1) A parallel ~~Map~~-Reduce algorithm to efficiently support itemset mining on high dimensional data

2) Supporting large scale frequent itemset mining for high-dimensional data

PaMPa-HD: EXTENSION - TO MODIFY

Daniele Apiletti, Elena Baralis, Tania Cerquitelli, Paolo Garza, Fabio Pulvirenti[a], and Pietro Michiardi[b]

[a]*Dipartimento di Automatica e Informatica*
Politecnico di Torino
Torino, Italy
Email: name.surname@polito.it
[b]*Data Science Department*
Eurecom
Sophia Antipolis, France
Email: pietro.michiardi@eurecom.fr

Abstract

Frequent closed itemset mining, a data mining technique for discovering hidden correlations in transactional datasets, are among the most complex exploratory techniques in data mining. Thanks to the spread of distributed and parallel frameworks, the development of scalable approaches able to deal with the so called Big Data has been extended to frequent itemset mining. Unfortunately, most of the current algorithms are designed to cope with low-dimensional datasets, delivering poor performances in those use cases characterized by high-dimensional data. This work introduces PaMPa-HD, a parallel MapReduce-based frequent closed itemset mining algorithm for high dimensional datasets. The experimental results, performed on two real-life high-dimensional use cases, show the efficiency of the proposed approach.

in terms of load balancing, execution time and robustness to memory-issues.

Keywords: high-dimensional data, Hadoop framework
~~frequent~~ closed itemset mining.

Preprint submitted to Information Systems *June 30, 2016*

cambiare la rivista - i tempi di risposta di IS sono lunghissimi

1. Introduction

In the last years, the increasing capabilities of recent applications to produce and store huge amounts of information, the so called "Big Data", have changed dramatically the importance of the intelligent analysis of data. In both academic and industrial domains, the interest towards data mining, which focuses on extracting effective and usable knowledge from large collections of data, has risen. The need for efficient and highly scalable data mining tools increases with the size of the datasets, as well as their value for businesses and researchers aiming at extracting meaningful insights increases.

Frequent (closed) itemset mining is among the most complex exploratory techniques in data mining. It is used to discover frequently co-occurring items according to a user-provided frequency threshold, called minimum support. Existing mining algorithms revealed to be very efficient on simple datasets but very resource intensive in Big Data contexts. In general, the application of data mining techniques to Big Data collections is characterized by the need of huge amount of resources. For this reason, we are witnessing the explosion of parallel and distributed approaches, typically based on distributed frameworks, such as Apache Hadoop [1] and Spark [2]. Unfortunately, most of the scalable distributed techniques for frequent itemset mining have been designed to cope with datasets characterized by few items per transaction (low dimensionality, short transactions), focusing, on the contrary, on very large datasets in terms of number of transactions. Currently, only single-machine implementations exist to address very long transactions, such as Carpenter [3], and no distributed implementations at all.

(*) PAMPA-HD has been throughly evaluated on ~~both~~ real high dimensional datasets: ~~show in terms of execution time~~,
Experimental results show the efficiency and effectiveness of PAMPA-HD in ~~dealing~~ performing the frequent closed itemset mining with good load balancing ~~and robustness to memory issues~~.

applications/ ~~have continuously~~ generate a large number of events characterized by a variety of features. Thus, high dimensional datasets have continuously been generated.

Nevertheless, many ~~researchers in~~ scientific ~~domains~~ such as bioinformatics or networking, ~~often require to deal with this type of data~~. For instance, most gene expression datasets are characterized by a huge number of items (related to tens of thousands of genes) and a few records (one transaction per patient or tissue). Many applications in computer vision deal with high-dimensional data, such as face recognition. Some smart-cities studies have built this type of large datasets measuring the occupancy of different car lanes: each transaction describes the occupancy rate in a captor location and in a given timestamp [4]. In the networking domain, instead, the heterogeneous environment provides many different datasets characterized by high-dimensional data, such as URL reputation, advertising, and social network datasets [1, 5].

This work introduces PaMPa-HD, a parallel MapReduce-based frequent closed itemset mining algorithm for high-dimensional datasets. ~~based~~ PAMPA-HD relies on the Carpenter algorithm. PaMPa-HD outperforms the single-machine Carpenter implementation and the ~~best~~ state-of-the-art distributed approaches, in ~~both~~ execution time ~~and minimum~~ support threshold. Furthermore, ~~the implementation~~ the PAMPA-HD design takes into account crucial design aspects, such as load balancing and robustness to memory-issues. (*)

lower
by supporting

The paper is organized as follows: Section 2 introduces the frequent (closed) itemset mining problem, Section 3 briefly describes the centralized version of Carpenter, and Section 4 presents the proposed PaMPa-HD algorithm. Section 5 describes the experimental evaluations proving the effectiveness of the proposed technique. Section 6 provides a brief review of the state of the art, and Section 7 discusses possible applications of PaMPa-HD. Finally, Section 8 introduces future works and conclusions.

To effectively deal with those high-dimensional datasets, novel and distributed approaches are needed.

$\mathcal{D}$	
tid	items
1	a,b,c,l,o,s,v
2	a,d,e,h,l,p,r,v
3	a,c,e,h,o,q,t,v
4	a,f,v
5	a,b,d,f,g,l,q,s,t

(a) Horizontal representation of $\mathcal{D}$

TT	
item	tidlist
a	1,2,3,4,5
b	1,5
c	1,3
d	2,5
e	2,3
f	4,5
g	5
h	2,3
l	1,2,5
o	1,3
p	2
q	3,5
r	2
s	1,5
t	3,5
v	1,2,3,4

(b) Transposed representation of $\mathcal{D}$

$TT\|_{\{2,3\}}$	
item	tidlist
a	4,5
e	-
h	-
v	4

(c) $TT\|_{\{2,3\}}$: example of conditional transposed table

Figure 1: Running example dataset $\mathcal{D}$

2. Frequent itemset mining background

Let $\mathcal{I}$ be a set of items. A transactional dataset $\mathcal{D}$ consists of a set of transactions $\{t_1, \ldots, t_n\}$, where each transaction $t_i \in \mathcal{D}$ is a set of items (i.e.,

$t_i \subseteq \mathcal{I}$) and it is identified by a transaction identifier (tid_i). Figure 1a reports an example of a transactional dataset with 5 transactions. ~~The dataset reported in Figure 1a is~~ used as a running example through the paper. It is

An itemset I is defined as a set of items (i.e., $I \subseteq \mathcal{I}$) and it is characterized by a tidlist and a support value. The tidlist of an itemset I, denoted by $tidlist(I)$, is defined as the set of tids of the transactions in D containing I, while the support of I in $\mathcal{D}$, denoted by $sup(I)$, is defined as the ratio between the number of transactions in D containing I and the total number of transactions in $\mathcal{D}$ (i.e., $|tidlist(I)|/|\mathcal{D}|$). For instance, the support of the itemset $\{aco\}$ in the running example dataset $\mathcal{D}$ is 2/5 and its tidlist is $\{1, 3\}$. An itemset I is considered frequent if its support is greater than a user-provided minimum support threshold $minsup$.

Given a transactional dataset D and a minimum support threshold $minsup$, the Frequent Itemset Mining [6] problem consists in extracting the complete set of frequent itemsets from D. In this paper, we focus on a valuable subset of frequent itemsets called frequent closed itemsets [3]. Closed itemsets allow representing the same information of traditional frequent itemsets in a more compact form.

A transactional dataset can also be represented in a vertical format, which is usually a more effective representation of the dataset when the average number of items per transactions is orders of magnitudes larger than the number of transactions. In this representation, also called transposed table Italico TT, each row consists of an item i and its list of transactions, i.e., $tidlist(\{i\})$. Let r be an arbitrary row of TT. $r.tidlist$ denotes the tidlist of row r. Figure 6a reports the transposed representation of the running example reported

citerei la figura 1B

in Figure 1a.

Given a transposed table TT and a tidlist X, the conditional transposed table of TT on the tidlist X, denoted by $TT|_X$, is defined as a transposed table such that: (1) for each row $r_i \in TT$ such that $X \subseteq r_i.tidlist$ there exists one tuple $r'_i \in TT|_X$ and (2) r'_i contains all tids in $r_i.tidlist$ whose tid is higher than any tid in X.

For instance, consider the transposed table TT reported in Figure 6a. citare 1B The projection of TT on the tidlist $\{2, 3\}$ is the transposed table reported in Figure 1c.

Each transposed table $TT|_X$ is associated with an itemset composed by the items in $TT|_X$. For instance, the itemset associated with $TT|_{\{2,3\}}$ is $\{acfn\}$ (see Figure 1c).

3. The Carpenter algorithm

Non ~~...~~ ~~...~~ inizierei così. Io lo toglierei

~~Emettere una frase qui, sent~~

As discussed in section 6, the most popular techniques (e.g., Apriori [7] and FP-growth [8]) adopt the itemset enumeration approach ~~to mine the frequent itemsets.~~ to perform itemset mining However, itemset enumeration revealed to be ineffective with datasets with a high average number of items per transactions [3]. To tackle this problem, the Carpenter algorithm [3] was proposed. Specifically, Carpenter is a frequent itemset extraction algorithm devised to handle datasets characterized by a relatively small number of transactions but a huge number of items per transaction. To efficiently solve the itemset mining problem, Carpenter adopts an effective depth-first transaction enumeration approach based on the transposed representation of the input dataset. To illustrate the centralized version of Carpenter, we will use the running example dataset

(see Section 6 for a further discussion)

1B?

Recall that in

D reported in Figure 1a, and more specifically its transposed version (see Figure 6a). ~~As already described in Section 2, in~~ the transposed representation each row of the table consists of an item *i* with its tidlist. For instance, the last row of Figure 6a points that item *e* appears in transactions 1, 2, 3, 4.

1B

Carpenter builds a transaction enumeration tree where each node corresponds to a conditional transposed table $TT|_X$ and its related information (i.e., the tidlist X with respect to which the conditional transposed table is built and its associated itemset). The transaction enumeration tree, when pruning techniques are not applied, contains all the tid combinations (i.e., all the possible tidlists X). Figure 2 reports the transaction enumeration tree obtained by processing the running example dataset. To avoid the generation of duplicate tidlists, the transaction enumeration tree is built by exploring the tids in lexicographical order (e.g., $TT|_{\{1,2\}}$ is generated instead of $TT|_{\{2,1\}}$). Each node of the tree is associated with a conditional transposed table on a tidlist. For instance, the conditional transposed table $TT|_{\{2,3\}}$ in Figure 1c matches the node {2,3} in Figure 2.

Carpenter performs a depth first search of the enumeration tree to mine the set of frequent closed itemsets. Referring to the tree in Figure 2, the depth first search would lead to the visit of the nodes in the following order: {1}, {1,2}, {1,2,3}, {1,2,3,4}, {1,2,3,4,5}, {1,2,3,5}, {...}. For each node, Carpenter applies a procedure that decides if the itemset associated with that node is a frequent closed itemset or not. Specifically, for each node, Carpenter decides if the itemset associated with the current node is a frequent closed itemset by considering: 1) the tidlist X associated with the node, 2)

Scrivere una frase che sintetizzi il funzionamento di Carpenter_Poi i dettagli.

Per esempio.

Carpenter performs 2 steps

- 1) Building a transaction enumeration tree by exploiting different pruning rules to avoid the generation of a portion of the tree when not necessary

2) ~~A depth first~~ ~~sear~~ visiting the tree through a depth first search. ~~Different pruning rules are exploited to avoid the visit of tree when not~~ ~~(stop)~~ ~~necessary~~.

Ingrandire un po' il font (io non leggo :-(( )

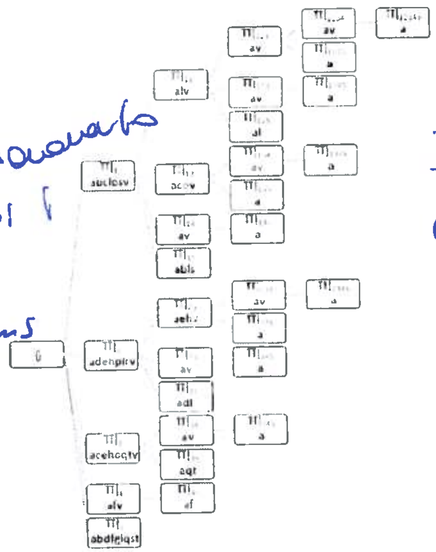

Figure 2: The transaction enumeration tree of the running example dataset in Figure 1a. For the sake of clarity, no pruning rules are applied to the tree.

the conditional transposed table $TT|_X$, 3) the set of frequent closed itemsets found up to the current step of the tree search, and 4) the enforced minimum support threshold (*minsup*). Based on the theorems reported in [3], if the itemset *I* associated with the current node is a frequent closed itemset then

FAI UN CHECK X VERIFICARE CHE

I is included in the frequent closed itemset set. Moreover, by exploiting the analysis performed on the current node, part of the remaining search space (i.e., part of the enumeration tree) can be pruned, to avoid the analysis of nodes that will never generate new closed itemsets. At this purpose, three pruning rules are applied on the enumeration tree, based on the evaluation performed on the current node and the associated transposed table $TT|_X$:

- **Pruning rule 1.** If the size of X, plus the number of distinct tids in the rows of $TT|_X$ does not reach the minimum support threshold, the subtree rooted in the current node is pruned.
- **Pruning rule 2.** If there is any tid tid_i that is present in all the tidlists of the rows of $TT|_X$, tid_i is deleted from $TT|_X$. The number of discarded tids is updated to compute the correct support of the itemset associated with the pruned version of $TT|_X$.
- **Pruning rule 3.** If the itemset associated with the current node has been already encountered during the depth first search, the subtree rooted in the current node is pruned because it can never generate new closed itemsets.

The tree search continues in a depth first fashion moving on the next node of the enumeration tree. More specifically, let tid_i be the lowest tid in the tidlists of the current $TT|_X$, the next node to explore is the one associated with $X' = X \cup \{tid_i\}$.

Among the three rules mentioned above, pruning rule 3 assumes a global knowledge of the enumeration tree explored in a depth first manner. This,

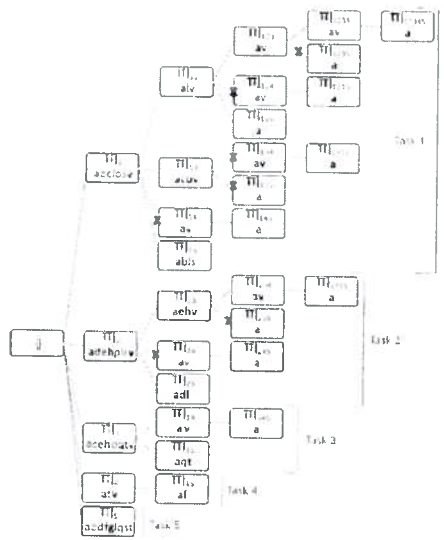

Figure 3: Running toy example: each node expands a branch of the tree independently. Pruning rule 1 and 2 are not applied. The pruning rule 3 is applied only within the same task: the red crosses on the edges represent pruned nodes due to local pruning rule 3, e.g. the one on node {2 4} represents the pruning of node {2 4}.

as detailed in section 4, is very challenging in a distributed environment that adopts a shared-nothing architectures, like the ones we address in this work.

SONO ARRIVATA QUI

4. The PaMPa-HD algorithm

Given the complete enumeration tree (see Figure 2), the centralized Carpenter algorithm extracts the whole set of closed itemsets by performing a depth first search (DFS) of the tree. Carpenter also prunes part of the search space by applying the three pruning rules illustrated above. The PaMPa-HD algorithm proposed in this paper splits the depth first search process in a set of (partially) independent sub-processes, that autonomously evaluate sub-trees of the search space. Specifically, the whole problem can be split by assigning each subtree rooted in $TT|_X$, where X is a single transaction id in the initial dataset, to an independent sub-process. Each sub-process applies the centralized version of Carpenter on its conditional transposed table $TT|_X$ and extracts a subset of the final closed itemsets. The subsets of closed itemsets mined by each sub-process are merged to compute the whole closed itemset result. Since the sub-processes are independent, they can be executed in parallel by means of a distributed computing platform, e.g., Hadoop. Figure 3 shows the application of the proposed approach on the running example. Specifically, five independent sub-processes are executed in the case of the running example, one for each row (transaction) of the original dataset.

Partitioning the enumeration tree in sub-trees allows processing bigger enumeration trees with respect to the centralized version. However, this approach does not allow fully exploiting pruning rule 3 because each sub-process works independently and is not aware of the partial results (i.e., closed itemsets) already extracted by the other sub-processes. Hence, each sub-process can only prune part of its own search space by exploiting its

"local" closed itemset list, while it cannot exploit the closed itemsets already mined by the other sub-processes. For instance, Task T2 in Figure 3 extracts the closed itemset av associated with node $TT|_{2,3,4}$. However, the same closed itemset is also mined by T1 while evaluating node $TT|_{1,2,4}$. In the centralized version of Carpenter, the duplicate version of av associated with node $TT|_{1,2,4}$ is not generated because $TT|_{1,2,4}$ follows $TT|_{1,2,3}$ in the depth first search, i.e., the tasks are serialized and not parallel. Since pruning rule 3 has a high impact of the reduction of the search space, as detailed in Section 5, its inapplicability leads to a negative impact on the execution time of the distributed algorithm as described so far. To address this issue, we share partial results among the sub-processes. Each independent sub-process analyzes only a part of the search subspace, then, when a maximum number of visited node is reached, the partial results are synchronized through a synchronization phase. Of course, the exploration of the tree finishes also when the subspace has been completely explored. Specifically, the sync phase filters the partial results (i.e. nodes of the tree still to be analyzed and found closed itemsets) globally applying pruning rule 3. The pruning strategy consists of two phases. In the first one, all the transposed tables and the already found closed itemsets are analyzed. The transposed tables and the closed itemsets related to the same itemset are grouped together in a bucket. For instance, in our running example, each element of the bucket B_{av} can be:

- a frequent closed itemset av extracted during the subtree exploration of the node $TT_{3,4}$.
- a transposed table associated to the itemset av among the ones that still have to be expanded (nodes $TT_{1,2,3}$ and $TT_{2,3,4}$).