Mining Distributed Frequent Itemsets Using a Gossip based Protocol

Maryam Bagheri, Seyed-Hassan Mirian-Hosseinabadi, Hoda Mashayekhi, Jafar Habibi
Department of Computer Engineering
Sharif University of Technology - Tehran, Iran
ma_bagheri@ce.sharif.edu, hmirian@sharif.edu, hmashayekhi@ce.sharif.edu, jhabibi@sharif.edu

Abstract—Recently, there has been a growing attention in frequent itemset mining in distributed systems. In this paper, we present an algorithm to extract frequent itemsets from large distributed datasets. Our algorithm uses gossip as the communication mechanism and does not rely on any central node. In gossip based communication, nodes repeatedly select other random nodes in the system, and exchange information with them. Our algorithm proceeds in rounds and provides all nodes with the required support counts of itemsets, such that each node is able to extract the global frequent itemsets. For local iteration and generation of candidate itemsets, a trie data structure is used, which facilitates the process and reduces execution time. We further propose an improvement to our algorithm by grouping nodes and arranging them into a hierarchical structure. By performing aggregation tasks in groups, communication overhead is effectively reduced. We evaluate our proposal using simulation, and show advantages of our algorithms in reducing execution time and communication overhead, while preserving accuracy.

Keywords-data mining; frequent itemset mining; gossip; trie

I. INTRODUCTION

Finding frequent itemsets is one of the most investigated fields of data mining. Many data mining techniques such as correlation induction, association rule learning, classification, and clustering, employ frequent itemsets to accomplish their goals. The ultimate goal of frequent itemset mining is to discover the itemsets, whose frequency of appearance in a dataset is greater than a specific threshold. Available studies on frequent itemset mining can be classified into two general categories. Apriori techniques [11] and FP-Growth [12].

In distributed systems, however, gathering data at a central site is infeasible due to large communication overhead, scalability and privacy issues. In an alternative approach, nodes can communicate with each other to execute the mining algorithm in a distributed manner. This problem has been addressed by some researchers and a number of distributed algorithms have been proposed [16, 1, 13].

One of the major approaches towards distributed data mining is a two phase approach. In the first phase, nodes compute local models from their data, and in the second phase, these local models are combined to produce the global result. In distributed frequent itemset mining, each node first calculates the frequencies of its local data. Afterwards, nodes communicate with each other to aggregate local information and obtain globally frequent patterns. It is therefore very crucial to combine local results in a communication effective way. On the other hand, scalability requirements inhibit any dependency on a central server.

In this paper we aim to find all frequent itemsets in the data stored in a distributed system. To accomplish this goal, we propose a gossip based approach to execute the Apriori algorithm in a distributed manner. Following the two phase approach introduced above, our algorithm consists of iterations on two subsequent local and global steps. In the former step, we propose a novel method to facilitate computation of local frequencies, based on the trie data structure. The second step involves a gossip phase to induce the global frequent patterns.

Gossip based algorithms employ randomized communication, thus our algorithm ensures robust information dissemination in gossip phase, during data aggregation. Up to now, most works for mining association rules and frequent patterns have focused on a single node or traditional client-server models. In addition many of the existing parallel implementations are targeted to shared memory environments, or otherwise, pose heavy data exchange among nodes when average length of transactions is long or the size of database is large. Our method does not require a central server and addresses the communication efficiency and scalability requirements. Simulation results show the effectiveness of our method. To the best of our knowledge, this paper is the first work for identifying frequent itemsets using gossip in a distributed environment.

II. RELATED WORKS

Apriori algorithm is the most well-known association rule mining algorithm. After Apriori, many proposals, such as AprioriHybrid and AprioriTID [7] have extended the Apriori algorithm. Khairuzzaman et al. [15] propose a tree structure, called PP-tree (Parallel Pattern tree), which captures the database contents with a single scan and performs the FP-growth mining on it. This parallel algorithm works independently at each local site and locally generates frequent patterns which are merged at the final stage. In [2] a parallel data mining algorithm in a distributed peer-to-peer network is designed which focuses on a loop architecture. For the first time, [6] provides a formal method of gossip protocols for data dissemination. Kempe et al. [9] propose some algorithms for computing aggregate functions such as sum, average, approximate uniform random sample and quantiles using uniform gossip.
To the best of our knowledge, [10] is the only available proposal on identifying frequent items in a large distributed dataset using gossip. This algorithm works by layering a small space “sketch” of data using a gossip-based data dissemination mechanism. In this proposal every node holds a single item, and nodes gossip to find the global frequency of each item. Our proposal, in contrast, is a general algorithm for discovering frequent itemsets of any size.

III. PROBLEM DEFINITION

The frequent itemset mining problem is defined as follows: Let \( S = \{s_1, s_2, ..., s_k\} \) denote items in a certain domain. An itemset is some arbitrary subset \( X \subseteq S \). A transaction, \( t \), is also a subset of \( S \) associated with a unique transaction id. A database \( DB \) is a list of transactions, and its size is presented as \( |DB| \). Given an itemset \( X \) and a database \( |DB| \), support of \( X \) in \( DB \) is defined as follows:

\[
Sup(X) = \left| \{(t \in |DB|) \mid X \subseteq t\} \right|.
\]

In other words, \( Sup(X) \) is the number of transactions in \( DB \) which contain all the items of \( X \). In a distributed system, assume we have \( n \) nodes. The database \( DB \) is horizontally divided into \( n \) partitions denoted as \( db_1, db_2, ..., db_n \), and each partition \( db_i \) is assigned to a node \( i \). For an itemset \( X \), \( Sup_i(X) \) shows the support count for \( X \) in \( db_i \). For a given minimum support threshold, \( min_{\text{sup}} \), \( X \) is globally frequent (i.e., frequent) if \( Sup(X) \geq min_{\text{sup}} \times |DB| \). The frequent itemset mining seeks to discover the complete set of frequent itemsets in \( DB \), with regards to a predefined minimum support threshold.

IV. PREREQUISITES

In this section we describe the trie data structure and gossip protocol, which are necessary to understand our proposal.

A. Trie based Apriori

Bodon [3], proposed to use a trie data structure, implemented by means of a hash tree, to improve Apriori performance. He proposed a fast Apriori implementation using the trie data structure instead of a hash tree which was used in the classical approaches. We will adopt this idea that was used in central environment, in our distributed model.

A trie is a rooted tree in which each edge is labeled with an item. Each tree node represents itemset made by concatenating edge labels on the path from root to that node. Each tree node also has two links to its immediate children, and stores the support count for the itemset it is representing.

The idea of trie data structure is that all itemsets sharing a common stem or prefix, hang off a common node. The elements in an itemset can be recovered in a scan from the root to the leaf. All itemsets in the trie can be recovered by a depth-first scan of the tree.

In order to calculate support counts of itemsets, the following procedure is taken: for each transaction \( t \) in the database, take all \( k \)-subsets of \( t (k \in \{1,2, ..., t\}) \), and search for them in the trie structure. The value of the tree node which represents each subset will be incremented if trie contains the itemset. Eventually, nodes which have support counts below the minimum support will be pruned.

B. Gossip based Protocols

These protocols operate on unstructured networks and have no overhead of sub-network formation and maintenance. Gossip protocols usually perform in rounds. In each round, each node communicates with a random other node. This random selection is based on uniform random sampling over the entire set of nodes [8]. A node \( u \) is said to call a node \( v \) if \( u \) chooses \( v \) as a communication partner. Once a call is established, we assume that information can be exchanged in both directions. Gradually all nodes will conclude the global result. Random selection of nodes will ensure robustness and fast convergence. Gossip algorithms can be used as potentially effective solutions, as they are inherently scalable, easy to deploy, robust, and resilient to failure. Gossip protocols generally consist of active and passive threads. In the active thread, a node selects another random node and initiates a connection with it. While in the passive thread, the node is contacted by another node wishing to establish a connection. The two threads collectively consist of three main parts as follows [14]:

- **Node Selection**: returns a node from the system. This function is used to select the gossip target.
- **Data Exchange**: returns the data to be exchanged over a gossip communication.
- **Data Processing**: specifies the processing that should be performed on the exchanged data and the resulting state.

V. DISTRIBUTED FREQUENT ITEMSET MINING

In this section, we describe our algorithm for mining frequent itemsets from a distributed system. We consider a distributed architecture and divide the database into \( n \) \((n \text{ is the number of nodes)}\) non-overlapping partitions and assign each part to a node. Our algorithm executes in rounds and each round consists of two phases: local frequency computation and gossip-based aggregation. In the first phase of round \( k \), nodes compute local support counts for candidate itemsets of size \( k \). The second phase is mainly an aggregation phase in which nodes combine local frequencies of candidate \( k \)-itemsets using gossip, and conclude global frequencies. At this stage each node can decide on frequent \( k \)-itemsets before entering the next round. In the two subsequent sections we will describe each of these phases.

A. Local Frequency Computation

This phase executes completely locally in each node. The main goal of this phase is to compute local candidate itemsets along with their local support counts. In round \( k \) of the algorithm, each node seeks to find the frequent \( k \)-itemsets. Hence, in local frequency computation phase of round \( k \), each node uses the frequent \((k-1)\)-itemsets from the previous round to generate local candidate \( k \)-itemsets. These itemsets are extracted from local databases, means they only contain items from local transactions. Next, the local support for each candidate itemset is computed.
To initiate the algorithm in round 1, each node simply computes the local support for each local item. This bootstrap information is the feed for the first aggregation phase which will proceed to find the frequent 1-itemsets. The subsequent rounds follow as described.

B. Gossip based Global Aggregation

The main aim of this step is to aggregate the local support counts to produce global supports for each itemset. At the end of this phase, each node will be aware of these global values and is able to produce frequent itemsets of a particular size. Thus this phase consists of a gossip step followed by a local pruning operation.

In round \(k\), nodes will decide on global frequency counts for itemsets of size \(k\). After gossiping is over, nodes will have the desired information to prune local data from infrequent itemsets and enter the next round. The detailed gossip algorithm is presented in fig. 1.

During gossip in round \(k\), every node will continuously communicate with other random nodes, until all nodes agree on support counts of candidate k-itemsets. The global support of an itemset \(X\) which is gradually updated in node \(v\), is denoted by \(V_v(X)\). Before a node starts the gossip phase we have: \(V_v(X) = \text{Sup}_v(X)\). As described before, \(\text{Sup}(X)\) is the number of transactions which contain all the items of \(X\). As the gossip proceeds, \(V_v(X)\) will eventually approach \(\text{Sup}(X)\).

During each data exchange of two nodes \(v\) and \(u\), the local k-itemset lists if the two nodes will be compared. If a common candidate itemset \(X\) is discovered, \(V_v(X)\) and \(V_u(X)\) are exchanged and updated at each node. Hence, the two nodes should exchange the k-itemsets, along with local support for each itemset.

If nodes simply add the received support count to their locally maintained support, the latter value can increase infinitely as an effect of loops in communication. To avoid fooling nodes into unlimitedly increasing their local support, and achieving correct functionality of the gossip algorithm, each node will additionally maintain a weight attribute for each candidate itemset. The weight parameter for an itemset \(X\), \(W_v(X)\), omits the effect of contacting duplicate nodes, or re-considering previously added support counts for \(X\). After exchanging data with node \(u\), node \(v\) will update \(V_v(X)\) and \(W_v(X)\) for an arbitrary itemset \(X\) as showed in fig. 1 line 8 and 9. At the end of gossiping, \(V_v(X) / W_v(X)\) represents the approximate value of \(\text{Sup}(X)\) in node \(v\).

Having the value of \(\text{Sup}(X)\) for the k-itemset \(X\) in round \(k\), node \(v\) can proceed to prune this candidate itemset if \(\text{Sup}(X) < \text{min}_{\sup}\). Otherwise, it will mark this itemset as frequent and proceed to the next round of the algorithm.

VI. IMPROVEMENTS

In this section we provide some enhancements to the basic distributed frequent itemset mining. The improvements are provided according to individual phases of the algorithm. The first phase essentially involves generating itemsets and computing local supports. We employ the trie data structure to achieve better performance in local computations.

Algorithm: Gossip Aggregation

1: Let \(V_u\) set of \(u\)'s support count in round \(k\)
2: Let \(W_u\) set of \(u\)'s weight in round \(k\)
3: Let \(V_i(X) = \text{Sup}_i(X)\) for \(i \in \{1,2,...,n\}\) and any Itemset \(X\)
4: while (there is more frequent itemsets) do
5:     each node \(u\) independently and uniformly
At random, calls a node \(v \in \{1,2,...,n\}\)
6:     Let be the set of nodes that called \(v\) in round \(k\)
7:     \(u\) and \(v\) compare itemsets. If there was a common itemset, \(X:\)
8:     \(V_v(x) = (V_v(x) + \sum_{u \in E_v} V_u(x))/2\)
9:     \(W_v(x) = (W_v(x) + \sum_{u \in E_v} W_u(x))/2\)
10: end while

The second phase, on the other hand, includes global message passing among nodes in a gossip protocol. Thus, the main goal is to reduce the communication burden of the algorithm in this phase. The optimization technique uses a hierarchical data structure to group nodes and improve the efficiency of global computations.

A. Employing Trie Data Structure

To improve the efficiency of our algorithm we adopt a trie data structure. Every node maintains a trie structure locally. During data exchange among nodes, nodes synchronize their local tries so that at the end of each round, all the nodes have the same data. This similarity involves the trie structure as well as the itemset supports held at each trie node. This is feasible as during the gossip phase, nodes reach a consensus on support count of each itemset. The pruning operation which done immediately after gossiping terminates in each round, is readily performed on the trie structure by removing all tree nodes which represent infrequent itemsets.

The trie data structure is initiated at each node, prior to the algorithm, and updated after each phase of gossiping. Initially the trie contains local 1-itemsets. At the beginning of each round, when candidate itemsets are generated, in local frequency computation phase, the new candidates are added to the trie. The local support counts of these candidates can also be updates as follows: the node traverses the trie recursively for each transaction. When reaching a leaf, if it represents a k-itemset, the support is increased by one. The trie is updated during data exchange between nodes, in the next phase. When gossip terminates, each node is able to traverse the trie and prune infrequent itemsets.

B. Grouping Nodes

Although gossiping has the advantage of avoiding any structure in the system, when a network contains large number of nodes, it may take a long time for the gossip to converge. In many systems, on the other hand, it is feasible to have a partial structure generally in form of a hierarchy. The nodes are arranged into disjoint groups and each group has a central node. Now the distributed frequent itemset...
mining can be modified to execute more efficiently in this hierarchy. Fig. 2 represents an example of grouping nodes. Black and red points correspond to the ordinary nodes in the system, and central nodes of the groups respectively.

After performing phase one of the algorithm in each node, the local candidates and their corresponding supports are sent to the central nodes. The central node of each group aggregates the candidate itemsets and their supports for the nodes located in its group. Next, only the central nodes enter phase two of the algorithm and initiate a gossip to determine global support of each candidate itemset. They are then able to prune the infrequent itemsets, and send the results back to ordinary nodes in their group. The discussed approach effectively reduces number of nodes which are gossiping with each other, and increases the scalability of the algorithm. In [5] a hierarchical architecture is introduced. We use the same idea to group nodes into \( n/\log(n) \) groups each of size \( \log(n) \). To construct the hierarchy, each node can act as a central node with probability \( \log(n)/n \). The central nodes start sending join messages to other nodes. Ordinary nodes acknowledge the first join message received and attach themselves to the corresponding red node.

In [5] it is shown that for grouping nodes into \( n/\log(n) \) groups of size \( \log(n) \), each central node should send \( \mathcal{O}(\log(n) \times \log(\log(n))) \) messages, and at the end of the joining phase, at most \( \mathcal{O}(n \times \log(n)) \) nodes remain ungrouped. These nodes can join one of the existing groups by means of \( \mathcal{O}(\log(n)) \) messages.

VII. EXPERIMENTAL RESULTS

This section presents the simulation results for evaluating our proposed distributed frequent pattern mining algorithm.

We developed a compact simulator in Java to analyze the algorithms. The simulations are executed on a PC with a 2.4GHz processor and 2GB RAM. We have tested our proposal using the Kosarak and Mushroom datasets, from the FIM repository. The details of the two datasets are provided in table 1. As described in previous sections, nodes communicate with each other by message passing. To consider the communication time between nodes in the gossip phase, we made a simplified assumption: transmission of a message from one node to another node is performed on a 100 Mbps link. So the transmission time for a message of size \( |M| \) bytes is equal to \( |M|/(100,000,000 \times 8) \) seconds.

During simulation, each time a message is transmitted between two nodes, its time added to the measured time taken by those nodes. To calculate size of a message in addition to contemplating the message load we consider a 32 bytes header like the TCP protocol. To prove the scalability of our design, we have varied the number of nodes and measured the convergence time, which is the time required for the system to find all frequent itemsets. Fig. 3, shows the results for Kosarak and Mushroom datasets respectively. In every experiment, all the data is distributed horizontally among nodes. As observed when number of nodes is more than 1, no significant change is detected in the required running time of the algorithm. This is primarily because the main bottleneck of the algorithm is the local computations. This is clear from the observation that when number of nodes is one, i.e., the algorithm runs centrally, we have the largest running time. As soon as the data is distributed among nodes, the running time decreases significantly.

When number of nodes increase, the number of local transactions in each node decreases. Therefore, less computation time is required for the local steps of the algorithm. On the other hand, more computation time is required for the gossiping phase, such that the total required time units, remains approximately the same.

The results show the scalability of our system with regard to the network size.

Fig. 3 exposes the effectiveness of employing the trie data structure to improve performance of the algorithms. The trie data structure has an effective role in reducing the comparison time of itemsets, pruning them and calculating the supports. From the figures it is clear that adopting this structure can reduce the computation time, as expected.

To further measure the effectiveness of the trie data structure, fig. 4 compares the execution time of our algorithm before and after using trie. Number of transactions per node is varied from 1000 to 5000. As number of transactions increases, execution time increases very smoothly. Also employing the trie data structure reduces the time needed to execute the algorithm.

One of the improvements to the basic frequent itemset mining algorithm, proposed in this paper, was grouping the nodes into hierarchy. The main aim of this design is to reduce number of nodes which contribute in the gossiping phase, such that the overall performance is increased. Fig. 5 shows the running time of the algorithm before and after applying the grouping strategy. As the diagram presents, grouping leads to reduction of execution time especially when number of nodes is significantly large. When number of nodes is 10000 we see the most difference between the computation time of the basic algorithm, and the modified algorithm with grouping.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Length of each transactions</th>
<th>Number of transactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>KOSARAK</td>
<td>Variant : 1to 660</td>
<td>999000</td>
</tr>
<tr>
<td>MUSHROOM</td>
<td>Constant : 23</td>
<td>8124</td>
</tr>
</tbody>
</table>

TABLE 1. DETAILS OF THE TEST DATASETS
This is predictable as grouping has a prominent effect on reducing amount of necessary communications in the gossiping phase.

Fig. 6 shows how the accuracy of the obtained results varies against the final collection of frequent itemsets, as number of rounds increases. The accuracy measures the number of correctly identified frequent itemsets per number of itemsets, averaged across several executions of our implementation.

The algorithm can achieve high accuracies in some initial rounds which is sufficient for many applications. However to discover all frequent itemsets of larger sizes, it should continue executing for more rounds.

The results show the effectiveness of our algorithm in achieving high accuracy in acceptable running times.

Fig. 7 and 8, compare exchanged messages in our algorithm and DTFIM [17]. DTFIM is a distributed trie-based frequent itemset mining, which uses a trie data structure to find frequent itemsets. The main difference between the algorithm explained in this paper and DTFIM is employing gossip as the communication mechanism. In DTFIM every node must synchronize their data so that all local copies are the same at the end of each stage. This synchronization simply done by broadcasting local tries between every pair of nodes which performs in O(n^2). While in gossip algorithm random nodes exchange information in some few rounds in O(nlogn).

VIII. CONCLUSION AND FUTURE WORKS

In this paper, we proposed a new gossip based algorithm for efficient distributed itemset mining.

We showed that gossip based communication can reduce the computation overhead of discovering frequent itemsets in a distributed environment.

Our algorithm was showed to be scalable in terms of the required communication and computation costs.
Furthermore, we offered two effective improvements to our algorithm. Using the trie data structure, we facilitated itemset generation and maintenance to improve execution time. Also adopting a hierarchical architecture in the system enhanced the performance of the algorithm when dealing with large number of processing units.

Many large scale distributed systems, such as P2P networks, deal with dynamic datasets. In future we aim to extend our proposed algorithm, to be able to adapt to dynamic datasets in large scale networks.

REFERENCES

[4] Ming-Syan Chen, Jiawei Han, Philip S. Yu,. 1996. Data Mining: An Overview from a Database Perspective, IEEE Transactions on Knowledge and Data Engineering, (December 1996), 866-883.