $list\text{-}of\text{-}mixsets$;
2: $mixset(X_2) \leftarrow$ the second in $list\text{-}of\text{-}mixsets$;
3: $mixset(c) \leftarrow mixset(X_1) \cap mixset(X_2)$;
4: **if** $\sigma(c) \geq minsup$ **then** output $\langle c, mixset(c)\rangle$;

For example, when calling the job with $k = 3$, one invocation of the reduce function takes {a,c,f} and the list of $\langle${a,c},{}$_{diffset}\rangle$ and $\langle${a,f},{}$_{diffset}\rangle$ as input, and outputs $\langle${a,c,f}, {}$_{diffset}\rangle$. The reduce phase eventually outputs $\langle${a,c,f}, {}$_{diffset}\rangle$, $\langle${a,c,m}, {}$_{diffset}\rangle$, $\langle${a,f,m}, {}$_{diffset}\rangle$, and $\langle${c,f,m}, {}$_{diffset}\rangle$ for the job with $k = 3$.

MapReduce Job: mrMiningSubtrees
Input: L_k, $minsup$
Output: k'-frequent itemsets for $k' \geq k$
Map (key = k-itemset X, value = $mixset(X)$)
1: output $\langle prefix(X, k\text{-}1), \langle X, mixset(X)\rangle\rangle$;
Reduce (key = (k-1)-$prefix$, value = $siblings$)
1: mine($siblings, minsup$);
Subroutine: mine($siblings, minsup$)
1: **for each** $\langle X, mixset(X)\rangle \in siblings$ **do**
2: $children.clean()$;
3: **for each** $\langle Y, mixset(Y)\rangle \in siblings$ after X **do**
4: $Z \leftarrow X \cup Y$;
5: $mixset(Z) \leftarrow mixset(X) \cap mixset(Y)$;
6: **if** $\sigma(Z) \geq minsup$ **then**
7: $children.push(\langle Z, mixset(Z)\rangle)$;
8: output Z;
9: **end if**
10: **end for**
11: **if** $children \neq \emptyset$ **then** mine($children, minsup$);
12: **end for**

Table 3. Features of datasets used in experiments

Dataset	numTrans	Items	maxLenTrans	avLenTrans	Category
BMS-WebView-1	59,602	497	267	2.5	sparse
BMS-WebView-2	77,512	3,340	161	5.6	sparse
Connect4	67,557	150	43	43.0	dense
Pumsb*	49,045	2,088	63	50.5	dense
T40I10D100k	99,999	150	68	37.0	mixed
T30I10D100k	99,998	120	53	27.8	mixed

4.4 The mrMiningSubtrees Job

This MapReduce job splits the itemset enumeration tree into small subtrees, and then mines the subtrees in parallel.

In the map phase of this job, each invocation of the map function inputs an ⟨itemset, mixset⟩ pair, and outputs the prefix of the itemset as the key and the ⟨itemset, mixset⟩ pair as the value.

In the reduce phase, each invocation of the reduce function can get a list, *siblings*, of all the itemsets with a same prefix together and their mixsets as the input value, and the prefix as the input key. Therefore, the branches of the itemset enumeration tree rooted at *siblings* can be recursively searched by the mine subroutine.

For example, given all the frequent 2-itemsets, {a,c}, {a,f}, {a,m}, {c,f}, {c,m}, {c,p}, and {f,m} together with their mixsets as the input, the map phase will distribute these pairs into three groups with {a}, {c}, and {f} as their prefixes respectively if the lexicographic order is used, or into four groups with {a}, {m}, {p}, and {c} respectively if the ascending order of supports is used. The subtree holding the itemsets in a group, e.g., {a,c}, {a,f}, and {a,m}, will be mined by one invocation of the reduce function.

5 Experimental Evaluation

We evaluate our Peclat by experimentally comparing with DPC [13] and mahout (closed) [22], an implementation of PFP [10], that mines frequent closed itemsets. All algorithms were implemented in Java.

Six datasets[1,2,3] are used in experiments and described in Table 3. All experiments were performed on a Hadoop-0.20.2 cluster of three nodes, 1 master and 2 workers, where each node is equipped with an Intel(R) Core(TM) i5-2400 3.10GHz CPU, 2GB RAM, and a 500GB hard disk running Ubuntu11.10.

[1] http://www.sigkdd.org/kdd-cup-2000/.
[2] http://fimi.ua.ac.be/data/.
[3] http://sourceforge.net/projects/ibmquestdatagen/.

5.1 Performance Comparison with Other Algorithms

We compare the running time of DPC [13], mahout(closed) [22], and our Peclat on six datasets. Peclat runs in a breadth-first search for two levels and then switch to a depth-first search. The result is summarized by Fig. 2.

For sparse datasets as in Fig. 2(a)(b), the algorithms from the least efficient to the most efficient are DPC, mahout(closed), and Peclat. For example, when *minsup* = 0.1 % for BMS_WebView_1, DPC takes 282 s, mahout(closed) 153, and Peclat 97. When *minsup* = 0.3 % for BMS_WebView_2, DPC takes 313 s, mahout(closed) 141, and Peclat 102.

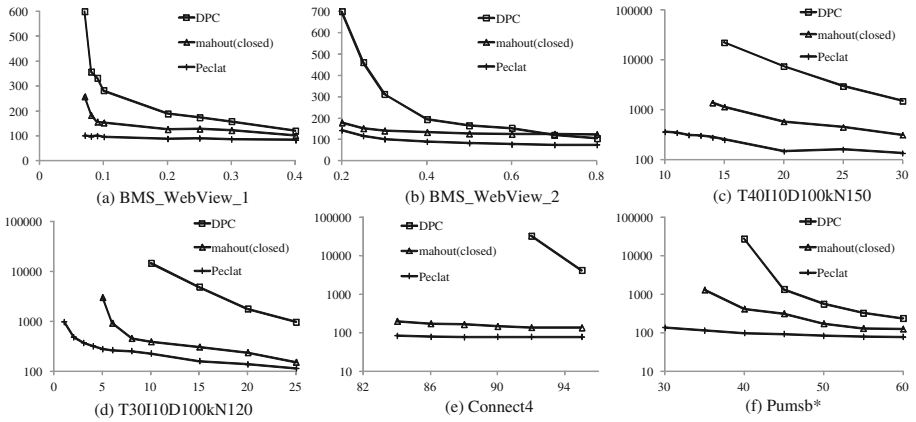

Fig. 2. Running-time (seconds) of three algorithms vs. *minsup*(%)

For datasets of mixed characteristics as in Fig. 2(c)(d), DPC is the least efficient, and Peclat is the most efficient. For example, when *minsup* = 15 % for T40I10D100k, DPC takes 22,432 s, mahout(closed) 1,150, and Peclat 254. When *minsup* = 10 % for T30I10D100k, DPC takes 14,690 s, mahout(closed) 401, and Peclat 227.

For dense datasets as in Fig. 2(e)(f), Peclat beats mahout(closed), and mahout(closed) beats DPC. For example, when *minsup* = 92 % for Connect4, DPC takes 32,41 s, mahout(closed) 136, and Peclat 78. When *minsup* = 40 % for Pumsb*, DPC takes 27,548 s, mahout(closed) 450, and Peclat 98.

The observations are as follows.

- Our Peclat is the most efficient algorithm, and DPC [13] is the least efficient for all the six datasets.
- Although the evaluation favours mahout(closed) as it only finds frequent closed itemsets, i.e., it works on a much easier mining problem, our Peclat still outperforms mahout(closed) significantly.

5.2 Anatomy of Opportunistic Vertical Mining Approach

To explain the observed results in the last subsection, this subsection analyzes the three techniques in our opportunistic vertical mining approach.

Selecting Vertical Format per Itemset. We first study the numbers of tidsets and diffsets chosen by our Peclat algorithm in the mining process, denoted as tidset_num and diffset_num respectively.

The aggregate tidset_num and diffset_num with varying *minsup* are shown in Fig. 3(a)(b), and tidset_num and diffset_num at each level of the itemset enumeration tree (i.e. grouped by the size of itemsets) are shown in Fig. 3(c)–(f). For T40I10D100k, our Peclat algorithm selects the tidset format for 25 % of mixsets and the diffset format for 75 %. For T30I10D100k, 45 % of mixsets are in the tidset format, and 55 % are in the diffset format.

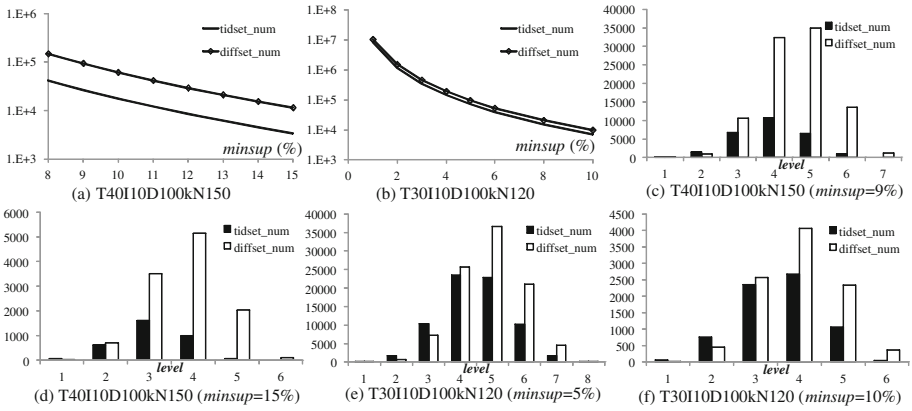

Fig. 3. Numbers of tidsets and diffsets (aggregate and per level)

Our Peclat, depicted as Peclat_mixset in Fig. 4(a)(b) as it employs the mixset format, is compared with variants that employ the tidset and diffset formats. By piecing together the results in Figs. 3 and 4(a)(b), we observe the following.

- Our mixset format is superior to both tidset and diffset since no dataset is purely sparse or purely dense. Selecting a data format on a per itemset basis greatly reduces the computation cost and the space usage.
- Our Peclat algorithm is good for all datasets regardless of data characteristics while other algorithms [20] may need manual fine-tuning by users and cannot take full advantage of a hybrid data format.

Ordering Itemsets in Ascending Supports. We study the improvement by sorting itemsets in the ascending order of supports by comparing our Peclat with its variant, Peclat_Lexi, that sorts itemsets in the lexicographic order.

Such an improvement is significant as shown in Fig. 4(c)(d). For example, when $minsup = 8\,\%$ for T40I10D100k, Peclat takes 443 s while Peclat_Lexi takes 640 s. When $minsup = 1\,\%$ for T30I10D100k, Peclat takes 983 s while Peclat_Lexi takes 4,403 s.

Pushing Additional Pruning. Finally, we analyze the effect of applying the Apriori's pruning in improving the breadth-first search. To do so, we compare two variants of our algorithm, Peclat_BF that integrates the Apriori's pruning in the breadth-first search and Peclat_BF_noPrune that does not.

The result is that the improvement by integrating the Apriori's pruning is significant as shown in Fig. 4(e)(f). For example, when *minsup* = 15 % for T40I10D100k, Peclat_BF takes 1,168 s while Peclat_BF_noPrune takes 2,551 s. When *minsup* = 10 % for T30I10D100k, Peclat_BF takes 1,320 s while Peclat_BF_noPrune takes 2,306 s.

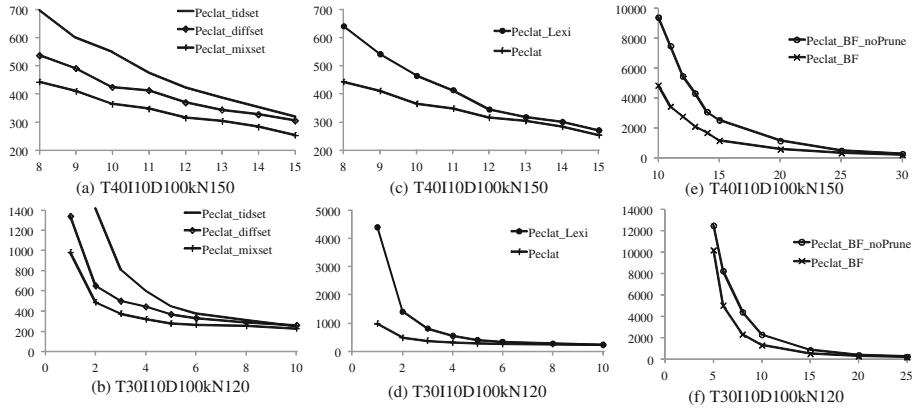

Fig. 4. Running-time of (a)(b) using tidsets, diffsets, mixsets; (c)(d) with/without ascending order of supports; (e)(f) with/without more pruning on 2 datasets

6 Conclusion

This paper has proposed Peclat, a parallel Eclat-like algorithm based on MapReduce. Peclat outperforms the existing algorithms because it improves the Eclat algorithm in three aspects including a hybrid data format that saves both time and space, an ordering of itemsets that helps balancing workloads, and additional pruning for breadth-first search.

Acknowledgements. This work was supported in part by the National Natural Science Foundation of China (61272306), and the Zhejiang Provincial Natural Science Foundation of China (LY12F02024).

References

1. Agrawal, R., Shafer, J.: Parallel mining of association rules. IEEE Trans. Knowl. Data Eng. **8**, 962–969 (1996)
2. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: 20th VLDB, p. 487 (1994)

3. Chen, X., He, Y., Chen, P., Miao, S., Song, W., Yue, M.: HPFP-Miner: a novel parallel frequent itemset mining algorithm. ICNC **3**, 139–143 (2009)
4. Cyrans, J.-D., Ratt, S., Champagne, R.: Adaptation of apriori to MapReduce to build a warehouse of relations between named entities across the web. In: 2010 DBKDA, pp. 185–189 (2010)
5. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
6. Dunkel, B., Soparkar, N.: Data organization and access for efficient data mining. In: 15th ICDE, pp. 522–529 (1999)
7. Farzanyar, Z., Cercone, N.: Efficient mining of frequent itemsets in social network data based on MapReduce framework. In: ACM International Conference on Advances in Social Networks Analysis and Mining, pp. 1183–1188 (2013)
8. Hammoud, S.: MapReduce network enabled algorithms for classification based on association rules. Ph.D. Thesis, Brunel University (2011)
9. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: 2000 SIGMOD, pp. 1–12 (2000)
10. Li, H., Wang, Y., Zhang, D., Zhang, M., Chang, E.Y.: PFP: parallel FP-growth for query recommendation. In: 2008 ACM Conference on Recommender System (RecSys 2008), pp. 107–114 (2008)
11. Li, L., Zhang, M.: The strategy of mining association rules based on cloud computing. In: BCGIN, pp. 475–478 (2011)
12. Li, N., Zeng, L., He, Q., Shi, Z.: Parallel implementation of apriori algorithm based on MapReduce. In: ACIS International Conference on Software Engineering, Artificial Intelligence, Networking & Parallel/Distributed Computing, pp. 236–241 (2012)
13. Lin, M.-Y., Lee, P.-Y., Hsueh, S.-C.: Apriori-based frequent itemset mining algorithms on MapReduce. In: ICUIMC (2012)
14. Riondato, M., DeBrabant, J.A., Fonseca, R., Upfal, E.: PARMA: a randomized parallel algorithm for approximate association rule mining in MapReduce. In: 21st CIKM, pp. 85–94 (2012)
15. Sarawagi, S., Thomas, S., Agrawal, R.: Integrating association rule mining with databases: alternatives and implications. In: 1998 SIGMOD, pp. 343–354 (1998)
16. Shenoy, P., Haritsa, J.R., Sudarshan, S.: Turbo-charging vertical mining of large databases. In: 2000 SIGMOD, pp. 22–33 (2000)
17. Sohrabi, M.K., Barforoush, A.A.: Parallel frequent itemset mining using systolic arrays. Knowl. Based Syst. **37**, 462–471 (2013)
18. Yang, X.Y., Liu, Z., Fu, Y.: MapReduce as a programming model for association rules algorithm on Hadoop. In: ICIS (2010)
19. Zaki, M.J.: Scalable algorithms for association mining. IEEE Trans. Knowl Data Eng. **12**(3), 372–390 (2000)
20. Zaki, M.J., Gouda, K.: Fast vertical mining using diffsets. In: 9th SIGKDD, pp. 326–335 (2003)
21. Zaki, M.J., Ogihara, M., Parthasarathy, S., Li, W.: Parallel algorithms for discovery of association rules. Data Min. Knowl. Disc. **1**(4), 343–373 (1997)
22. Apache Mahout. http://mahout.apache.org/