An Efficient Method for Mining Distributed Frequent Itemsets: MDFI

Houda Essalmi, Anass El Affar

Abstract: Discovering frequent itemsets is an interesting problem in the context of parallel and distributed databases. Computation cost and communication/synchronization overhead are important elements in the distributed domain. In this work, we propose an efficient algorithm for mining distributed frequent itemsets (MDFI) which can significantly reduce the number of candidate itemsets and communication costs by adopting a Master/Slaves scheme of communication. We present performance comparisons for our algorithm against Apriori and FP-growth algorithms using two datasets with different minimum support.

Keywords: Distributed Datasets, Frequent Itemsets, Apriori Algorithm, Support, Performance.

1. Introduction

Finding association rules in large transactional databases is a core problem in the field of knowledge discovery and data mining; it aims to find the patterns of co-occurrences of attributes in a database (Agrawal et al., 1993). A procedure of mining frequent itemsets is a principal step and most expensive of association rules discovering introduced in (Agrawal et al., 1993). Apriori (Agrawal and Srikant, 1994) is one of the most famous and effective algorithms for finding frequent patterns which its ultimate goal is to discover the Itemsets, whose frequency of occurring in a transactional dataset is greater than a specific threshold. The FP-growth (Han et al., 2000) is the improvement of Apriori algorithm which was employed the pattern growth method to discover frequent Itemsets without using generation of candidates Itemsets.

Due to the increase in data volume and the computational power required, algorithms of searching frequent itemsets have proven to be ineffective, thus the introduction of new parallel versions has become compulsory. Current parallel and distributed algorithms are based on the Apriori sequential algorithm. The CD (Count Distribution) (Agrawal and Shafer, 1996) algorithm is the basic algorithm of data parallelism; it’s a simple parallelization of Apriori; this algorithm underestimates communications between sites because only local supports of candidates Itemsets are exchanged at each iteration on different sites. The PDM (Parallel Data Mining) (Park et al., 1995), FDM (Fast Distributed Mining) (Cheung et al., 1996), NPA (Non-Partitioned Apriori) (Shintani and Kitsuregawa, 1996) algorithms are all similar and offer CD enhancements using either hashing techniques or candidates pruning techniques. The basic DD (Data Distribution) algorithm (Agrawal and Shafer, 1996) of task parallelism, the database partition on each site is issued to all other sites. However, to calculate the support each site must browse the entire database (its local partition and all remote partitions) in all iterations. HPA (Hash-based Parallel mining of Association rules) (Shintani and Kitsuregawa, 1996) and IDD (Intelligent Data Distribution) (Han et al., 1997) algorithms are similar to DD except that HPA carries out the overall reduction on a master site, and IDD performs partitioning of basic prefixes candidates. There are other CAD (Candidate Distribution) (Agrawal and Shafer, 1996) type algorithms exploiting the extraction domain in the sense that it distributes the transactions and the candidates based on their prefix so that each site can proceed independently of the others.

A number of research works have explored to address the problem of frequent itemsets mining in parallel and distributed environments. We can mention some works: The researchers in (Kung and J, 2020) have proposed a distributed and shared-memory parallel algorithm for frequent itemsets mining operating in Master - Slaves model named DP3 which is based on the state-of-the-art serial algorithm PrePost+ (Deng and Lv, 2015), they have used a tree structure named FPO for organize the mining result of frequent itemsets which provide the optimal compactness for light data transfers and the highly efficient aggregations with pruning ability. (Sawant and Shah, 2018) have provided an efficient system that implements data distribution by performing horizontal fragmentation using web-based framework. It generates frequent itemsets by Apriori algorithm in a distributed environment on original data and incremental data which reduces execution time and computational costs. The authors in (Goyal et al., 2018) have developed a new framework and an algorithm that efficiently generate and scan candidate Itemsets for the neighboring sites, the overhead of each site is reduced but a site does not scan candidate k-Itemsets of the neighboring site in the distributed database. (Vasoya and Koli, 2016) have suggested a hybrid architecture where the entire database divided into several clusters of variable sizes, and then each cluster is converted into a matrix by matrix algorithm in the slave system and generates frequent itemset from each cluster in less time.

However, most of the algorithms proposed usually have a high number of data scans and multiple synchronization and communication stages which degrades their performance. Our main contribution aims to
introduce a new algorithm called MDFI (Mining Distributed Frequent Itemsets) to extract the frequent Itemsets in distributed contexts. Our purpose is to obtain a valid result on all the data by minimizing the communications/synchronizations overhead required between different sites in the case where the database is distributed between these sites, which reduces the number of candidates generated and the communication cost (the amount of exchanged messages) which is an important factor in measuring their performance.

The organization of this paper is as follows: Section 2 presents the communication scheme of our approach. The methodology of the proposed work is given in section 3. The experimental results of our algorithm are described in section 4. We finally conclude the paper in conclusion section.

2. Communication Scheme

2.1 Reduce the Inter-sites Communication Cost

The use of distributed parallel architectures suffers from an overhead in inter-site communications among processors during frequent itemsets generation (Tseng et al., 2010) on the other side, the use of Master/Slaves scheme reduces significantly the number of communications/synchronizations required for the distributed computation of global frequent Itemsets, for instance in work (Vasoya and Koli, 2016) the proposed system including scheme Master/Slaves gives better time complexity and space complexity, at first the Master processor partitions the whole database into different clusters and distributes the clusters to Slave processors (Tassa, 2013). Each Slave generates frequent itemsets using improve Apriori algorithm and submit frequent itemsets to the Master processor.

We assume that the database is fragmented horizontally between P sites. Let $C_K$ be the number of candidates Itemsets at pass K. Therefore, the algorithms that follow the scheme of broadcast communication requires at each pass k that each site Pi broadcast the local support calculated in site Pi to all the other sites (Agrawal and Shafer, 1996), for example The CD algorithm requires $(P_i - 1 * |C_K|)$ communication overhead at each iteration k (Ashrafi et al., 2004). In the case of Master/Slaves; At each iteration k, the P Slaves sites send a message to the Master site and the latter responds with a message to all the P Slaves sites. Thus, automatically we can deduce that the total number of message broadcasting size in the Master/Slave scheme is lower compared to the scheme of broadcast communication.

2.2 Elimination of Redundant Calculations in the Candidate’s Generation Phase

Distributed algorithms for extracting frequent Itemsets that use broadcast to exchange messages suffer a redundant calculation problem of candidates at each site. At each iteration, the same set of frequent Itemsets is found and the same redundant calculation of their global Supports is done in each site.

We opt in our approach to centralize the phase of generation of candidates Itemsets at the Master site. This allows us to avoid redundant calculations of candidates Itemsets in each Slave site and to make the efficient use of the global resources of the distributed system. The Master site will be only responsible for the generation of global candidates Itemsets and their distribution to all of the Slaves sites.

3. Methodology

3.1 Proposed MDFI (Mining Distributed Frequent Itemsets) Algorithm

Let I be a set of items and DB database of transaction. In a distributed context, the DB is partitioned in $\{DB_1, DB_2, ..., DB_P\}$ and distributed through the P sites $\{S_1, S_2, ..., S_P\}$. Let D be a size of the DB database and the $d_i$ a size of the partition $DB_i$. We illustrate below an example of a database $\beta$ presented in the table 1 for an alphabet I= {a, b, c, d, e}, (m=5 element). The database is fragmented and distributed on 2 Slaves sites.

<table>
<thead>
<tr>
<th>Tid</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>acd</td>
</tr>
<tr>
<td>2</td>
<td>bce</td>
</tr>
<tr>
<td>3</td>
<td>abce</td>
</tr>
</tbody>
</table>
Houda Essalmi⁴, Anass El Affar⁵

For a given Itemset Y, let \( \text{Supp}(Y) \) and \( \text{Supp}_i(Y) \) be a Support of Y in DB and DB₁ respectively with \( \text{Supp}(Y) = \frac{\text{card}(Y)}{T_G} \) where \( T_G \) is the set of transactions in DB, and \( \text{card}(Y) \) is the number of transactions in \( T_G \) containing Y, \( \text{Supp}_i(Y) = \frac{\text{card}(Y)}{T_i} \) where \( T_i \) is the set of transactions in DB₁, and \( \text{card}(Y) \) is the number of transactions in \( T_i \) containing Y. In this case, \( \text{Supp}(Y) \) is called global support and \( \text{Supp}_i(Y) \) local support of Y on the site of \( S_i \). For a given minimum support threshold \( \text{Sup}_{\text{min}} \), Itemset Y is globally frequent if \( \text{Supp}(Y) \times D \geq \text{Sup}_{\text{min}} \times D \). Similarly, Itemset Y is locally frequent in the site \( S_i \) if \( \text{Supp}_i(Y) \times d_i \geq \text{Sup}_{\text{min}} \times d_i \).

Our proposed algorithm MDFI (Mining Distributed Frequent Itemsets) for the distributed frequent Itemsets is essentially based on the Apriori sequential algorithm, it is composed of 3 steps:

**Step 1 Construction of the CountList Structure**

The CountList structure is a 2-dimensional matrix size \((m \times m)\). The \( m \) represents the number of database items. The CountList matrix represents a projection of the database. Each cell \((i, j)\) corresponds to the frequency of the Itemset composed of elements \( y_i y_j \). The cells represent the frequencies of 1-Itemsets (the cells of the diagonal) and 2-Itemsets (the higher cells than the diagonal of the matrix) with minimum support \( \text{Sup}_{\text{min}} = 2 \).

At the implementation level of the CountList, the Itemsets are ordered in lexicographic order, we eliminate redundant (symmetric) elements and therefore its size is reduced to \( \frac{1}{2}(m^2+m) \) as illustrated in the following example:

![CountList optimized](image)

**Figure 2. CountList optimized**

The CountList structure is built by browsing the \( \beta \) database for each transaction, incrementing the frequency of the 1-Itemsets and the different 2-Itemsets contained in the transaction.

\[
\text{Support}(y_i) = \frac{\text{CountList}(i,1)}{n}, |y_i| = 1, \text{Support}(y_jy_i) = \frac{\text{CountList}(i,j-(i-1))}{n}, |y_jy_i| = 1, n=|\beta|.
\]

In our approach each Slave site calculates its local CountList structure (CountList\(_L\)) by browsing its local database portion. Then, we calculate the supports of the 1-Itemsets by a simple access to the CountList\(_L\) and we eliminate those that are less than \( \text{Sup}_{\text{min}} \). The 2-Itemsets candidates are then generated from these frequent 1-Itemsets. It should be noted that the calculation of frequent 1-Itemsets and frequent 2-Itemsets requires only one run of the local database instead of two runs for its Apriori algorithm. Subsequently, the Slaves sites send the CountList\(_L\) content to the Master site. This latter calculates the global support in CountList\(_G\) structure by merging
(the sum of the values) all the CountList, received from different Slaves sites as illustrated in the figure 3 below in the case of a Master site and 2 Slaves sites.

At the end The Master site deduces the list of global frequent 1-Itemsets and frequent 2-Itemsets from CountListG without access to databases and therefore without communication with the Slaves sites. List of 1-Itemsets= {a :3, b :5, c :5, e :5}, List of 2-Itemsets= {ab :2, ac :3, ae :2, bc :4, be :5, ce :4}.

Step 2 Extraction of Frequent K-Itemsets (K ≥ 3) Global

The Master site iteratively builds the list of global candidates k-Itemsets (K ≥ 3) by introducing a graphical structure that facilitates the search for global frequent k-Itemsets (K ≥ 3).

To illustrate the process, we use an example with the transactions indicated in table 1 above with a Master site and 2 Slaves sites. The graphical structure is initialized first by the list of global frequent 2-Itemsets: {ab, ac, ae, bc, be, ce} ordered lexicographically as nodes at level 1 of the graphical structure. Each node contains the frequent element with its support. At level 2, we build the global 3-Itemsets candidates by self-joins the global 2-Itemsets of level 1.

The 2 nodes ab and ae having (k-2) prefixed items in common so an abc node is formed. A link is established between the 2 global frequent nodes and the newly formed abc node, the minimum support of the 2 global frequent nodes is assigned to the abc node and considered as an approximate value of the global real support of the Itemsets abc. The 2 nodes ab and ae share the same first item a, therefore an abe node is formed with an established link between the 2 nodes and abe, its approximate support is the minimum support of the 2 nodes ab and ae. The ab node does not have the same prefix with the bc node and be node, similarly, the first item of ab node and ce node are different, so the graphical structure is not able to generate new nodes. The same process is continued with the other nodes {ac, ae, be, be, ce} until all the frequent nodes are obtained in level 2{abc, abe, ace, bce} as shown in the figure 4 below.

At level 3, to find the 4-Itemsets nodes, we merge the abc and abe global frequent nodes of level 2 having (k-2) the same ab prefix, so a first abce node is formed, its approximate support is the minimal support of the abc and abe global frequent nodes. The 2 global frequent nodes abc and ace do not contain the same (k-2) items in common. A link does not make between nodes abc and ace. The same procedure is continued until the graphical structure not be able to generate other nodes.

At the end a list of K-Itemsets global candidates (K ≥ 3) is generated with approximate values of their supports without any exchange made with Slaves sites.

Step 3 Refining Global Frequent Itemsets

At this algorithm stage, a validation phase must be performed to refine this set of global frequent Itemsets.
The Master site sends to the Slaves sites the list of the K-Itemsets (K ≥ 3) candidates built in the previous step, then the Slaves sites calculate the real supports of the K-Itemsets received by browsing their local databases portions and return the results to the Master site.

The Master site determines the global frequent K-Itemsets by eliminating the non-globally frequent. It is noted that the sets of 1-Itemsets and 2-Itemsets are not included in this process because their real supports were calculated in step 1 of the algorithm via the global CountList structure.

In the case of the Apriori algorithm with a Master site and 2 Slaves sites, at the first iteration, the 2 Slaves sites calculate the local supports of 1-Itemsets candidates and send these calculators to the Master site which merges the candidates received P1 and P2 and determines the 1-Frequent Itemsets, then at the second iteration the 2 Slaves sites calculate the 2-Itemsets candidates and return them to Slaves sites. This process is repeated for iteration 2, 3 and 4. In this case, the Apriori algorithm performs 4 iterations for the calculation of frequent Itemsets. Therefore, there will be 4 communications phases between the Master site and the Slaves sites.

Our MDFI algorithm requires 2 accesses to the database one access in step 1 and another in step 3 so our algorithm requires 2 exchanges between sites instead of 4 in the case of Apriori algorithm. The utility of the graphical structure in step 2 enables the reduction of the number of candidates Itemsets generated and thus, the cost of calculating frequent Itemsets on one hand and also the reduction of the number of access to local databases of sites Slaves for the calculation of the real supports of the local candidate Itemsets on the other hand.

3.2 Process of MDFI Algorithm

Distributed Frequent Itemsets are generated using the proposed Algorithm MDFI. The methodology of the proposed work is given below in figure 5.

![Figure 5. Architecture of the Proposed MDFI](image)

The Process of our algorithm MDFI for Distributed frequent Itemsets show below:

- Fragmented horizontally the database by distinguishing the Master site and several Slaves sites;
An Efficient Method for Mining Distributed Frequent Itemsets: MDFI

- Calculate the local support 1-Itemsets and 2-Itemsets by the CountList structure of each Slave site and eliminate those which are inferior to $Sup_{min}$;
- Send the content of the local CountList to the Master site;
- Calculate the global CountList structure by the sum of the local CountList structures;
- In the list of frequent 2-Itemsets extracted from the CountList, find the frequent k-Itemsets ($K \geq 3$) based on a graphical structure constituted initially of an ordered set of global frequent 2-Itemsets;
- Calculate the approximate support for each candidate in the list of frequent K-Itemsets ($K \geq 3$) found at each level of the graphical structure already mentioned in the step 2 of our algorithm;
- Send the list of frequent k-Itemsets ($K \geq 3$) generated to all Slaves sites;
- Each Slave site calculate the real local supports of the candidates k-Itemsets received;
- Return the results to the Master site to determine the frequent k-Itemsets, which must be superior to $Sup_{min}$;
- Extract the distributed frequent Itemsets.

4. Experimental Results

In this section, the datasets T40l10D100K and Chess are used for experiments to evaluate the performance of MDFI algorithm. These datasets are available at the FIMI (Goethals & Zaki, 2003) (Fournier et al., 2014). The description of the selected data sets is shown in table 2:

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Items</th>
<th>Transactions</th>
<th>Average length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chess</td>
<td>75</td>
<td>3196</td>
<td>37</td>
</tr>
<tr>
<td>T40l10D100K</td>
<td>1000</td>
<td>100,000</td>
<td>40</td>
</tr>
</tbody>
</table>

The experiments are carried out in a local network. The databases are fragmented horizontally and distributed on the Slaves sites. The result performed using Intel® Core (TM) i7 CPU with the speed of 2.80GHZ with 4G of RAM, running Windows 10. We performed tests on a local network composed of a Master site and a number of Slaves sites respectively of three (03), five (05) and seven (07) Slaves sites. We have implemented and evaluated MDFI algorithm in java programming language and tested using the NetBeans IDE.

We compare the performance against the algorithms Apriori and FP-growth by applying the scheme of Master/Slaves. Apriori and FP-growth are implemented in java by researchers in (Fournier et al., 2014). The following figures describe the running time with various support degree for datasets T40l10D100K and Chess. The x-axis denotes the minimum support and y-axis represents the execution time.

Figure 6. Runtime evaluation for Chess datasets

The results achieved with the Chess database containing a reduced number of transactions show that the MDFI algorithm surpasses the Apriori and FP-growth algorithms especially for low Support values. This is due to the reduced number of candidates Itemsets generated by the MDFI algorithm compared to the Apriori and FP-growth
algorithms, which directly affects the cost of communication where the volume of messages exchanged between sites is reduced compared to 2 Apriori and FP-growth algorithms. Besides, the MDFI algorithm requires at least one iteration than the Apriori and FP-growth algorithms and therefore one less communication phase.

Subsequently, we further increase the size of the given database to test the scalability of the MDFI algorithm, the figure 7 above shows the results obtained, we found a better performance of the MDFI algorithm compared to Apriori and FP-growth algorithms. This improvement is in fact due to the reduced number of communications phases required for the MDFI algorithm for the calculation of the global frequent Itemsets. We also notice that the results achieved by the algorithms tend to come closer to each other in the case where the number of sites (Slaves nodes) is high (the case of the use 07 Slaves nodes). This is mainly due to the fine granularity of the data distribution which resulted a costly communications overhead for calculation of the global frequent Itemsets.

The experiments carried out show that the MDFI algorithm is more efficient and more scalable than the Apriori and FP-growth algorithms. The algorithm registers a good acceleration with the increase in the size of the database and the increase in the number of nodes and also with the increase in the number of transactions. In the case of fine granularity of parallelism (number of nodes), the MDFI algorithm approaches the performance of the Apriori and FP-growth algorithms.

5. Conclusions
In this paper, a new distributed algorithm named MDFI, is proposed to mine all frequent Itemsets in distributed datasets. This algorithm aims at reducing the computation overhead of discovering frequent Itemsets by 3 phases using the scheme of Master /Slaves: the construction of CountListG structure for the generation of candidates 1-Itemsets and 2-Itemsets and the graphical structure for the generation of candidates k-Itemsets (K ≥ 3). A validation step is necessary to refine all the candidates generated. The experimental evaluation on different horizontally divided datasets with various minimum support values show that the proposed algorithm achieves good performance and high scalability in terms of the required communication and computation costs compared to Apriori and FP-growth algorithms.

References