Abstract—Frequent itemset mining is a fundamental and essential issue in data mining field and can be used in many data mining tasks. Most of these mining tasks require multiple passes over the database and if the database size is large, which is usually the case, scalable high performance solutions involving multiple processors are required. In this paper, we present a novel parallel frequent itemset mining algorithm which is called HPFP-Miner. The proposed algorithm is based on FP-Growth and introduces little communication overheads by efficiently partitioning the list of frequent elements list over processors. The results of experiment show that HPFP-Miner has good scalability and performance.

Keywords—frequent itemset; data mining; parallel; HPFP-Miner; FP-Growth

I. INTRODUCTION

Frequent itemset mining is a fundamental and essential issue in data mining field and can be used in many data mining tasks which include association rules analysis, correlations analysis, classification, clustering, outlier analysis, etc. Since 1993 Agrawal et al.[1] proposed association rules analysis problem through research of the relationship among itemsets in custom transaction database and proved the core of association rules analysis problem was mining the frequent itemsets which could generate strong association rules, the issue of mining frequent itemsets has been an active research topic.

However, the huge search space makes frequent pattern mining on large databases a very computationally intensive problem. The performance of a serial frequent pattern mining algorithm is limited by its single processor computing capability and finite memory space; thus, they do not scale to large datasets. Therefore, it is very necessary to exploit high performance parallel/distributed computing to relieve the current frequent pattern mining algorithms from the sequential bottleneck, and helps the FPM algorithms scale to massive datasets[2].

FP-Growth[3] algorithm proposed by J.Han in 2000 is one of the most popular frequent itemset mining algorithm. It adopts divide-and-conquer method which compresses transaction database to a frequent pattern tree—FP-tree first, then mines complete frequent itemsets from FP-tree. The intrinsic divide-and-conquer nature of FP-growth can easily be parallelized. In 2001, Osmar R. Zaïane et al. proposed MLFPT[4] which is a simple parallel algorithm based on FP-growth for mining frequent itemsets on shared memory machines. The key idea behind this task parallel paradigm is the divide-and-conquer nature of the FP-growth algorithm. Later, Javed and Khokhar et al. proposed another FP-growth-based parallel algorithm called PFP-tree[5] but running on a distributed memory architecture. The FP-tree building process is similar to that in MLFPT except that multiple FP-trees reside in different physical memory spaces. The mining task is also divided in the same way as in MLFPT. Another difference is that the conditional pattern base of each 1-itemset must be exchanged among processors so that each can hold the needed portion of the database locally.

In this paper, we present a novel parallel frequent itemset mining algorithm, called HPFP-Miner, to mine the complete set of frequent itemsets on parallel distributed memory architecture. We implement HPFP-Miner on four real world datasets, and compare the results with two famous FP-Growth-like algorithms. The experimental results show that our algorithm has a better scalability and performance.

II. RELATED WORK

For the sake of completeness, we give the relative problem description of frequent itemset mining and FP-Growth in this section.

Definition 1: Let \( I=\{i_1, i_2, \cdots, i_n\} \) be a set of items, where an itemset \( X \) is a subset of \( I \) such that \( X \subseteq I \). An itemset of length \( k \) is called \( k \)-itemset. Let \( D \) be a set of transactions, where each transaction \( T \) is a set of items such that \( T \subseteq D \). A unique identified TID is given to each transaction. The support of an itemset \( X \) is the number of transactions in \( D \) that contain \( X \). If the support of \( X \) is greater than or equal to a predefined support threshold \( \lambda \), it is called a frequent itemset or frequent pattern, otherwise, it is infrequent.

The frequent pattern mining problem is to discover the complete set of all patterns contained in at least a specified support threshold \( \lambda \), of transactions in the transaction database.

FP-Growth-like algorithms adopt divide-and-conquer method which can be stated as follow: First, it compresses the database representing frequent items into a frequent-pattern tree, or FP-tree, which retains the itemset association information. It then divides the compressed database into a set of conditional databases (a special kind of projected database), each associated with one frequent item or “pattern fragment,” and mines each such database separately[6].

FP-Growth opened up a new way to efficiently mine frequent pattern. However, its low time and space utilization...
still need to improve. Many variants of FP-Growth appeared recently. The representations include FP-Growth* [7] proposed by G.Grahne et al. in 2003 and AFOPT [8][9] proposed by G.Liu et al. at the same year. FP-Growth* adopts a new array technique to enhance the operation capability. It constructs a two-dimensional array at the same time of building FP-Tree to preserve the support counts of all 2-itemsets. By using this array FP-Growth* only need scan FP-Tree once at the time of each recursion, which improve the efficiency of FP-Growth. AFOPT construct a sample and compact data structure Ascending Frequency Ordered Prefix-Tree (AFOPT) based on FP-Tree. The algorithm adopts top-down scan to reduce the number of conditional databases and depress the overhead of scanning each conditional database. We have studied on the FP-Growth-like algorithm for a long time and already get some achievements, such as F-Miner in [10] and LPS-Miner in [11].

### III. DESIGN AND CONSTRUCTION OF HPFP-FOREST

In this paper, we proposed a new frequent pattern mining algorithm HPFP-Miner for distributed memory parallel environment. Before talking about the algorithm, we would like to introduce a FP-tree-like data structure HPFP-Tree and HPFP-Forest which is composed by many HPFP-Trees.

Unlike FP-tree, the root of HPFP-Tree is not “null” but an item to identify this tree. HPFP-Tree compresses the FP-Tree greatly by eliminating the parent-to-child pointer; only reserve the pointer which links to its parent for each node. Each HPFP-Tree independently and completely represents the node that the current database takes the root node as prefix. Therefore, many HPFP-Trees can be assigned to different processors to achieve parallel processing. We take an example to implement the building and mining procedure of HPFP-Tree and HPFP-Forest. Given a transaction database D as shown in table 1, the minimum support threshold $\lambda = 2$. Suppose our algorithm has 3 viable processors at the time of execution.

**TABLE I. TRANSACTION DATABASE D**

<table>
<thead>
<tr>
<th>TID</th>
<th>Transactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>T100</td>
<td>I1, I2, I5</td>
</tr>
<tr>
<td>T200</td>
<td>I2, I4</td>
</tr>
<tr>
<td>T300</td>
<td>I2, I3</td>
</tr>
<tr>
<td>T400</td>
<td>I1, I2, I4</td>
</tr>
<tr>
<td>T500</td>
<td>I1, I3</td>
</tr>
<tr>
<td>T600</td>
<td>I2, I3</td>
</tr>
<tr>
<td>T700</td>
<td>I1, I3</td>
</tr>
<tr>
<td>T800</td>
<td>I1, I2, I3, I5</td>
</tr>
<tr>
<td>T900</td>
<td>I1, I2, I3</td>
</tr>
</tbody>
</table>

First, in an initial scan, the support value of each item is determined and we get the 1-frequent itemset list L ordered in support descending: \{\{I2: 7\}, \{I1: 6\}, \{I3: 6\}, \{I4: 2\}, \{I5: 2\}\} (the number after “:” indicates the pattern’s support value). This part is processed by processor 0.

Secondly, the rest processors built empty HPFP-Tree in their own memories independently, in which HPFP-Trees are linked by pointers. In the example, the HPFP-Tree in processor 1 contains nodes I2, I3 and I5 which point to HPFP-Tree labeled with T12, T13 and T15 separately, and the HPFP-Forest in processor 2 contain nodes I1 and I4 which point to HPFP-Tree labeled with T11 and T14.

The second database scan is still processed by processor 0. Scanning each record in the database and deleting all the items dissatisfy the minimum support. Items in each record are ordered by 1-frequent itemset order. For example, T100 becomes F100 = \{I2, I1, I5\} (F100 is the ordered frequent item list of T100). Afterwards, processor 0 assigns all combinations of the sub-itemsets according to the first item of sub-itemsets. For instance, \{I2,I1,I5\} and \{I5\} are assigned to HPFP-Trees which the root are T12 and T15 in processor 1, and \{I1,I5\} to HPFP-Tree which the root is T11 in processor 2. The finished HPFP-Forest in each processor is shown in figure 1 and the algorithm of building HPFP-Forest is shown in algorithm 1.

![Figure 1. HPFP-Forest in each processor](image.png)

**Algorithm 1:** HPFP-Forest Initialization

**Input:** transaction database $D$, minimum support $\lambda$, number of viable processor numprocs(except processor 0)

**Procedure:**

for each transaction $t$ of $D$

for each item of $t$ // $t$ represent transaction of $D$

$\text{support}[\text{item}]++$;

$numItem = \text{max}(\text{items})$;

$\text{sort}(\text{support1, largeItem1, 0, numItem});$ //sort descending

for each transaction $t$ of $D$

$\text{sortTrans}(\text{length}_t)$;

for $i = \text{length}_t$; $i > 1; i - = 1$

$\text{dec} = (\text{numItem} - i)/\text{numprocs} + 1$;

$\text{MPI Send}(\text{message}, \text{length}_msg, \text{MPI_INT, dec, 99, MPI_COMM_WORLD})$;
IV. HPFP-MINER ALGORITHM

In the previous section we introduced a new data structure HPFP-Tree, which can reserve transaction database in memory more efficiently. In order to process parallel mining of frequent pattern from transaction database, in this section we propose a novel parallel frequent pattern mining algorithm—HPFP-Miner.

A. HPFP-Miner Algorithm

HPFP-Miner algorithm can be stated as following two main algorithms: the Parallel Algorithm and Fminer which are shown in algorithm 2 and 3.

Algorithm 2: Parallel Algorithm
Input: minimum support threshold $\lambda$

procedure:

1. MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
2. MPI_Comm_rank(MPI_COMM_WORLD,&myid);
3. numprocs-=1;
4. MPI_Recv(message,length,MPI_INT,0,99,MPI_COMM_WORLD,&status);
5. insertNode(message,length,numprocs);
6. sign=numItem/numprocs;
7. for(k=0;k<sign;k+=1)
   1. i=k*numprocs+myid;
   2. for(j=numItem-i-1;j>=0;j-=1)
      1. prune(root+k,j,numItem-i-1);//prune infrequent nodes in HPFP-Tree
      2. merge(root+k,numItem-i-1);//merge pruned HPFP-Tree
      3. Fminer(root+k,numItem-I,FPT ,0);
   3. delete_tree(k,numItem-1);//release memory

Algorithm 2 is a parallel algorithm running in a distributed memory parallel environment. In this algorithm, the viable processors (except processor 0) accept the combinations assigned from processor 0, and then contribute HPFP-Forest and mine frequent pattern on their HPFP-Forest independently. The prune part of the algorithm is similar to the prune algorithm of LPS-Miner in [11].

Algorithm 3: Fminer
Input: root node of mined HPFP-Tree, height of the tree num, initial mode FPT, length of mode length
Output: all frequent pattern of this HPFP-Tree

procedure:

1. FPT[length++]=root->item;
2. if(isSingle(root,num,P))//if there is single frequent branch P
   1. for each subpath Y of P
      1. L=FPT U Y;
      2. MPI_File_write_shared(file,L,length_L,MPI_INT,status);

Else

1. for(j=num-1;j>=0;j-=1)
   1. build_newtree(newroot,root,j);
   2. merge(newroot,j-1);
   3. Fminer(newroot,j,FPT,length);
   4. Delete_tree(newroot,j);

Algorithm 3 is a recursion algorithm. If the HPFP-Tree contains a single-frequent-branch P, enumerate all the combinations of the subpaths (which must include the root node) of P, and combine these combinations with the existing pattern (the initial existing pattern is null) to generate new pattern. The shared file pointer mode is used to assure the mining result from many processors exporting in the same file. While the HPFP-Tree does not have any single-frequent-branch, a new HPFP-Tree will be built and mined by Fminer again.

B. Single Frequent Branch

In the algorithm 3 we mentioned a term called single frequent branch, now we give its definition.

Definition 2: If a HPFP-Tree has no branches, i.e. for each node N in the tree, it has at most one child node, and then we can say the HPFP-Tree contains a single-frequent-branch. The branch goes from the leaf node to the root node of the HPFP-Tree.

If an HPFP-Tree contains a single-frequent-branch P, then mining this tree will become very easy. We can enumerate all the combinations of the subpaths (which must include the root node to assure the discovered frequent itemsets are not repetitive or omitted) of P, and combine these combinations with the pre-existing pattern. The other nodes in the HPFP-Tree are no need to consider.

V. EXPERIMENTAL ANALYSIS

In this section, we compare the performance of HPFP-Miner algorithm with other two frequent pattern mining algorithms mentioned in previous sections, FP-growth* and F-Miner. The code of FP-Growth* is downloaded from the web of international authoritative FIMI (Frequent Itemset Mining Implementations Repository) http://fimi.cs.helsinki.fi/.

All the experiments have been conducted on a Downing 4000L platform at Gansu Computing Center. The platform contain 48 2.2GHz CPUs and 24 2.2GHz Core 2CPUs. It has 14 146G Ultra30 SCSI hard disks. The proposed algorithm has been implemented using GCC version 3.4.3 and MPI library mpich2-1.0.7.

The experiment chooses two dense datasets accidents and connect, and two sparse datasets BMS-POS and BMS-WebView-1. All the datasets can be downloaded from the website of FIMI. Accident dataset is a real traffic accident
dataset includes statistical traffic accident data of Flanders of Belgium from 1991 to 2000. Connect dataset contains game data collected from UCI machine learning workgroup by Roberto Bayardo. BMS-WebView-1 dataset is a representation of sparse transaction datasets, and it is derived from visiting data of a real E-business website. BMS-POS is a real world datasets from several years of point-of-sale data.

Figure 2 shows the running time of the three compared algorithm on datasets accidents, connect, BMS-POS and BMS-WebView-1 separately. From these four figures, we can see the performance of the proposed algorithm is much better than existing algorithms. On the first two datasets, with support threshold decreasing, the gaps between algorithms increase. The running time of HPFP-Miner clearly outperforms than other two algorithms. The last two dataset, we choose low support, and the results are clearer. HPFP-Miner is treble faster than F-Miner, while FP-Growth* is the slowest.

In the experiment, we only have three viable processors, and HPFP-Miner will run faster with viable processors increasing.

VI. CONCLUSIONS

We have presented two data structures named HPFP-Tree and HPFP-Forest, which both have been used in our new algorithm named HPFP-Miner for mining complete frequent patterns. HPFP-Tree compresses its node greatly by eliminating the parent-to-child pointer, which reduces the cost of accessing the tree significantly. We also have proposed an efficient scalable parallel algorithm HPFP-Miner for mining frequent patterns on multiprocessor systems. The proposed algorithm has little communication overheads by efficiently partitioning the list of frequent elements list over processors. In this parallel algorithm, we use one processor (processor 0) to take charge of scanning the whole database and assign transactions to the rest viable processors. The rest processors build HPFP-Forest and discover the frequent itemsets. The results of experiment show that HPFP-Miner has good scalability and performance.

ACKNOWLEDGMENT

We gratefully acknowledgement supports by research for high performance computing of spatial data mining from Gansu Computing Center.

REFERENCES


