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

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

non ho anche l'affiliation di Eurecom?

Abstract

is?

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 execution time, load balancing and robustness to memory issues.

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

U

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" [1], have changed dramatically the importance of the intelligent analysis of data. Data mining, together with machine learning [2], is considered one of the fondamental tools on which Big Data analytics are based. 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 [3] and Spark [4]. 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 [5], and no distributed implementations at all.
Nevertheless, many scientific applications, such as bioinformatics or networking, generate a large number of events characterized by a variety of features. Thus, high-dimensional datasets have been continuosly generated. 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. An increasing portion of big data is actually related to geospatial data [6] and smart-cities. Some 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 [7]. 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 [7, 8]. To effectively deal with those high-dimensional datasets, novel and distributed approaches are needed.

This work introduces PaMPa-HD [9], a parallel MapReduce-based frequent closed itemset mining algorithm for high-dimensional datasets. PaMPa-HD relies on the Carpenter algorithm. PaMPa-HD outperforms the single-machine Carpenter implementation and the state-of-the-art distributed approaches, in execution time and by supporting lower minimum support threshold. Furthermore, the PaMPa-HD design takes into account crucial design aspects, such as load balancing and robustness to memory-issues. PaMPa-HD has been thoroughly evaluated on real high dimensional datasets. Experimental results show the efficiency and the effectiveness of PaMPa-HD in

Aggiungere citazione

The source code of PAMPA can be downloaded from XXX.

performing the frequent closed itemset mining with good load balancing.

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.

presents

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. It is used as a running example through the paper.

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 $\mathcal{D}$ 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 $\mathcal{D}$ 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 $\mathcal{D}$ and a minimum support threshold $minsup$,

Lo scriverei anche nelle lettere per l'editor.

Algorithm 1 PaMPa-HD at a glance

```
1: procedure PaMPa-HD(minsup; initial TT)
2:     Job 1 Mapper: process each row of TT
         and send it to reducers, using as key values
         the tids of the tidlists
3:     Job 1 Reducer: aggregate TT|, and run
         local Carpenter until expansion threshold is
         reached or memory is not enough
4:     Job 2 Mapper: process all the closed itemset
         or transposed tables from the previous job
         and send them to reducers
5:     Job 2 Reducer: for each itemset belonging
         to a table or a frequent closed, keep
         the eldest in a Depth First fashion
6:     Job 3 Mapper: process each closed itemset
         and TT|, from the previous job.
         For the transposed tables run local Carpenter
         until expansion threshold is reached
7:     Job 3 Reducer: for each itemset belonging
         to a table or a frequent closed, keep
         the eldest in a Depth First fashion
8:     Repeat Job 3 until there are no more
         conditional tables
9: end procedure
```

The Job 1, whose pseudocode is reported in Algorithm 2, is developed to distribute the input dataset to the independent tasks, which will run a local and partial version of the Carpenter algorithm. The second job performs the synchronization of the partial results and exploits the pruning rules. At the end, the last job interleaves the Carpenter execution with the synchronization phase.

Job 1 (Algorithm 2) non a capo

Each mapper is fed with a transaction of the input dataset, which is supposed to be in a vertical representation, together with the minsup parameter. As detailed in Algorithm 2, each transaction is in the form *item, tidlist*. For each transaction, the mapper performs the following steps. For each tid t_i of the input tidlist, given $TL_{greater}$ the set of tids $(t_{i+1}, t_{i+2}, \ldots t_n)$ greater than the considered tid t_i (lines 2-7 in Algorithm 2).

- If $|TL_{greater}| >= minsup$, output a key-value pair <key= t_i; value= $TL_{greater}$, item>, then analyze t_{i+1} of the tidlist.
- Else discard the tidlist.

For instance, if the input transaction is the tidlist of item b (b, 1 2 3) and minsup is 1, the mapper will output three pairs: <key=1; value=2 3, b>, <key=2; value=3, b>, <key=3; value=b>.

After the map phase, the MapReduce shuffle and sort phase aggregates the <key,value>pairs and delivers to reducers the nodes of the first level of the tree, which represent the transposed tables projected on a single tid (lines 10-13 in Algorithm 2). The tables in Figure 6 illustrate the processing of a row of the initial Transposed representation of D. Reducers run a local Carpenter implementation from the input tables. Given that each key matches a single transposed table TT_X, each reducer builds the transposed tables with the tidlists contained in the "value" fields.

From this table, a local Carpenter job is run. Carpenter recursively processes a transposed table expanding it in a depth-first manner (see Section 3 for further details). At each iteration of the Carpenter subroutine, a counter is increased. When the count is over the given maximum expansion threshold, the main routine is not invoked anymore. In this case, all the intermediate results are written to HDFS.

1. the transposed table is composed using the tidlists from each key-value and a local Carpenter job is run
2. each recursion of the Carpenter subroutine increases a counter which is compared to the expansion threshold before each recursion
3. if the count is below the threshold another Carpenter recursion is scheduled
4. else, Carpenter main routine is not invoked anymore but all the intermediate results are written to HDFS

During the local Carpenter process, the found closed itemsets and the explored branches are stored in memory in order to apply a local pruning. The closed itemsets are emitted as output at the end of the task, together with the tidlist of the node of the tree in which they have been found. This information is required by the synchronization phase in order to establish which element is the eldest in a depth first exploration.

Job 2 (Algorithm 3) The synchronization phase is a straightforward MapReduce job in which mappers input is the output of the previous job: it is composed of the closed frequent itemsets found in the previous Carpenter tasks and intermediate transposed tables that still have to be expanded. The itemsets are associated to their minsup and the tidlist related to the node of the tree in which they have been found; the transposed tables are associated to the table content, the corresponding itemset and the table tidlist. For each itemset, the mappers output a pair of the form <key=itemset;

item	tidlist
a	1,2,3,4,5

(a) Transposed representation of $\mathcal{D}$: tidlist of item a

<table>
<tr><th>key</th><th>value</th></tr>
<tr><td>1</td><td>2,3,4,5 |a</td></tr>
<tr><td>2</td><td>3,4,5 |a</td></tr>
<tr><td>3</td><td>4,5 |a</td></tr>
<tr><td>4</td><td>5 |a</td></tr>
<tr><td>5</td><td>- |a</td></tr>
</table>

(b) Emitted key-value entries from the example row in Table 1b

<table>
<tr><th>key</th><th>value</th></tr>
<tr><td>3</td><td>4,5 |a</td></tr>
<tr><td>3</td><td>- |c</td></tr>
<tr><td>3</td><td>- |e</td></tr>
<tr><td>3</td><td>- |h</td></tr>
<tr><td>3</td><td>- |o</td></tr>
<tr><td>3</td><td>5 |q</td></tr>
<tr><td>3</td><td>5 |t</td></tr>
<tr><td>3</td><td>4 |v</td></tr>
</table>

(c) key-value entries for key 3

$TT\|_{(3)}$	
item	tidlist
a	4,5
c	-
e	-
h	-
o	-
q	5
t	5
t	5
v	4

(d) $TT|_{(3)}$: composed with the received values

Figure 6: Job 1 applied to the running example dataset: local Carpenter algorithm is run from the Transposed Table 6d.

Algorithm 2 Dataset distribution and local and partial Carpenter execution (Job 1)

```
procedure MAPPER(minsup, item_i, tidlist TL)
    for j = 0 to |(TL)| - 1 do
     tidlist TL_greater = set of tids greater than
     the considered tid t_j
        if |TL_greater| ≥ minsup then
            output <key= t_j; value= TL_greater, item>
        else Break
        end if
    end for
end procedure
procedure REDUCER(key = tid X, value = tidlists TL[ ])
    Create new transposed table TT|_X
    for each tidlist TL_i of TL[ ] do
        add TL_i to TT|_X (populate the transposed table)
    end for
    while max_exp is not reached do
        Run Carpenter(minsup, TT|_X)
    end while
        Output<itemset; tidlist + Transposedtable 1 rows>
    for each frequent closed itemset found do
        Output(<itemset; tidlist + support>)
    end for
end procedure
```

value=tidlist.minsup>(lines 6 - 11 of Algorithm 3): for each table, the mappers output a pair of the form <key=itemset; value=tidlist.table_content>(lines 2 - 5 of Algorithm 3). The shuffle and sort phase delivers to the reducers the pairs aggregated by keys. The reducers, which matches the buckets introduced in Section 4, compare the entries and emit, for the same key or itemset, only the oldest version in a depth first exploration (lines 15 - 21 of Algorithm 3). For instance, referring to our running example in Figure 5, in

the bucket of the itemset *ae* are collected the entries related to the nodes T_{123} and T_{234}. Since the tidlist 123 is previous than 234 in a depth-first exploration order, the reducer keeps and emits only the entry related to the node T_{123}. With this design, the redundant tables are discarded with a pruning very similar to the one related to a centralized memory at the cost of a very MapReduce-like job.

Job 3 (Algorithm 4) non a capo

This is a mixture of the two previous jobs. In the Map phase all the remaining tables are expandend by a local Carpenter routine. The Reduce phase, instead, applies the same kind of synchronization that is run in the synchronization job. The job has two types of input: transposed tables and frequent closed itemsets. The former are processed respecting a depth-first sorting and expanded until it is reached the maximum expansion threshold (lines 5 - 7 of Algorithm 4). From that moment, the tables are not expanded but sent to the reducers (line 8 of Algorithm 4). Please note that the tree exploration processing the initial transposed tables in a depth-first order is more similar to a centralized architecture, enhancing the impact of the pruning rule 3. The latter (i.e. the frequent closed itemsets of the previous PaMPa-HD job) are processed in the following way. If in memory there is already an oldest depth-first entry of the same itemset, the closed itemset is discarded. If there is not, it is saved into memory and used to improve the local pruning effectiveness (lines 2 - 3). At the end of the task, all the frequent closed found are sent to the reducers. This job is iterated until all the Transposed Tables have been processed.

Algorithm 3 Synchronization Phase and exploitation of the pruning rule 3 (Job 2)

```
procedure MAPPER(Frequent Closed itemset;
    Transposed table)
  if Input I is a table then
    itemset ← ExtractItemset(I)
    tidlist ← ExtractTidlist(I)
    Output(<itemset; tidlist − table I rows>)
  else (i.e. input I is a frequent closed itemset)
    itemset ← ExtractItemset(I)
    tidlist ← ExtractTidlist(I)
    support ← ExtractSupport(I)
    Output(<itemset; tidlist + support>)
  end if
end procedure
procedure REDUCER(key = itemset;
    value = itemsets & tables T[ ])
  oldest ← null
  for each itemset or table T of T[ ] do
    tidlist ← ExtractTidlist(T)
    if tidlist previous of oldest in a Depth-First Search then
      oldest ← T
    end if
  end for
  Output(<itemset + oldest>)
end procedure
```

Thanks to the introduction of a global synchronization phase (Job 2 and Job 3 in Algorithms 3 and 4), the proposed PaMPa-HD approach is able to apply pruning rule 3 and handle high-dimensional datasets, otherwise not manageable due to memory issues.

Algorithm 4 Interleaving of the Carpenter execution and synchronization phase (Job 3)

```
procedure MAPPER(Frequent Closed itemset;Transposed table)
  if Input I is a frequent closed itemset then
    save I to local memory
  else (i.e. input I is a Transposed Table)
    while max_exp is not reached do
      Run Carpenter(minsup, TT|_X)
    end while
    Output(<itemset; tidlist − table I rows>)
  end if
  for each frequent closed itemset found do
    Output(<itemset; tidlist + support>)
  end for
end procedure
procedure REDUCER(key = itemset;
    value = itemsets & tables T[ ])
  oldest ← null
  for each itemset or table T of T[ ] do
    tidlist ← ExtractTidlist(T)
    if tidlist previous of oldest in a Depth-First Search then
      oldest ← T
    end if
  end for
  Output(<itemset + oldest>)
end procedure
```

we discuss
and experimentally evaluate some self-tuning strategies to automatically set the max_exp parameter (section 5.1)

(Section 5.2)

5. Experiments

In this section, we present a set of experiments to evaluate the performance of the proposed algorithm. Firstly, NEXT we assess the impact on performance of the maximum expansion threshold value. ~~Secondly~~ third, we evaluate the speed of the proposed algorithm, comparing it with the state-of-the-art distributed approaches (Section 5.3). Finally, ~~we focus on more technical aspects of our approach. Specifically,~~ we experimentally analyze the impact of (i) the number of transactions of the input dataset (Section 5.4), (ii) the number of parallel tasks (Section 5.5), and (iii) the communication costs and load balancing behavior (Section 5.6).

Aggiungere i trattini

Experiments have been performed on two real-world datasets. The first is the **PEMS-SF** dataset [13], which describes the occupancy rate of different car lanes of San Francisco bay area freeways (15 months worth of daily data from the California Department of Transportation [14]). Each transaction represents the daily traffic rates of 963 lanes, sampled every 10 minutes. It is characterized of 440 rows and 138,672 attributes (6 x 24 x 963), and it has been discretized in equi-width bins, each representing 0.1% occupancy rate.

togliere il bold

Since PaMPa-HD is designed to cope with high-dimensional datasets characterized by a small number of transactions, we have used several down-sampled versions (in terms of number of rows) of the datasets to measure the impact of the number of transactions on the performance of the algorithm.

The second dataset is the **Kent Ridge Breast Cancer** [15], which contains gene expression data. It is characterized by 97 rows that represent patient samples, and 24,482 attributes related to genes. The attributes are numeric (integers and floating point). Data have been discretized with an

togliere il bold

equal-depth partitioning using 20 buckets (similarly to [5]).

The discretized versions of the real datasets are publicly available at http://dbdmg.polito.it/PaMPa-HD/.

Table 1: Datasets

Dataset	Number of transactions	Number of different items	Average number of items per transaction
PEMS-SF Dataset	440	8,685,087	138,672
Kent Ridge Breast Cancer Dataset	97	489,640	24,492

PaMPa-HD is implemented in Java 1.7.0_60 using the Hadoop MR API. Experiments were performed on a cluster of 5 nodes running Cloudera Distribution of Apache Hadoop (CDH5.3.1). 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.

5.1. Impact of the maximum expansion threshold

In this section we analyze the impact of the maximum expansion threshold (*max_exp*) parameter, which indicates the maximum number of nodes to be explored before a preemptive stop of each distributed sub-process is forced. This parameter, as already discussed in Section 4, strongly affects the enumeration tree exploration, forcing each parallel task to stop before completing the visit of its sub-tree and send the partial results to the Synchronization phase. This approach allows the algorithm in this phase to globally apply pruning rule 3 and reduce the search space. Low values of *max_exp* threshold increases the load balancing, because the global problem is split into simpler and less memory-demanding sub-problems, and, above all, facilitate the global application of pruning rule 3, hence a smaller subspace is searched. However, higher values allow a more efficient execution, by limiting the start and stop of distributed tasks (similarly to the context switch penalty) and the synchronization overheads. Above all, higher values enhance the pruning effect of the state centralized memory. In order to assess the impact of the expansion threshold parameter, we have performed two set of experiments. In the first one we perform the mining on the PEMS-SF (100 transactions) dataset with a minsup 10, by varying *max_exp* from 100 to 100,000,000. The minsup value has been empirically selected in order to let the mining problem being deep enough to show different performance. In Figure 7 are shown the results in terms of execution time and number of iterations (i.e., the number of jobs)[1]. It is clear how the *max_exp* parameter can influence the performance, with wall-clock times that can be doubled with different configurations. The best performance in terms of execution time is achieved with a maximum expansion threshold equal to 10,000 nodes. With lower values, the execution times are slightly longer, while there is an evident performance degradation with higher *max_exp* values. This result highlights the importance of the synchronization phase. Increasing the *max_exp* pa-

[1]Please note that in all the experiments, for sake of clarity, the confidence intervals (obtained after a sufficient number of executions and with complementary level of significance of 95%) are omitted from the graphs.

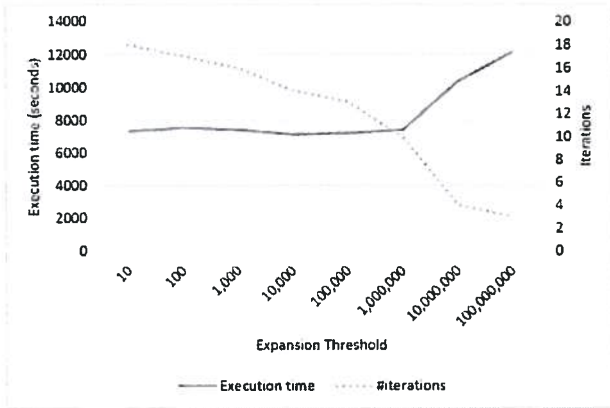

Figure 7: Execution time and number of iterations for different *max_exp* values on PEMS-SF dataset with *minsup*=10.

rameter makes the number of iterations decreasing, but more useless tree branches are explored, because pruning rule 3 is globally applied less frequently. Lower values of *max_exp*, instead, raising the number of iterations, introduce a slight performance degradation caused by iterations overheads.

The same experiment is repeated with the Breast Cancer dataset and a minsup value of 5. As shown in Figure 8, even in this case, the best performances are achieved with *max_exp* equal to 10,000. In this case, differences are more significant with lower *max_exp* values, although with a non-negligible performance degradation with higher values.

The value of *max_exp* impacts also the load balancing of the distributed computation among different nodes. With low values of *max_exp*, each task

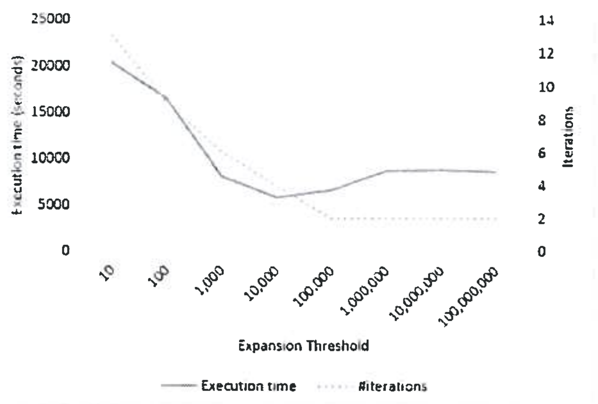

Figure 8: Execution time and number of iterations for different max_exp values on Breast Cancer dataset with $minsup$=5.

explores a smaller enumeration sub-tree, decreasing the size difference among the sub-trees analyzed by different tasks, thus improving the load balancing. Table 2 reports the minimum and the maximum execution time of the mining tasks executed in parallel for both the datasets and for two extreme values of max_exp. The load balance is better for the lowest value of max_exp.

The max_exp choice has a non-negligible impact on the performances of the algorithm. However, as demonstrated by the curves in Figures 7 and 8, it is very dependent on the use case and distribution of the data. In the next subsection we introduce and motivate some tuning strategies related to max_exp.

Table 2: Load Balancing

	Task execution time Breast Cancer		Task execution time PEMS-SF	
Maximum expansion threshold	Min	Max	Min	Max
100,000,000	7 m	2h 16m 17s	11s	2h 20m 28s
10	6m 21s	45m 16s	6s	2m 24s

5.2. *Proposed strategies* Self-tuning strategies - Lo metterei come sezione 5.1

This section introduces some heuristic strategies related to the max_exp parameter. The aim of this experiment is to identify an heuristic technique which is able to deliver good performances without the need by the user to manually tune the max_exp parameter. To introduce the techniques, we provide motivations behind their design in the following. Because of the enumeration tree architecture, the first tables of the tree are the most populated. Each node, in fact, is generated from its parent node as a projection of the parent transposed table on a tid. In addition, the first nodes are, in the average, the ones generating more sub-branches. By construction, their transposed table tidlists are, by definition, longer than the ones of their children nodes. This increases the probability that the table could be projected on a tid. For these reasons, the tables of the initial mining phase are the most heavy to be processed. On the other hand, the number of nodes to process by each local Carpenter iteration tends to increase with the number of iterations. Still, this factor is mitigated by (i) the decreasing size of the tables and (ii) the eventual end of some branches expansion (i.e. when there four

are not more tids in the node transposed table). These reasons motivated us to introduce some strategies that assume a maximum expansion threshold that is increased with the number of iterations. These strategies start with very low values in very initial iterations (i.e. when the nodes are more heavy to be processed) and increase *max_exp* during the mining phases.

im italico

The strategy #1 is the most simple: the *max_exp* is increased with a factor of X at each iteration. For instance, if the *max_exp* is set to 10, and X is set to 100 at the second iteration it is raised to 1000 and so on. In addition to this straightforward approach, we have tried to leverage information 5 about the execution time of each iteration and the pruning effect (i.e. the percentage of transposed tables / nodes that are pruned in the synchronization job). Specifically strategy #2 consists in increasing, at each iteration, the *max_exp* parameter with a factor of $X^{T_{old}/T_{new}}$, given T_{new} and T_{old} the execution time of the previous two jobs. The motivation is to balance the growth of the parameter in order to achieve a stable execution times among the iterations. For strategy #3, we take into account the relative number of pruned tables. Indeed, this value cannot be easily interpreted. An increasing pruning percentage means that there are a lot of tables that are generated uselessly. However, an increasing trend is also normal, since the number of nodes that are processed increases exponentially. Given that our intuition is to rise the *max_exp* among the iterations, in strategy #3, we increase the *max_exp* parameter with a factor $X^{Pr_{old}/Pr_{new}}$, given Pr_{new} and Pr_{old} the relative number of pruned tables in the previous two jobs. Finally strategy #4 is inspired by the congestion control of TCP/IP (a data transmission protocol used by many Internet applications [16]). Precisely, the *max_exp* is

handled like the congestion window size (i.e. the number of packets that are sent without congestion issues). This strategy, called "Slow Start", assumes two types of growing of the window size: an exponential one and a linear one. In the first phase, the window size is increased exponentially until it reaches a threshold ("ssthresh", which is calculated from some empirical parameters such as Round Trip Time value). From that moment, the growth of the window becomes linear, until a data loss occurs. In our case, we just inherit the two growth factor approach. Therefore, our "slow start" strategy consists in increasing the *max_exp* of a factor of X until the last iteration reaches an execution time greater than a given threshold. After that, the growth is more stable, increasing the parameter of a factor of 10 (for this reason $X \geq 10$). We have fixed the threshold to the execution time of the first two jobs (Job 1 and Job 2). These jobs, for the architecture of our algorithm, consists of the very first Carpenter iteration. They are quite different than the others since the first Mapper phase has to build the initial projected transposed tables (first level of the tree) from the input file. This choice is consistent with our initial aim, that is to normalize the execution times of the last iterations which are often shorter than the first ones. **Fabio & Paolo:Non siamo sicuri che convenga inserire questa parte sul time out. Michiardi: I guess it is ok: mechanisms like speculative execution work similarly, hence to me the approach is not shocking. @TANIA: allora eliminiamo e magari lo mettiamo come idea per i future works.** The increasing *max_exp* value introduced by the described strategies, however, leads to a degradation of the load balancing between the parallel tasks of the job. To limit this issue, we have introduced a timeout of

Lo mettiamo nei future work

Table 3: Strategies

Strategy #1(X)	Increasing at each iteration with a factor of X
Strategy #2(X)	Increasing at each iteration with a factor of $X^{T_{old}/T_{new}}$
Strategy #3(X)	Increasing at each iteration with a factor of $X^{Pr_{old}/Pr_{new}}$
Strategy #4	Slow start, with a fast increase factor of X

1 hour. After that, all the tasks will be forced to run the synchronization job. From the algorithmic point of view, this is not a loss, since the ~~the~~ tables are expanded in a depth-first fashion. The last tables, hence, are the ones with the highest probability to be pruned. Although, in this way, we are limiting to 1 hour the amount of time in which we are not completely exploiting the resources of the commodity cluster (i.e. only few very long tasks running). A value of 1 hour has been empirically proved to be a good trade-of between load balancing and a good leveraging of the centralized memory pruning.

Strategy #1 is the one achieving the best performances for both the datasets. ~~In~~ Table 4 ~~are returned~~ reports the best performance for each strategy, in terms of relative performance difference with the best results obtained with a fixed *max_exp* parameter. For PEMS-SF dataset, even strategies #2 and #3 are able to achieve positive gains. For Breast Cancer dataset strategy #1 is the best, followed by strategy #4: these are the only ones achieving significant positive gain over the fixed *max_exp* approach. All the strategies

Table 4: Strategies performance

Strategies	PEMS-SF	Breast Cancer
Strategy #1	-6.48% (X = 10)	-19.03% (X = 100.000)
Strategy #2	-3,73% (X = 1.000)	-0.02% (X = 10.000)
Strategy #3	-4.42% (X = 100)	+1.59% (X = 100)
Strategy #4	+9.39% (X = 100)	-16.17% (X = 1,000)

are evaluated with X from 10 to 10,000. The max value has been increased in the cases in which the performance suggest a decreasing execution time trend.

Since the best performance is achieved with values of 10 and 100,000 respectively for PEMS-SF and Breast Cancer datasets (Figures 9 and 10), we will use this configuration for the experiments comparing PaMPa-HD with other distributed approaches. The difference may be caused by the characteristics of the dataset: evidently, PEMS-SF dataset benefits of more synchronization phases.

Fabio: queste figure sono indispensabili? Michiardi: in my opinion either: i) we omit them and only report # in the text ii) we put a table. Fabio: Se approvate, le eliminerei e specifico la percentuale vincente nel testo, visto che la figura 9 pessima e con quel picco cosi' alto potrebbe suscitare domande scomode. In questo modo si

LASCIAMOLO SOLO NEL TESTO

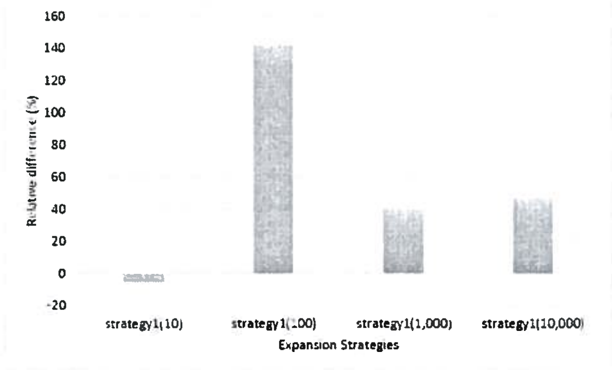

Figure 9: Relative gains on Pens-SF dataset with *minsup*=10. Strategy1 and different X values.

elimina anche il dubbio se invertire gli esperimenti coi due dataset come suggeriva Paolo. #Daniele @Fabio: per me ok eliminare, propongo tabella come Pietro @TANIA: secondo me con la tabella si vede ancora di piu che i risultati a volte sono molto negativi, tanto c'e gia' la tabella generica. con paolo proponiamo di lasciarlo solo nel testo.

5.3. ~~Running time~~ — EXECUTION ~~Rutta~~, Here we analyze

~~After the identification of a good trade-off strategy in the previous section.~~ ~~we analyze~~ the efficiency of PaMPa-HD by comparing it with three distributed state-of-the-art frequent itemset mining algorithms:

1. Parallel FP-growth [17] available in Mahout 0.9 [18], based on FP-

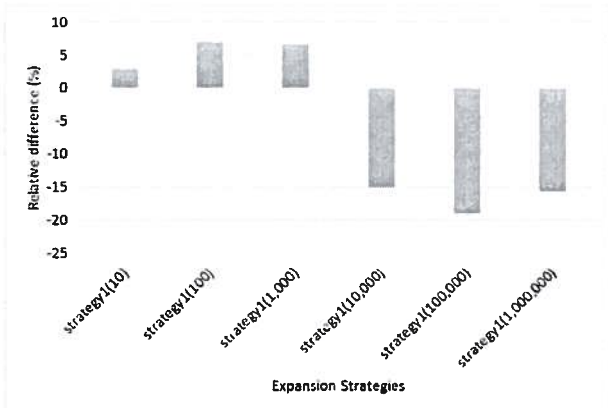

Figure 10: Relative gains on Breast Cancer dataset with *minsup*=5. Strategy1 and different X values.

growth algorithm [12]

2. DistEclat [19], based on Eclat algorithm [20]
3. BigFIM [19], inspired from Apriori [11] and DistEclat

This set of algorithms represents the most cited implementations of frequent itemset mining distributed algorithms. All of them are Hadoop-based and are designed to extract the frequent closed itemsets (DistEclat and BigFIM actually extract a superset of the frequent closed itemsets). The parallel implementation of these algorithms has been aimed to scale in the number of transactions of the input dataset. Therefore, they are not specifically developed to deal with high-dimensional datasets as PaMPa-HD. For details about the algorithms, see Section 6.

The first set of experiments has been performed with the 100-rows version PEMS-SF dataset [13] and minsup values 35 to 5.[2]

As shown in Figure 11, in which minsup axis is reversed to improve readability, PaMPa-HD is the only algorithm able to complete all the mining task to a minsup value of 5 rows or 5%. All the approaches show similar behaviors with high minsup values (from 30 to 35). With a minsup of 25, PFP shows a strong performance degradation, being not able to complete the mining. In a similar way, BigFIM shows a performance degradation with a minsup of 20, running out of memory with a minsup of 15. DistEclat, instead, shows very interesting execution time until running out of memory with a minsup of 10. PaMPa-HD, even if slower than DistEclat with minsup values from 25 to 15, is able to complete all the tasks.

The second set of experiments are performed with the Breast Cancer dataset [15]. As reported in Figure 12 (Even in this case, minsup axis is reversed to improve readability, the minsup is absolute), PaMPa-HD is the most reliable and fast approach. This time, BigFIM is not able to cope either with the highest minsup values, while PFP shows very slow execution times and runs out of memory with a minsup value of 6. DistEclat is able to achieve good performances but it is always slower than PaMPA-HD (with a minsup value equal to 4, it is not able to complete the mining within several days of computation). From these results, we have seen how traditional best-in-class approaches such as BigFIM, DistEclat and PFP are not suitable for high-dimensional datasets. They are slow and/or not reliable when coping with the curse of dimensionality. PaMPa-HD, instead, demonstrated to be most suitable approach with datasets characterized by a high number of items and a small number of rows. After the comparison with the state of the art distributed frequent itemset mining algorithm, the next experimental subsections will experimentally describe the behavior of PaMPa-HD with respect to the number of transactions, number of independent tasks, communication costs and load balancing.

[2]The algorithms parameters, which will be introduced in Section 6, has been set in the following manner. PFP has been set to obtain all the closed itemsets; the prefix length of the first phase of BigFIM and DistEclat, instead, has been set to 3, as suggested by the original paper [19], when possible (i.e. when there were enough 3-itemsets to execute also the second phase of the mining).

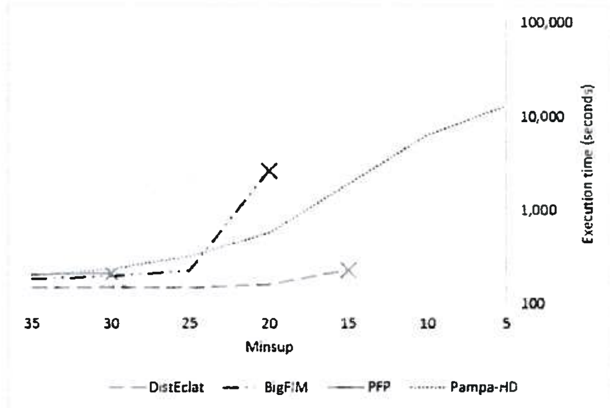

Figure 11: Execution time for different Minsup values on the PEMS-SF dataset (100-rows).

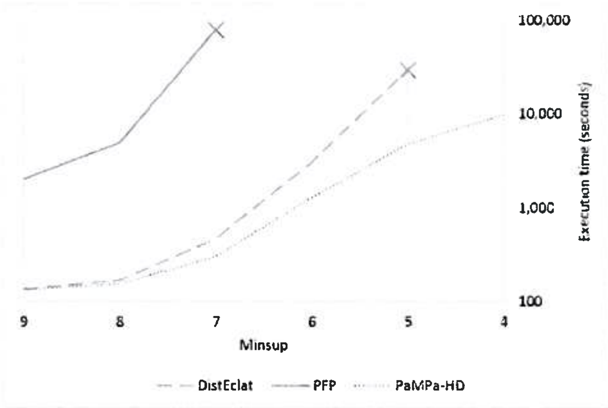

Figure 12: Execution time for different Minsup values on the Breast Cancer dataset.

5.4. *Impact of the number of transactions*

This set of experiments measures the impact of the number of transactions on PaMPa-HD performances. At this aim, it will be used the PEMS-SF datasets in three versions (100-rows, 200-rows and full). The algorithm is very sensitive to this factor: the reasons are related to its inner structure. In fact, the enumeration tree, for construction, is strongly affected by the number of rows. A higher number of rows leads to:

1. A higher number of branches. As shown in the example in Figure 2, from the root of the tree, it is generated a new branch for each tid (transaction-id) of the dataset.
2. Longer and wider branches. Since each branch explores its research subspace in a depth-first order, exploring any combination of tids, each

branch would result with a greater number of sub-levels (longer) and a greater number of sub-branches (wider)

Therefore, the mining processes related to the 100-rows version and to the 200-rows or the full version of PEMS-SF dataset are strongly different. With a number of rows incremented by, respectively, 200% and more of the 400%, the mining of the augmented versions of PEMS-SF dataset is very challenging for the enumeration-tree based PaMPa-HD. The performance degradation is resumed in Figure 13, where, for instance, with a minsup of 35%, the execution times related to the 100-rows and the full version of the PEMS-SF dataset differ of almost two orders of magnitude.

The behavior and the difficulties of PaMPa-HD with datasets with an incremental number of rows, is, unfortunately, predictable. This algorithmic problem represents a challenging and interesting open issues for further developments.

5.5. *Impact of the number of nodes*

The impact of the number of independent tasks involved in the algorithm execution is a non-trivial issue. Adding a task to the computation would not only deliver more resources such as memory or CPU, but it also leads to split the chunk of the enumeration tree that is explored by each task. On one hand, this means to reduce the search space to explore, lightening the task load. On the other hand, this reduces the state centralized memory and the impact of the related pruning. It can be interpreted as a trade-off between the benefits of the parallelism against the state. In Figure 14 and Figure 15, it is reported the behavior of PaMPa-HD with a mining process on

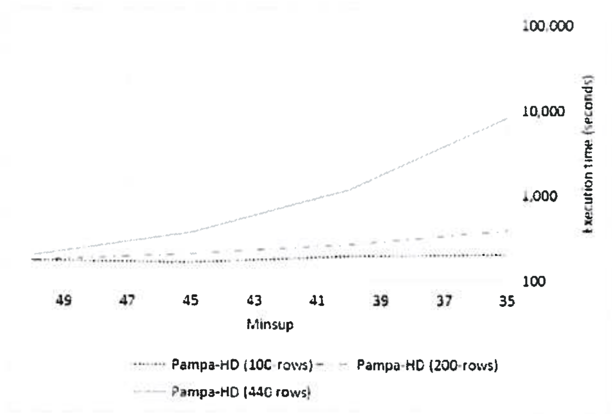

Figure 13: Execution times for different version of the PEMS-SF for PaMPa-HD.

the datasets PEMS-SF and Breast Cancer. The minsup values, respectively of 20 and 6, have been chosen in order to be deep enough to show performance differences among the different degree of parallelism. Interestingly, the mining on PEMS-SF dataset is less sensitive to the number of reducers, with an execution time that is just halved when the independent tasks included in the computation pass from 1 to 17. The experiment of Breast Cancer instead, Figure 15, shows a stronger performance gain. ~~(Lo stesso esperimento per quanto riguarda PEMS l'ho fatto con un minsup~~ piu' basso e la linea era ancora pi orizzontale) The behavior is related to the dataset data distribution. For PEMS-SF dataset, the advantages related to additional independent nodes into the mining is mitigated by the loss of state in the local pruning phase inside the nodes. With additional

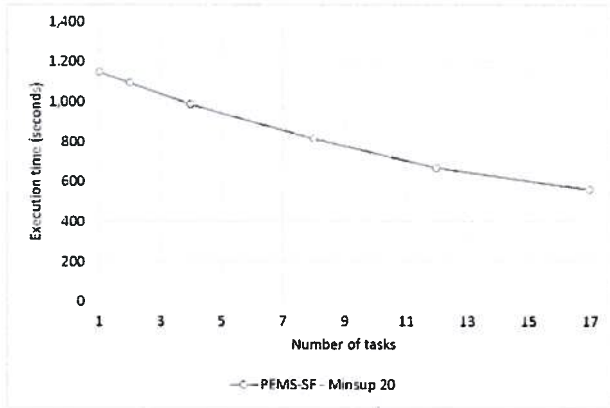

Figure 14: Execution times for PEMS-SF datasets with different number of parallel tasks.

nodes, each node is pushed to a smaller exploration of the search space, decreasing the effectiveness of the local pruning. These specific results recall a very popular open issue in distributed environments. In problems characterized by any kind of "state" benefit (in this case, the local pruning inside the tasks), a higher degree of parallelism does not lead to better performance a priori.

5.6. *Load Balancing and communication costs*

The last analysis are related to the load balancing and the communication costs of the algorithm. These issues ~~are very underestimated but they~~ represent very important factor in such a distributed environment. Communication costs ~~represent~~ are among the main bottlenecks related to the

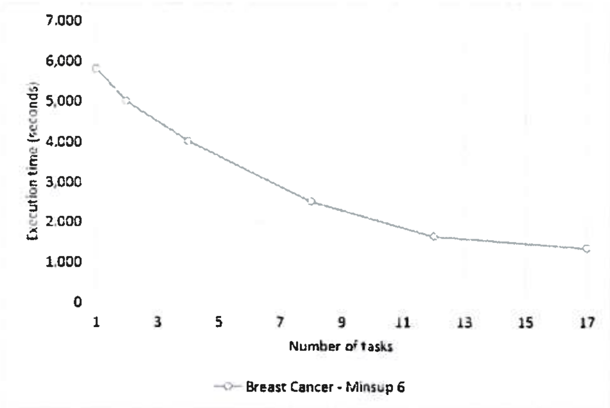

Figure 15: Execution times for Breast Cancer dataset with different number of parallel tasks.

performance of parallel algorithms [21]. A bad-balanced load among the independent tasks leads to few long tasks that block the whole job.

PaMPa-HD, being based on Carpenter algorithm, ~~as shown in the previous sections~~, mainly consists on the exploration of an enumeration tree. The basic idea behind the parallelization is to explore the main branches of the tree independently within parallel tasks (Figure 3). For this reason, each task needs the information (i.e. transposed tables) related to its branch expansion. The ideal behavior of a distributed algorithm would be to distribute the least amount of data, avoiding redundant informations as much as possible. The reason is that network communications are very costly in a Big Data scenario. Unfortunately, the structure of the enumeration tree of PaMPa-HD

assumes that some pieces of data of the initial dataset is sent to more than one tasks. For instance, some data related to nodes $TT|_2$ and $TT|_3$ are the same, because from node $TT|_2$ will be generated the node $TT|_{2,3}$. This is an issue related to the inner structure of the algorithm and a full independence of the initial data for each branch cannot be reached.

In addition, the architecture of the algorithm with its synchronization phase, burdens of the I/O costs. In fact, in order to prune some useless tables and improve the performances, the mining process is divided in more phases writing the partial results into HDFS. However, as we have already seen when studying the impact of the *max_exp* (Figure 7 and Figure 8), in some cases additional synchronization phases leads to better execution times, despite their related overhead. In Figure 16 and Figure 17 ~~it is shown~~ the communication cost during a mining process. The spikes are related to the shuffle phases, in which the redundant tables and closed itemsets are removed. The flat part of the curve between the spikes is longer in the case is reported
of Breast Cancer dataset because of the adopted strategy. Its mining has been executed with a more aggressive increasing of the *max_exp* parameter (steps of 10 for PEMS-SF dataset, 10,000 for Breast Cancer dataset), which leads to a very long period without synchronization phases.

The load balancing is evaluated comparing the execution time of the fastest and slowest tasks related to the iteration job in which this difference is strongest. The most unbalanced phase of the job is, not surprisingly, the mapper phase of the Job 3. This job is iterated until the mining is complete and it is the one more involved by the increasing of increasing of the *max_exp* parameter (iterations characterized by high *max_exp* value are

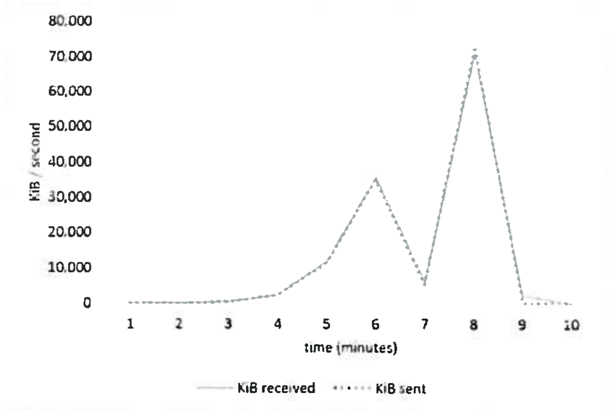

Figure 16: Received and sent data in the commodity cluster network during PEMS-SF dataset mining, minsup=20.

likely characterized by long and unbalanced task). The difference among the fastest and the slowest mapper, as shown by Table 5. It is clear that the mining on PEMS-SF dataset is more balanced among the independent tasks. Even in this case, the reason is the different increment value in the Strategy #1 (10 for PEMS-SF dataset, 10,000 for Breast Cancer dataset). A slower *max_exp* increasing leads to more balanced tasks.

6. Related work

~~**Questo tutto vecchio, rifrasare? #Daniele: secondo me se alcune parti sono uguali, non e' un problema, perche' e' roba tua e hai cambiato tanto nel resto. Sicuramente aggiungere qualche**~~

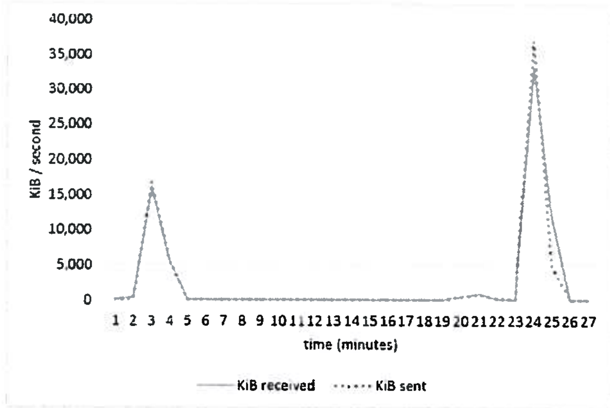

Figure 17: Received and sent data in the commodity cluster network during Breast Cancer dataset mining, minsup=6.

~~**citazione, visto che e' passato tempo e questa non e' una conferenza, credo sia utile / importante. FABIO: l'avevo fatto tempo fa, stasera lo rifaccio per sicurezza**~~

Frequent itemset mining represents a very popular data mining technique used for exploratory analysis. Its popularity is witnessed by the high number of approaches and implementations. The most popular techniques to extract frequent itemsets from a transactional datasets are Apriori and Fp-growth. Apriori [11] is a bottom up approach: itemsets are extended one item at a time and their frequency is tested against the dataset. FP-growth [12], instead, is based on an FP-tree transposition of the transactional dataset and a recursive divide-and-conquer approach. These techniques explore the

Table 5: Load Balancing

Dataset	Slowest Task Execution time	Fastest Task Execution time
PEMS-SF	3mins 58 sec	3mins 37sec
Breast Cancer	20mins 33sec	8mins 42sec

search space enumerating the items. For this reason, they work very well for datasets with a small (average) number of items per row, but their running time increases exponentially with higher (average) row lengths [11, 20].

In recent years, the explosion of the so called Big Data phenomenon has pushed the implementation of these techniques in distributed environments such as Apache Hadoop [3], based on the MapReduce paradigm [22], and Apache Spark [4]. Parallel FP-growth [17] is the most popular distributed closed frequent itemset mining algorithm. The main idea is to process more sub-FP-trees in parallel. A dataset conversion is required to make all the FP-trees independent. A Spark implementation of Parallel FP-growth has been delivered with MLlib Library [23]. This version extracts all the frequent itemsets and not just the closed ones. BigFIM and DistEclat [19] are two recent methods to extract frequent itemsets. DistEclat represents a distributed implementation of the Eclat algorithm [20] an approach based on equivalence classes (groups of itemsets sharing the same prefixes), smartly merged to obtain all the candidates. BigFIM is a hybrid approach exploiting both Apriori and Eclat paradigms. BigFIM and DistEclat are divided in two phases. In the first one, the approaches use respectively an Apriori-like an Eclat-like strategy to mine the itemsets up to a fixed k-length. After that, the itemsets are distributed and used as prefixes for the longer itemsets. The last phase of the mining, both the approaches uses Eclat to extract all the closed itemsets. In addition, [24] introduces another Apriori-based frequent itemset miner. The contribution of this work is focused on the candidates handling, which are cached in memory ~~between each iteration.~~

(e.g. ~~load~~ load balancing, robustness to memory issues)

While the previous works have been designed for use cases characterized by datasets with a large amount of transactions, Carpenter algorithm [5], which inspired PaMPa-HD, has been specifically designed to extract frequent itemsets from high-dimensional datasets, i.e., characterized by a very large number of attributes (in the order of tens of thousands or more). The basic idea is to investigate the row set space instead of the itemset space. A detailed introduction to the algorithm is presented in section 3. This work extends our previous work [9]. The original algorithm assumes a slightly different architecture, assuming an additional independet synchronization job at each iteration. As already described in Section4.1, this implementation includes the synchronization phase in the Mining Job 3. Therefore, the number of MapReduce jobs (with their related overhead) are strongly reduced. Additionally, in order to better exploit the pruning rule in the local Carpenter iteration in each independent task, all the transposed tables are now processed (not only expanded) in depth-first order. This strategy decreases the possibility to explore an useless branch of the tree, i.e. a branch whose results would be completely overwritten by the closed itemsets obtained by branches older in depth-first fashion.

Io lo eliminerei

ALTERNATIVA

The idea of designing a parallel MapReduce ~~based~~ algorithm to efficiently support itemset mining on high dimensional data was first introduced in [9]. The PaMPa algorithm significantly enhances the algorithm performance proposed in [9] by providing (1) a more efficient approach to address the synchronization phase in the mining process; (2) a lower number of Map-Reduce jobs; and (3) a more efficient visit of the transposed tables. (@FABIO: controlla che sia giusto). Furthermore a variety of issues related to the distributed algorithms has been throughly evaluated on real datasets.

algorithms

7. Applications

exploits

Since PaMPa-HD is able to process extremely high-dimensional datasets, it enriches the set of tools able to deal with ~~high-dimensional environments~~ datasets characterized by a large variety of features. (e.g. [25], [26]). Consequently, many fields of applications which ~~are~~ based ~~on data mining tools and specifically on~~ frequent itemset and association rules [27], [28] could benefit of it. The first example is bioinformatics and health environments: researchers in this environment often cope with data structures defined by a large number of attributes, which matches gene expressions, and a relatively small number of transactions, which typically represent medical patients or tissue samples. Furthermore, smart cities and computer vision ~~environments~~ applications are two important ~~application~~ domains which can benefit from our distributed algorithm, thanks to their heterogeneous nature. Another field of application is the networking domain. Some examples of interesting high-dimensional dataset are URL reputation, advertisements, social networks and search engines. One of the most interesting applications, which we plan to investigate in the future, is related to internet traffic measurements. Currently, the market offers an interesting variety of internet packet sniffers like [29], [30]. Datasets, ~~in which the transactions represent~~ INCLUDE REPRESENT TRAFFIC flows and the item are flow attributes, ~~are already~~ a ~~very promising appli-~~ ~~cation~~ domain for data mining techniques [31], [32], [33].

to discover hidden correlations

domain

AN APPEALING DOMAIN WHERE PAMPA can be effectively exploited.

Collected

8. Conclusion

This work introduced PaMPa-HD, a novel frequent closed itemset mining algorithm able to efficiently parallelize the itemset extraction from extremely high-dimensional datasets. Experimental results show its scalability

good

efficient

and its performance in dealing with real-world datasets characterized by up to 8 millions different items and, above all, an average number of items per transaction over hundreds of thousands, on a small commodity cluster of 5 nodes. PaMPa-HD outperforms state-of-the-art algorithms, by showing a better scalability than all popular distributed approaches, such as PFP, DistEclat and BigFIM. Further developments of the framework can be related to the introduction of new pruning rules in specific use cases. This pruning, so far related to the post processing phase, would avoid the processing of useless data. **#Daniele @Fabio: non c'e' nessun altro futur work che ti viene in mente? solo pruning? Pietro diceva nei commenti di riprendere le open issue della scalabilita (vedi esperimento), magari possiamo mettere qualcosa su quello, tipo ridurre l'impatto del local state che limita la scalabilita...?** CONCORDO **Fabio: ci penso, tanto per dire. Quello del local state secondo me abbastanza insormontabile e caratterizza tutti gli alg distribuiti. ma ci sta. La mia idea sarebbe di cercare un modo per non fare la distribuzione per ramo ma per profondit. Lo scrivo io.** OK

Acknowledgement

The research leading to these results has received funding from the European Union under the FP7 Grant Agreement n. 619633 (Project "ONTIC").

References

[1] X. Jin, B. W. Wah, X. Cheng, Y. Wang, Significance and challenges of big data research, Big Data Research 2 (2) (2015) 59 – 64, visions on Big Data. doi:http://dx.doi.org/10.1016/j.bdr.2015.01.006.
URL http://www.sciencedirect.com/science/article/pii/S2214579615000076

[2] O. Y. Al-Jarrah, P. D. Yoo, S. Muhaidat, G. K. Karagiannidis, K. Taha, Efficient machine learning for big data: A review, Big Data Research 2 (3) (2015) 87–93. doi:10.1016/j.bdr.2015.04.001.
URL http://dx.doi.org/10.1016/j.bdr.2015.04.001

[3] D. Borthakur, The hadoop distributed file system: Architecture and design, Hadoop Project 11 (2007) 21.

[4] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, I. Stoica, Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, in: NSDI'12, 2012, pp. 2–2.

[5] F. Pan, G. Cong, A. K. H. Tung, J. Yang, M. J. Zaki, Carpenter: Finding closed patterns in long biological datasets, in: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '03, ACM, New York, NY, USA, 2003, pp. 637–642. doi:10.1145/956750.956832.
URL http://doi.acm.org/10.1145/956750.956832

[6] J.-G. Lee, M. Kang, Geospatial big data: Challenges and opportunities, Big Data Research 2 (2) (2015) 74 – 81, visions on Big Data. doi:http://dx.doi.org/10.1016/j.bdr.2015.01.003.
URL http://www.sciencedirect.com/science/article/pii/S2214579615000040

[7] M. Cuturi, UCI machine learning repository: PEMS-SF data set (2011).
URL https://archive.ics.uci.edu/ml/datasets/PEMS-SF

[8] J. Leskovec, A. Krevl, SNAP Datasets: Stanford large network dataset collection, http://snap.stanford.edu/data (Jun. 2014).

[9] D. Apiletti, E. Baralis, T. Cerquitelli, P. Garza, P. Michiardi, F. Pulvirenti, Pampa-hd: A parallel mapreduce-based frequent pattern miner for high-dimensional data, in: IEEE ICDM Workshop on High Dimensional Data Mining (HDM), Atlantic City, NJ, USA, 2015. doi:10.1109/ICDMW.2015.18.
URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7395755

[10] Pang-Ning T. and Steinbach M. and Kumar V., Introduction to Data Mining, Addison-Wesley, 2006.

[11] R. Agrawal, R. Srikant, Fast algorithms for mining association rules in large databases, in: VLDB '94, 1994, pp. 487–499.

[12] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, in: SIGMOD '00, 2000, pp. 1–12.

[13] M. Lichman, UCI machine learning repository (2013).
URL http://archive.ics.uci.edu/ml

[14] California department of transportation.

URL http://http://pems.dot.ca.gov/. Last access: April, 21st 2016

[15] M. L. data set repository, Breast cancer dataset (kent ridge.
URL http://mldata.org/repository/data/viewslug/breast-cancer-kent-ridge-2
Last access: July, 15th 2015 ← Scrivere una data più recente

[16] V. Jacobson, Congestion avoidance and control, SIGCOMM Comput. Commun. Rev. 18 (4) (1988) 314–329. doi:10.1145/52325.52356.
URL http://doi.acm.org/10.1145/52325.52356

[17] H. Li, Y. Wang, D. Zhang, M. Zhang, E. Y. Chang, PFP: parallel fp-growth for query recommendation, in: RecSys'08, 2008, pp. 107–114.

[18] Apache Software Foundation, Apache mahout:: Scalable machine-learning and data-mining library [online, cited 2016-03-15].

[19] S. Moens, E. Aksehirli, B. Goethals, Frequent itemset mining for big data, in: SML: BigData 2013 Workshop on Scalable Machine Learning, IEEE, 2013.

[20] M. J. Zaki, S. Parthasarathy, M. Ogihara, W. Li, New algorithms for fast discovery of association rules, in: KDD'97, AAAI Press, 1997, pp. 283–286.

[21] A. D. Sarma, F. N. Afrati, S. Salihoglu, J. D. Ullman, Upper and lower bounds on the cost of a map-reduce computation, Proc. VLDB Endow. 6 (1) (2013) 277–288. doi:10.14778/2535570.2488334.
URL http://dx.doi.org/10.14778/2535570.2488334

[22] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, in: OSDI'04, 2004, pp. 10–10.

[23] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. B. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, A. Talwalkar, MLlib: Machine learning in apache spark (May 2016). arXiv:1505.06807.
URL http://arxiv.org/abs/1505.06807

[24] H. Qiu, R. Gu, C. Yuan, Y. Huang, Yafim: A parallel frequent itemset mining algorithm with spark, in: Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International, IEEE, 2014, pp. 1664–1671.

[25] R. Vimieiro, P. Moscato, A new method for mining disjunctive emerging patterns in high-dimensional datasets using hypergraphs, Information Systems 40 (2014) 1 – 10. doi:http://dx.doi.org/10.1016/j.is.2013.09.001.
URL http://www.sciencedirect.com/science/article/pii/S0306437913001221

[26] P. Bermejo, L. de la Ossa, J. A. Gmez, J. M. Puerta, Fast wrapper feature subset selection in high-dimensional datasets by means of filter re-ranking, Knowledge-Based Systems 25 (1) (2012) 35 – 44, special Issue on New Trends in Data Mining. doi:http://dx.doi.org/10.1016/j.knosys.2011.01.015.
URL http://www.sciencedirect.com/science/article/pii/S095070511100027X

[27] J. Nahar, T. Imam, K. S. Tickle, Y.-P. P. Chen, Association rule

mining to detect factors which contribute to heart disease in males and females. Expert Systems with Applications 40 (4) (2013) 1086 – 1093. doi:http://dx.doi.org/10.1016/j.eswa.2012.08.028.
URL http://www.sciencedirect.com/science/article/pii/S095741741200989X

[28] B. Kamsu-Foguem, F. Rigal, F. Mauget, Mining association rules for the quality improvement of the production process, Expert Systems with Applications 40 (4) (2013) 1034 – 1045. doi:http://dx.doi.org/10.1016/j.eswa.2012.08.039.
URL http://www.sciencedirect.com/science/article/pii/S0957417412010007

[29] A. Finamore, M. Mellia, M. Meo, M. Munafò, D. Rossi, Experiences of internet traffic monitoring with tstat, IEEE Network 25 (3) (2011) 8–14.

[30] B. Claise, Cisco systems netflow services export version 9, rfc 3954 (informational) (2004).

[31] D. Apiletti, E. Baralis, T. Cerquitelli, S. Chiusano, L. Grimaudo, Searum: A cloud-based service for association rule mining, in: Proceedings of the 2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TRUSTCOM '13, IEEE Computer Society, Washington, DC, USA, 2013, pp. 1283–1290. doi:10.1109/TrustCom.2013.153.
URL http://dx.doi.org/10.1109/TrustCom.2013.153

[32] D. Brauckhoff, X. Dimitropoulos, A. Wagner, K. Salamatian, Anomaly extraction in backbone networks using association rules,

Networking, IEEE/ACM Transactions on 20 (6) (2012) 1788–1799. doi:10.1109/TNET.2012.2187306.

[33] D. Apiletti, E. Baralis, T. Cerquitelli, V. D'Elia, Characterizing network traffic by means of the netmine framework, Comput. Netw. 53 (6) (2009) 774–789. doi:10.1016/j.comnet.2008.12.011.
URL http://dx.doi.org/10.1016/j.comnet.2008.12.011