A Scalable Approach to FIM by Means of MR

Shabnam, Shashikala M K
Dept. of Computer Science and Engineering, VTU,
Karnataka, India

Abstract— Frequent Itemset Mining (FIM) is one of the most well-known techniques to extract knowledge from data. Frequent itemset mining is an exploratory data mining technique that has fruitfully been exploited to extract recurrent co-occurrences between data items. Since in many application contexts items are enriched with weights denoting their relative importance in the analyzed data, pushing item weights into the itemset mining process, i.e., mining weighted itemsets rather than traditional itemsets, is an appealing research direction. Existing parallel mining algorithms for frequent itemsets lack a mechanism that enables automatic parallelization, load balancing, data distribution, and fault tolerance on large clusters. This paper presents a scalable frequent weighted itemset mining algorithm based on the MapReduce paradigm. To demonstrate its actionability and scalability, the proposed algorithm was tested on a real Big dataset collecting approximately 34 millions of reviews of Amazon items. The mined itemsets represent combinations of items that were frequently bought together with an overall rating above average.

Keywords— Big Data, Frequent itemsets, Mining, BigFIM, MapReduce.

I. INTRODUCTION

Due to growth of IT industries, services, technologies and data, the huge amount of complex data is generated from the various sources that can be in various form. Such complex and massive data is difficult to handle and process that contain the billion records of million user & product information that includes the online selling data, audios, images, videos of social media, news feeds, product price and specification etc. The necessity of big data arrives from the worldwide famous companies like Google, Yahoo, Weibo, Facebook, Microsoft, Amazon, Flipkart and Twitter for the reasons of analysis of huge data which can be in unstructured form.

Already from the start, Frequent Itemset Mining (FIM) has been an essential part of data analysis and data mining. FIM tries to extract information from databases based on frequently occurring events, i.e., an event, or a set of events, is interesting if it occurs frequently in the data, according to a user given minimum frequency threshold. Many techniques have been invented to mine databases for frequent events [5], [7]. These techniques work well in practice on typical datasets, but they are not suitable for truly Big Data. Frequent itemsets mining (FIM) is a core problem in association rule mining (ARM), sequence mining, and the like. Speeding up the process of FIM is critical and indispensable, because FIM consumption accounts for a significant portion of mining time due to its high computation and input/ output (I/O) intensity. When datasets in modern data mining applications become excessively large, sequential FIM algorithms running on a single machine suffer from performance deterioration. To address this issue, we investigate how to perform FIM using MapReduce—a widely adopted programming model for processing big datasets by exploiting the parallelism among computing nodes of a cluster.

To support analysts in coping with Big Data there is a compelling need for smart large-scale data mining solutions. Hence, in the last few years an increasing research interest has been devoted to studying and developing supervised and unsupervised data mining algorithms (e.g., classification algorithms [2], and clustering algorithms [3]) that are able to scale towards Big Data. For instance, Mahout [4] and MLlib [5] are notable examples of MapReduce- and Sparkbased scalable machine learning and data mining suites.

Frequent itemset mining is an imperative part of data analysis and data mining. The main goal of FIM is to mine formation and reveal patterns from massive datasets on the basis of frequent occurrence, i.e., an event is interesting or number of events are interesting if it occurs/seems frequently in the data, according to a user given minimum frequency threshold. Many techniques have been invented to mine frequent itemsets from databases. These techniques work well in practice on typical datasets, but they are not applicable for real Big Data. Using frequent itemset mining technique to massive databases is not easy task. There are number of difficulties. First of all, databases having large amount of records do not fit into main memory. In such cases, solution is to use level wise breadth first search based algorithms, such as Apriori algorithm, in this approach frequency counting is getting by reading the dataset over and over again for each size of candidate itemsets. Unfortunately, the memory requirements for handling the complete set of candidate itemsets blows up fast and renders Apriori based schemes very inefficient to use on single machines. Secondly, current approaches tend to keep the output and runtime under control by increasing the minimum frequency threshold, automatically reducing the number of candidate and frequent itemsets.

In this paper, a MapReduce framework is proposed which is basically used for parallel processing of large datasets and it works on key-value pairs. Frequent itemset mining need to calculate support and confidence which can be done in parallel using MapReduce programming model. Faster processing can be achieved by calculating frequency of items using map functions which executes in parallel on set of hadoop clusters and reduce functions used to combine the local frequent items and give global frequent items.
In real-life applications items are unlikely to be equally important within the analyzed data. For example, items purchased at the market have different prices, medical treatments have different urgency levels, and genes are expressed in biological samples with different levels of significance. Hence, an appealing extension of traditional itemset mining algorithms is to use of item relevance weights into the mining process. To allow treating items/transactions differently based on their relevance in the frequent itemset mining process, the notion of weighted itemset has been introduced [15], [16], [17], [18]. A weight is associated with each data item and it characterizes its local significance within each transaction. Since, to the best of our knowledge, existing large-scale itemset mining algorithms are unable to consider item weights during the mining process, the problem of integrating weights into distributed-based algorithms is challenging.

We propose Parallel Weighted Itemset prospector, a new parallel and distributed framework to extract frequent weighted itemsets from potentially very large transactional datasets enriched with item weights [19]. The framework relies on a parallel and distributed-based implementation running on an Hadoop cluster [20]. To make the mining process scalable towards Big Data, most analytical steps performed by the system are mapped to the MapReduce programming paradigm [21].

II. BACKGROUND

Size, complexity and variability of Big Data are big challenges for recognize association rules and frequent itemset mining. —Market –Basket model is best example of association rule which is based on relationship among elements[6]. Association rule mining and frequent itemset mining is well known techniques of data mining. It discovers frequency of items purchased together. The whole database scan is necessary in FIM, it might create challenge when datasets size is scaling, as large datasets does not fit into memory. Several approaches exist for association rule mining [7], [8], [9]. Frequent itemsets play an essential role in finding correlations, clusters, episodes and many other data mining tasks. Value discovered from frequent itemsets can be used to make decisions in marketing.

Agrawal[6] in 1993 first proposed mining customer transaction database item sets problem, now FIM (frequent itemsets mining) has become an essential part of data mining. Most of the current algorithms are classified into two groups: Apriori-like algorithm and FP-growth (Frequent pattern) algorithm. Apriori rejects candidate sets by repeatedly scanning the database. The main advantage of FP Growth algorithm is FP-Tree. When faced with large data, these two algorithms are not well adapted. For the above algorithm, a solution is to consider only the large threshold value, the number of candidates can be reduced and minimized, but this will lead mining association rules out inaccurate due to low utilization data.

The mining of frequent itemsets is a basic and essential problem in many data mining applications. Algorithms for mining frequent itemsets can be basically classified into two types: one is algorithms based on horizontal layout dataset such as Apriori algorithm and FP-Growth algorithm; another is algorithms based on vertical layout database such as Eclat algorithm. Eclat algorithm takes advantage over algorithms based on horizontal layout database. It saves and reduces much time as it does not need to scan the whole database repeatedly.

Apriori is the most classical algorithm in history of data mining, the main idea behind the Apriori algorithm is to generate k+1 - frequent itemsets based on k-candidate itemsets By traversing the database to statistics candidate collection, then by using support threshold value candidate itemsets can be neglected. The pruning strategy of candidate itemsets is that if an itemset is not occurring frequently, then its superset so is. The algorithm is very simple, but main drawback is that Apriori algorithm requires too many times traversing the database and producing a large number of candidate sets, time and memory overhead will become a bottleneck. Comparing with Apriori algorithm, FP-growth is an improved algorithm. The main advantage of FP Growth is that only needs to scan the database twice, and construct a compressed data structure FP-Tree, which reduces the search space, while no candidate set, improved memory utilization. FP Growth adopts to depth-first mode policy. However, it constructs a large number of conditions pattern tree when recursive, when faced with huge amounts of data, the memory is difficult to put all of the pattern tree, and the tree traversal algorithm whose time complexity is higher. FPF is based on the Hadoop (MapReduce Framework) parallel algorithms, FPFF makes groups of the itemsets, as a condition database partitioned and divided to each node, each individual node independently generates the FP-Tree and mines frequent itemsets from individual partitioned database. FPF minimizes the traffic between nodes, increases the degree of polymerization of node. However, algorithm is not efficient if the database is discrete.

Grouping strategy of FPF has problems with memory and speed. To balance the groups of FPF Zhou et al.[10], has proposed algorithm for faster execution using single items which is also not an efficient way. Xia et al. [11], has been proposed Improved FP algorithm for mining frequent itemsets from massive small files datasets using small files processing strategy.

There are number of Hybrid methods are invented for mining frequent itemsets. MRPrePost is hybrid method for frequent itemset mining which combines DistEclat and PrePost algorithm. MREclat is also hybrid method for frequent itemset mining.

ClustBigFIM is modified BigFIM algorithm for generating frequent itemsets which uses parallel Kmeans and Eclat for finding potential extensions and Apriori for producing K-FIs.

a) Literature Survey

Basically, there are three classic frequent itemset mining algorithms that run in single node. Loop is the main logic behind success of Apriori [6] algorithms. In Apriorialgorithm loop k produces frequent itemsets with length k. By using the property
and o/p of k loop, loop k+1 calculate candidate itemsets. Property is: any subset in one frequent itemset must also be frequent. FP-Growth [12] algorithm creates an FP-Tree by two scan of the whole dataset and then frequent itemsets are mined from frequent pattern tree. Eclat[4] algorithm transposes the whole dataset into a new table. In this new table, every row contains list of sorted transaction ID of respective item. In last frequent itemsets are extracted by intersecting two transaction lists of that item.

Othman et al. [15], presented two different ideas for conversion Apriori algorithm into MapReduce task. In first way, all possible itemsets are extracted in Mapping phase, and then in Reduce phase itemsets those does not satisfy minimum support threshold are taken out. In second way, direct conversion from Apriori algorithm is carried out. Every loop from Apriori algorithm is converted into MapReduce task. These presented approaches are used by [13], [14]. In this approaches large data is shuffled between Map and Reduce tasks[15]. To solve these problems, they presented MRApriori algorithm. MRApriori is nothing but MapReduce based improved Apriori algorithm which uses two-phase structure.

Zang et al. [16], presented improved Eclat algorithm to increase the efficiency of FIM from large datasets. Parallel algorithm MREclat based on MapReduce framework is called as MREclat algorithm. MREclat also solves the problems of storage and capability of computation not enough when mining frequent itemsets from large complex datasets. MREclat algorithm has very high scalability and better speedup in comparison with other algorithm. Algorithm MREclat consists of three steps: in the initial step, all frequent 2-itemsets and their tid-lists from transaction database is getted; the second is the balanced group step, partition frequent 1-itemsets into groups; the third is the parallel mining step, the data got in the first step redistributed to different computing nodes according to the group their prefix belong to. Each node runs an improved Eclat to mine frequent itemsets. Finally, MREclat collects all the output from each computing node and formats the final result.

b) Weighted itemset mining

In the traditional itemset and rule mining tasks items belonging to each transaction of the source dataset are treated equally. To differentiate items based on their relevance within each transaction, in [15] the authors first addressed the issue of mining more informative association rules, i.e., the Weighted AssociationRules (WAR). WARs are association rules enriched with weights denoting item significance. Weights were introduced only during the rule generation step after performing the traditional frequent itemset mining process. To improve the efficiency of the mining process, the authors in [16] pushed item weights deep into the itemset mining process by exploiting the anti-monotonicity of the weighted support measure in an Apriori-based itemset mining process [9]. In [18] a FP-Growth-like weighted itemset mining algorithm process is presented. Unlike [15], [16], the algorithm proposed in [18] extracts infrequent (rare) itemsets rather than frequent ones. A parallel issue is the extraction of weighted itemsets and rules when coping with data not equipped with preassigned weights. For example, to generate appropriate item weights, in [17] the dataset is modeled as a bipartite hub-authority graph and evaluated by means of a well-known indexing strategy. The PaWI system specifically addresses the problem of mining frequent weighted itemsets from data enriched with preassigned weights. Unlike [15], [16], [17], [18], it focuses on designing a parallel and distributed approach which is able to cope with large-scale weighted datasets.

III. METHODOLOGY

Parallel Weighted Itemset Miner (PaWI) is a new data mining environment aimed to analyze Big Data equipped with item weights. The main environment blocks are briefly introduced below. A more detailed description is given in the following sections.

- **Data preparation.** To prepare the source data to the itemset mining process, data are acquired, enriched with item weights, and transformed using established preprocessing techniques (e.g., data filtering). The result is stored into an HDFS data repository.

- **Weighted itemset mining.** This step entails the extraction of all frequent weighted and unweighted itemsets from the prepared datasets. To scale towards Big data, the extraction process relies on a parallel and distributed-based itemset miner running on an Hadoop cluster [20].

- **Itemset ranking.** To ease the manual exploration of most interesting patterns, the results of the weighted and unweighted itemset mining process are compared with each other and most interesting patterns are selected based on a new quality measure which combines traditional with weighted support counts.

a) Data preparation

This step entails preparing data to the subsequent itemset mining process. The source data is acquired, stored in a transactional dataset, and equipped with item weights. A transactional dataset is a set of transactions [19]. Each transaction is a set of (not repeated) items. Depending on the context of analysis, items may represent different concepts (e.g., products, objects, places, stocks). For example, let us consider the dataset reported in Table I.

<table>
<thead>
<tr>
<th>Customer id</th>
<th>Item Purchased</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X,Y,Z</td>
</tr>
<tr>
<td>2</td>
<td>X,Y,Q</td>
</tr>
<tr>
<td>3</td>
<td>X,Y,Z</td>
</tr>
</tbody>
</table>
It is an example of (unweighted) transactional dataset consisting of five transactions, each one representing a different customer of a e-commerce company. For each customer the list of purchased items is known. For instance, customer with id 1 bought items X, Y, and Z. Note that each transaction, which represent a distinct electronic basket, may contain an arbitrary number of items. To consider the relative importance of the items within each transaction during the itemset mining process, items are enriched with weights. A transactional dataset whose items are equipped with weights is denoted as weighted transactional dataset [15]. Formally speaking, a weighted transactional dataset is a set of weighted transactions.

### Table II: Example of Weighted Dataset: Item Ratings Given By Customers

<table>
<thead>
<tr>
<th>Customer ID</th>
<th>Items purchased and Ratings</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt; X, 3 &gt; &lt; Y, 1 &gt; &lt; Z, 5 &gt;</td>
</tr>
<tr>
<td>2</td>
<td>&lt; X, 2 &gt; &lt; Y, 2 &gt; &lt; Q, 2 &gt;</td>
</tr>
<tr>
<td>3</td>
<td>&lt; X, 4 &gt; &lt; Y, 2 &gt; &lt; Z, 5 &gt;</td>
</tr>
<tr>
<td>4</td>
<td>&lt; X, 3 &gt; &lt; Y, 3 &gt; &lt; Q, 2 &gt;</td>
</tr>
<tr>
<td>5</td>
<td>&lt; X, 2 &gt; &lt; W, 5 &gt; &lt; Z, 4 &gt;</td>
</tr>
</tbody>
</table>

Each weighted transaction is a set of weighted items. A weighted item is a pair &lt;item, weight&gt;, where weight is the weight associated with the corresponding item. For example, let us consider the weighted transactional dataset reported in Table II. It extends the traditional transactional dataset in Table I by enriching items with the corresponding weights. More specifically, for each customer the rating (from one to five) given to each purchased item is known. For instance, customer with id 1 rated item X as 3, item Y as 1, and item Z as 5. The analyzed data are tailored to a weighted transactional data format.

Furthermore, if need be, ad hoc preprocessing steps are applied to the raw data to ensure high-quality results. For example, data filtering and discretization are examples of commonly used preprocessing steps [19]. Data filtering entails discarding the items/transactions that are irrelevant for subsequent analyses. For instance, recalling the previous example, duplicate entries of the same customer basket can be removed because they could bias itemset support counts. To ensure the scalability of the knowledge discovery process, the PaWI system performs data filtering as a distributed MapReduce job.

#### b) Weighted itemset mining

This step focuses on mining frequent weighted itemsets [15] from the prepared weighted dataset. A k-itemset (i.e., an itemset of length k) is a set of k items. The traditional support value of an itemset in a transactional dataset is given by its frequency of occurrence in the source dataset [6].

For example, {X, Y } is an itemset indicating the co-occurrence of items X and Y. If we disregard item weights, this itemset has a support equal to 4 in Table I because it occurs in four out of five transactions, meaning that most of the users purchased items X and Y together.

The goal of this paper is to extend traditional large-scale itemset miners to successfully cope with Big weighted data. Hence, for our purposes, the itemset support measure is extended, similar to [15], to the case of weighted data. As previously done in [18], the weighted support of an itemset I in a weighted transactional dataset T is defined as a linear combination of the aggregation weights computed on each transaction in T. Its expression follows.

**Definition 1:** Weighted support. Let D be a weighted transactional dataset, I be a weighted itemset &lt;ij, wij &gt; be an arbitrary weighted item such that ij ∈ I. Let T(I) be the subset of D’s transactions containing all the items in I and f an arbitrary aggregation function defined on item weights.

The weighted support of I in D is defined as

\[
wsup(I, D) = \sum_{t \in T(I)} f_{\{j\}, k_{\{j\}}, l_{\{j\}}, ..., z_{\{j\}}}(w_{j}, w_{k}, ..., w_{z})
\]

The weighted support is the summation of all the itemset aggregation weights derived by the aggregation function f for every transaction in T. An arbitrary aggregation function f (e.g., min, max, average, mode) can be potentially applied to aggregate item weights within each transaction. The choice of f depends on the considered use cases. Hereafter, similar to [18], we will consider f=min (i.e., the least weight of any item in I is considered), because, as discussed in Section IV, the selected patterns are deemed as particularly useful for analyzing real Big datasets. Recalling the running example, let us consider an analyst who would like to discover the combinations of items that were frequently bought together with an overall rating above average. To this aim, he may consider item ratings during support computation by weighting itemset occurrences within each transaction by the least item rating.

For instance, recalling the weighted transactional dataset in table for customer with id 1 between X and Y the item with least rating is Y (rating equal to 1), while for customer with id 5 is X (rating equal to 2). Hereafter we will denote as weighted support the support of an itemset by considering item weights, whereas as traditional support the itemset support disregarding item weights. For instance, {X, Y } has weighted support equal to 1+2+2+3+0=8 and traditional support equal to 4. Given a weighted transactional dataset D and an (analyst)provided minimum support threshold minsup, the PaWI system addresses the extraction of all frequent weighted itemsets from D.
To allow comparing weighted itemsets with traditional ones, PaWI allows experts to mine traditional itemsets as well. As discussed below, the support thresholds enforced during weighted and unweighted itemset mining are potentially different. The weighted itemset mining process relies on a parallel and distributed-based algorithm running on an Hadoop cluster [20]. To make the mining process scalable towards Big Data, the mining steps are mapped to the MapReduce programming paradigm. MapReduce [21] is a parallel programming framework providing both a relatively simple programming interface together and a robust computational architecture. MapReduce programs consist of two main steps. In the map step, each mapper processes a distinct chunk of the data and produces key-value pairs. In the reduce step, key-value pairs from different mappers are combined by the framework and fed to reducers as pairs of key and value lists. Reducers generate the final output by processing the key/value lists.

To efficiently perform frequent weighted itemset mining with MapReduce PaWI integrates a variant of the BigFIM algorithm [12] which is able to successfully cope with data enriched with weights. The exploitation of weights is challenging because ad-hoc data structures must be used to efficiently maintain the weights associated with each item and transaction by balancing the impacts on computational and communication costs. The following extensions have been proposed:

- **Distributed transaction splitting.** BigFIM relies on two established itemset miners: Apriori [9] and Eclat [11]. We modified the BigFIM algorithm to allow both Apriori and Eclat to successfully cope with weighted data. More specifically, our algorithm generates an equivalent version of the source dataset that includes only transactions with equally weighted items. Let us assume that the weight of an equivalent transaction \( t_q \) is \( w \). Then, the occurrence of any itemset in \( t_q \) will be weighted by \( w \) instead of by \( 1 \). Each transaction in the original dataset may correspond to a set of equivalent transactions in the equivalent dataset. A formal definition of the equivalence set of weighted transactions is given in [18]. Note that since two distinct transactions have disjoint equivalent sets the splitting process is straightforwardly parallelizable.

- **Weighted support counting.** Since items are equipped with weights, traditional support counting is replaced with weighted support counting, according to Definition 1. To accomplish itemset support counting different strategies are adopted according to the algorithm used. Specifically, to perform Apriori-based weighted itemset mining, itemset supports are counted by generalizing the word counting problem for documents [21] to weighted itemsets, i.e., each mapper receives a disjoint subset of dataset transactions (i.e., the documents) and reports the items/itemsets (i.e., the words) for which the weighted support count is performed. A reducer combines all partial weighted support counts and reports only the items/itemsets whose weighted support is above the threshold.

These frequent weighted itemsets are redistributed to all mappers to act as candidates for the next step of breadth-first search [9] and then the procedure is repeated to mine weighted itemsets of higher length. To perform Eclat-based weighted itemset mining, each mappers builds the weighted tidlist of the itemsets related to a subset of transactions. The weighted tidlist of an arbitrary item \( i,j \) consists of all pairs (transaction id, weight) such that the transaction related to transaction id contains item \( i,j \) with weight weight. For example, let us assume that a mapper receives the transactions contained in the dataset in Table II. For item \( Z \) it generates the following weighted tidlist: \( \{ (c_1, 3.5), (c_3, 3.5), (c_4, 5.4) \} \). The weighted tidli st consists of all pairs (customer id, weight) for which the transaction related to customer id contain item \( Z \) with weight weight. For instance, the transaction corresponding the electronic basket of the customer with id 1 contains item \( Z \) with weight 5. A reducer combines all partial weighted support counts and reports only the items/itemsets whose weighted support is above the threshold. Note that the equivalent transaction weights are not stored in the distributed cache, because Big datasets may potentially consist of millions of transactions.

c) **Itemset ranking**

The manual exploration of all the itemsets (weighted or not) mined from Big data is practically unfeasible. Hence, to support the knowledge discovery process experts may would like to access only a subset of most interesting patterns. This step focuses on ranking the mined itemsets according to their level of significance in the analyzed data. To filter and rank the mined itemsets, the support measure is the most commonly used quality index [6]. To cope with weighted data, for each candidate itemset the PaWI system computes both the traditional and weighted support measures.

While the traditional support value indicates the observed frequency of occurrence of the considered combination of items in the source dataset, in weighted support counting itemset occurrences are weighted by the least item weight (see Definition 1). To select itemsets whose average least item weight is maximal the PaWI system combines the weighted and traditional support measure in a new measure called AW-sup, i.e., the Average Weighted support.

The AW-sup measure is defined as the ratio between the weighted itemset support and the traditional itemset support. It indicates the average per-transaction weight of the least weighted item. Selecting top interesting itemsets based on this measure is potentially interesting in real applications. For example, let us consider again the example dataset in Table II. According to Definition 1, itemset \( \{ X, Y \} \) has weighted and traditional support values equal to 8 and 4, respectively. Since transactions represent electronic baskets, the weighted itemset support indicates the overall least item rating computed on the subset of customers who bought both items X and Y, while the traditional support measure indicates the simple frequency of occurrence of the combinations of items in the electronic basket dataset. The AW-sup value of \( \{ X, Y \} \) is 2, meaning that, on average, for each electronic basket containing items X and Y both items have been rated at least 2.
Ranking the mined itemsets by decreasing AW-sup allows experts to consider first the combinations of items that got maximal average ratings. Note that this statistics cannot be straightforwardly computed based on simple averages, because (i) it considers only the electronic baskets containing both X and Y, (ii) for each basket it selects the rating of the least weighted item between X and Y. Itemsets not satisfying the traditional support threshold are discarded because they represent combinations of items that rarely occur in the analyzed data. The setting of the minimum weighted support threshold is driven by the average rating of the selected items. More specifically, we are interested in exploring the frequent combinations of items with rating above average, i.e., the itemsets whose AW-sup is above a minimum threshold.

IV. EXPERIMENTAL EVALUATION

To assess the effectiveness and efficiency of the proposed approach, we performed a set of experiments on a real dataset collecting approximately 34 millions of Amazon reviews.

Amazon reviews dataset. The Amazon reviews dataset consists of approximately 34 millions of Amazon reviews spanning over a time period of 18 years (from June 1995 to March 2013). The number of users who made at least one review is more than 6 millions and the number of reviewed items is greater than 2 millions. In our context, each transaction consists of a set of items reviewed by a given user. The weight associated with each item is the rating (i.e., the number of stars) given by the user in her/his review. Hence, the transactional version of the Amazon dataset consists of more than 6 millions of transactions.

The average transaction length (i.e., the average number of reviews per user) is 4.1. To generate the transactional version of the dataset a MapReduce-based preprocessing step was applied to the original data. Then, a MapReduce job was applied to merge the reviews performed by the same user. More specifically, the reduce created a set of pairs (user identifier, review) and the reducer generated for each user the list of her/his reviews. The itemsets mined from the Amazon dataset represent combinations of items that are frequently rated together with high overall rating. This information could be exploited to improve, for instance, the quality of the traditional frequently bought together approach which disregards item ratings.

The experiments were performed on a cluster of 5 nodes running Cloudera’s Distribution of Apache Hadoop (CDH5.3). Each cluster node is a 2.67 GHz six-core Intel(R) Xeon(R) X5650 machine with 32 Gbyte of main memory running Ubuntu 12.04 server with the 3.5.0-23-generic kernel. All the reported execution times are real times obtained from the Cloudera Manager web control panel. The performed experiments aimed at: (i) analyzing the characteristics and usefulness of the mined weighted itemsets and (ii) evaluating the scalability of PaWI with respect to the number of nodes.

A. Result validation

Our approach allows experts to pinpoint combinations of highly rated items purchased together by Amazon customers. To demonstrate the effectiveness of our approach compared to traditional methods, we mined all the itemsets whose AW-sup is above 4 (i.e., average number of stars>4) from the Amazon dataset. In the performed experiments we set the traditional minimum support threshold to 0.005% (i.e., the combinations of items reviewed by at least 223 users) and the minimum weighted support threshold to 4 ∗ 0.005%=0.020% (i.e., the frequent combinations of items whose average rating is above 4).

![Fig 1: Itemset Ranking based on AW-sup](image)

Our assumption is that considering item rates during itemset mining significantly changes the final ranking of the extracted combinations of items. Figure 1 summarizes the mining result, which consists of 22 highly rated itemsets composed of at least two items each. For each itemset we reported (i) the AW-sup, (ii) the itemset ranking based on AW-sup (the itemset with top AWsup ranked first), (iii) the traditional (unweighted) support, and (iv) the ranking based on traditional unweighted support (the itemset with top support ranked first). The achieved results confirm that the itemset ranking significantly changes considering the AW-sup measure rather than the traditional support. The ranking based on AW-sup appears to be the most reliable, because it placed first the highly rated combinations of items.

To better highlight the difference between the two rankings, the chart in Figure 2 plots the itemset ranking based on traditional support (y-axis) versus the ranking based on AW-sup (x-axis). Figure 1 highlights that the top itemset selected by using the AW-sup measure is ranked 3rd if the traditional support is considered. Similar ranking
gaps appear for the other itemset rankings. Hence, considering item ratings instead of simple item frequency counts allows us to differentiate between preferred combinations of items and not. These types of correlations are particularly interesting because they often disregarded by traditional data analysis tools (e.g., the original Amazon recommender system considered only the correlations between pairs) or approximated from lower-order correlations among items.

B. Scalability with the number of cluster nodes

We evaluated the scalability of the proposed architecture by measuring the execution time spent while increasing the number of Hadoop cluster nodes. We performed the experiments on a small cluster consisting of 5 nodes. Figure 2 reports the execution time achieved by setting the traditional minimum support threshold to 0.005% and the minimum weighted support threshold to 0.020%.

The achieved results show that our approach scales roughly linearly in the number of nodes and the speedup approximately corresponds to the number of cluster nodes. The execution time of the proposed algorithm was approximately 8 minutes when all the five nodes were used at the same time. Hence, the execution time was quite small, despite we tested our system on a relatively small cluster. We performed also a set of experiments to evaluate the impact of the minimum support thresholds (weighted and traditional) on the number of mined itemsets and the execution time of the proposed algorithm.

The results, not reported here due to the lack of space, meet the expectations, i.e., both the number of itemsets and the execution time increase super-linearly while decreasing the minimum support threshold.

V. CONCLUSIONS

This paper presents a parallel and distributed solution to the problem of extracting frequent itemsets from Big Weighted Datasets. The proposed system, running on an Hadoop cluster, overcomes the limitations of state-of-the-art approaches in coping with datasets enriched with item weights. The experiments, performed on a real Amazon dataset, confirm the actionability of the mining result in real context. Future works will entail the application of the proposed approach to recommender systems. For example, discovering combinations of items that were frequently bought together with an overall rating above average could be useful for recommending additional items beyond those already purchased by a given user.

REFERENCES