PaMPa-HD: EXTENSION - TO MODIFY

Daniele Apiletti, Elena Baralis, Tania Cerquitelli,
Paolo Garza, Fabio Pulvirenti[1], and Pietro Michiardi[1]

[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 is among the most complex exploratory techniques in data mining, and provides the ability to discover hidden correlations in transactional datasets. The explosion of Big Data is leading to new parallel and distributed approaches. Unfortunately, most of them are designed to cope with low-dimensional datasets, whereas no distributed high-dimensional frequent closed itemset mining algorithms exists. This work introduces PaMPa-HD, a parallel MapReduce-based frequent closed itemset mining algorithm for high-dimensional datasets, based on Carpenter. The experimental results, performed on both real and synthetic datasets, show the efficiency and scalability of PaMPa-HD.

Keywords:

1. Experiments

We performed a set of experiment to evaluate the performance of the proposed algorithm. Firstly, we measured the performance impact of the maximum expansion threshold, evaluating the quality of a set of proposed strategies. After that, we measured the efficiency of the proposed algorithm, comparing it with the state of the art distributed approaches (Section **??**). After that, we will focus on the deepest execution details of our approach. Specifically, we will experimentally describe the impact of the number of transactions of the input dataset on the performance of PaMPa-HD (Section **??**). Then, we will meausre the performance impact with respect to the number of parallel tasks (see Section **??**), the communication costs and load balancing behavior, very important in such a distributed context.

We performed the experiments on two real life datasets. The first real dataset is the **PEMS-SF** dataset [**?**], 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 [**?**]). Each transaction represents the daily traffic rate of 963 lanes, sampled every 10 minutes. It is characterized of 440 rows and 138672 attributes (6 x 24 x 963), and it has been discretized in equi-width bins of 0.001. Because of the nature of PaMPa-HD, which is designed to cope with high-dimensional datasets characterized by a small number of transactions, we have used several reduced version (in terms of number of rows) to measure the impact of the number of transactions on the performance of the algorithm.

The second real dataset is the **Kent Ridge Breast Cancer** [**?**], 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, floating point). Data have been discretized with an equal depth partitioning using 20 buckets (similarly to [**?**]).

The discretized version of the real dataset and the synthetic dataset generator are publicly available at http://dbdmg.polito.it/PaMPa-HD/. **TO DO: aggiungere versione finale discretized pemsf**

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 CancerDataset	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.

1.1. 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 **??**, 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), 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 performed two set of experiments. In the first one we have performed the mining on the PEMS-SF (100 transactions) dataset with a Minsup 50, by varying *max_exp* from 100 to 100,000,000. The minsup value was empirically selected in order to let the mining problem being deep enough to show different performances. In Figure **??** are shown the results in terms of execution time and number of iterations (i.e., the number of jobs).

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

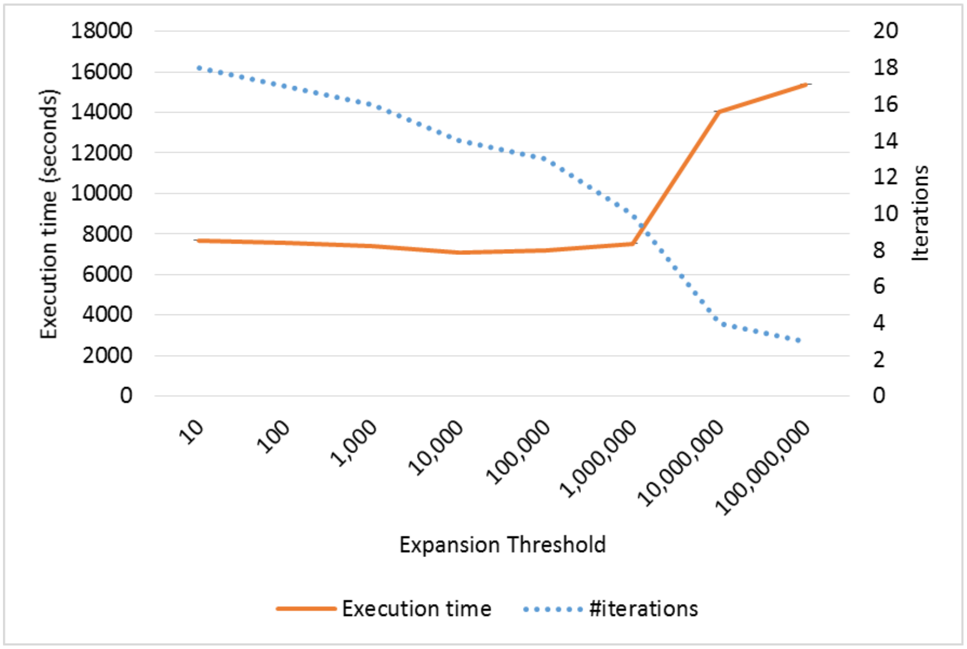

Figure 1: Execution time and number of iterations for different *max_exp* values on PEMS-SF dataset with *minsup*=50.

synchronization phase. Increasing the *max_exp* parameter 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 **??**, 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

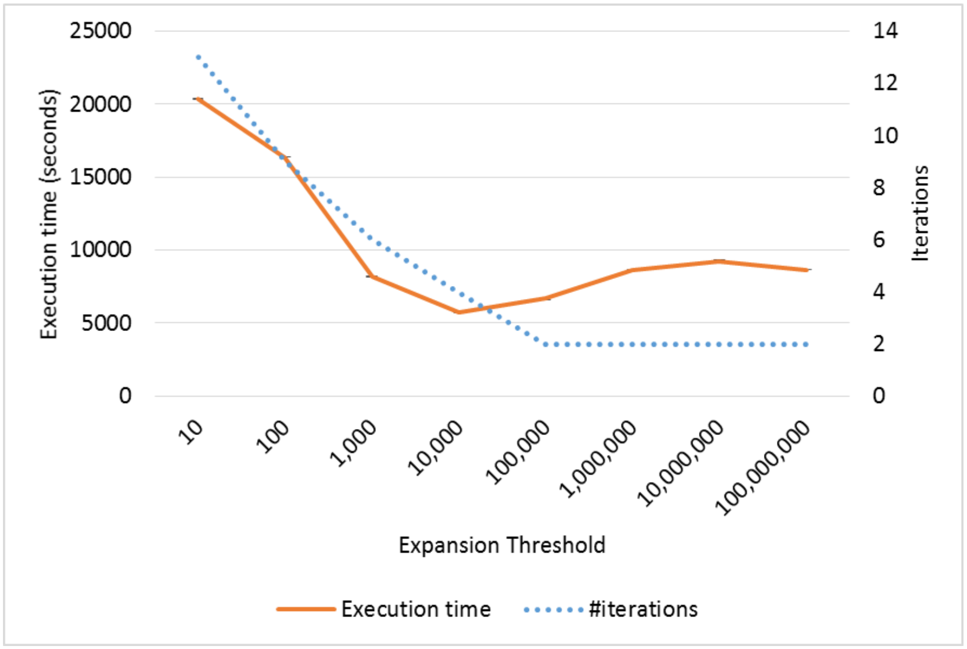

Figure 2: Execution time and number of iterations for different *max_exp* values on Breast Cancer dataset with *minsup*=5.

computation among different nodes. With low values of *max_exp*, each task explores a smaller enumeration sub-tree, decreasing the size difference among the sub-trees analyzed by different tasks, thus improving the load balancing. Table **??** 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 **??** and **??**, it is very dependent on the use case and distribution of the data. In the next subsection we will 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	44s	2h 20m 28s
10	6m 21s	45m 16s	6s	2m 24s

1.2. Proposed strategies

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 tune up the the *max_exp* parameter. Before the introduction of the techniques, let us motivate the reasons behind their design. 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 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.

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 informations 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 [?]). 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 empirically from RTT and other values). 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. **Non siamo sicuri che convenga inserire questa parte sul time out**. 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 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.

Table 3: Strategies performance

Strategies	PEMS-SF	Breast Cancer
Strategy #1	-42.13 % (X = 10,000)	-19.03 (X = 100,000)
Strategy #2	-30.63 % (X = 10,000)	-0.02 % (X = 10,000)
Strategy #3	-6.97 % (X = 100)	+1.59 % (X = 100)
Strategy #4	-42.03 % (X = 1,000)	-16.17 % (X = 1,000)

Although, in this way, we are limiting to 1 hour the amount of time in which we are not completely exploiting the resource of the commodity cluster (i.e. only few very long tasks running). A value of 1 hour has been empirically proven to be a good tradeof 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 **??** are resumed 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 very good performances. For Breast Cancer dataset strategy #1 is the best, followed by strategy #4: these are the only ones achieving significative positive gain over the fixed *max_exp* approach. All the strategies are evaluated with X from 10 to 100,000. The max value has been increased in the cases in which the performance suggest a decreasing

Table 4: 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

execution time trend.

Since the best performances are achieved with values of 10,000 and 100,000 respectively for PEMS-SF and Breast Cancer datasets (Figures **??** and **??**), we will use this configuration for the experiments comparing PaMPa-HD with the other distributed approaches.

1.3. Running time

After the identification of a good tradeof strategy in the previous section, we have used it to analyze the efficiency of PaMPa-HD comparing it with three distributed state-of-the-art frequent itemset mining algorithms:

1. Parallel FP-growth [**?**] available in Mahout 0.9 [**?**], based on FP-Growth algorithm [**?**]
2. DistEclat [**?**], based on Eclat algorithm [**?**]
3. BigFIM [**?**], inspired from Apriori [**?**] and DistEclat

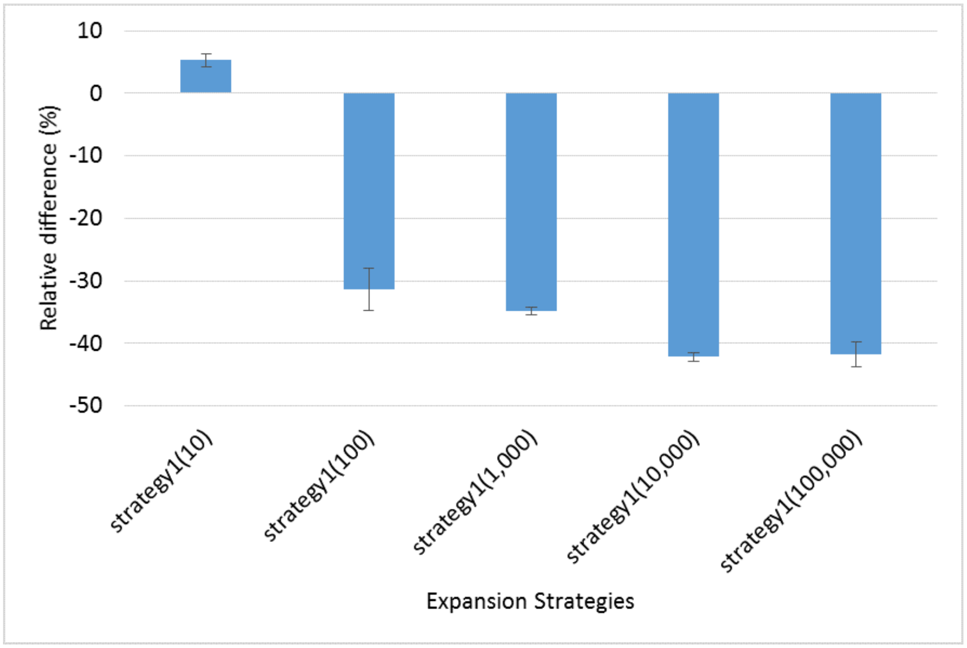

Figure 3: Relative gains on Pems-SF dataset with *minsup*=50, Strategy1 and different X values.

This set of algorithms represents the most cited implementation 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 **??**.

The first set of experiments has been performed with the 100-rows version PEMS-SF dataset [**?**] and minsup values 35 to 5. Please note that PFP parameters have been set to obtain all the closed itemsets. The prefix length

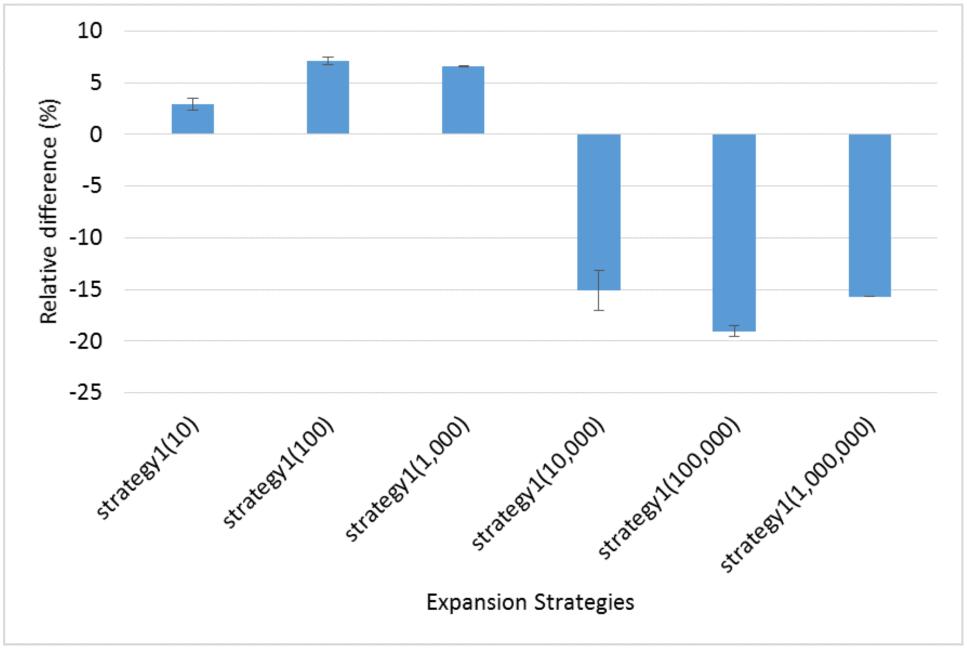

Figure 4: Relative gains on Breast dataset with *minsup*=5, Strategy1 and different X values.

of the first phase of BigFIM and DistEclat, instead, has been set to 3, as suggested by the original paper [?], when possible (i.e. when there were enough 3-itemsets to execute also the second phase of the mining). **Qui parlo di parametri che non ho ancora introdotto, direi di anticipare la parte di related works o metterli in calce.**

As shown in Figure **??**, in which minsup axis is reversed to improve readability, PaMPa-HD is the only algorithm able to complete all the mining task. All the approaches shown similar behaviours with high minsup values (from 30 to 35). With a minsup of 25, PFP shows a strong performance degradation, until running out of memory with a minsup of 20. In a similar way, BigFIM shows a performance degradation with a minsup of 20, running

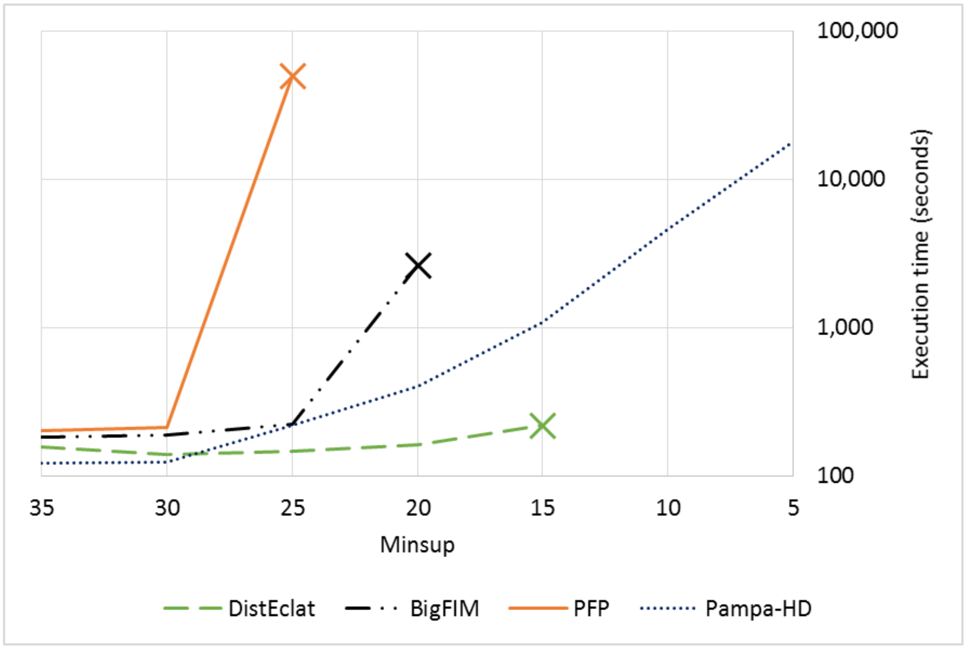

Figure 5: Execution time for different Minsup values on the PEMS-SF dataset (100-rows).

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 has been performed with the Kent Ridge Breast Cancer dataset [?]. As reported in Figure **??** (Even in this case, minsup axis is reversed to improve readability), 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, until it runs out of memory with a minsup of 4. From these results, we have seen how traditional

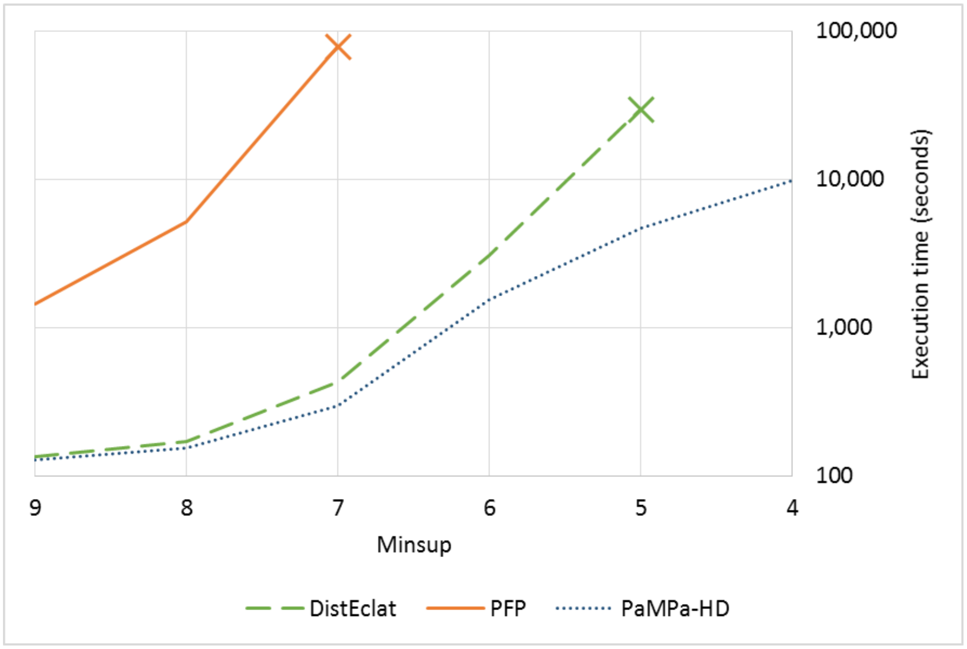

Figure 6: Execution time for different Minsup values on the Breast Cancer dataset.

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 cost and load balancing.

1.4. Impact of the number of transactions

This set of experiments measures the impact of the number of transactions on PaMPa-HD performances. The algorithm is very sensitive to this parameter: 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 **??**, 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 **??**, 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 behaviour and the difficulties of PaMPa-HD with datasets with an increment of the number of rows of is, unfortunately, predictable. This algorithmic problem represents a challenging and interesting open issues for futher developments.

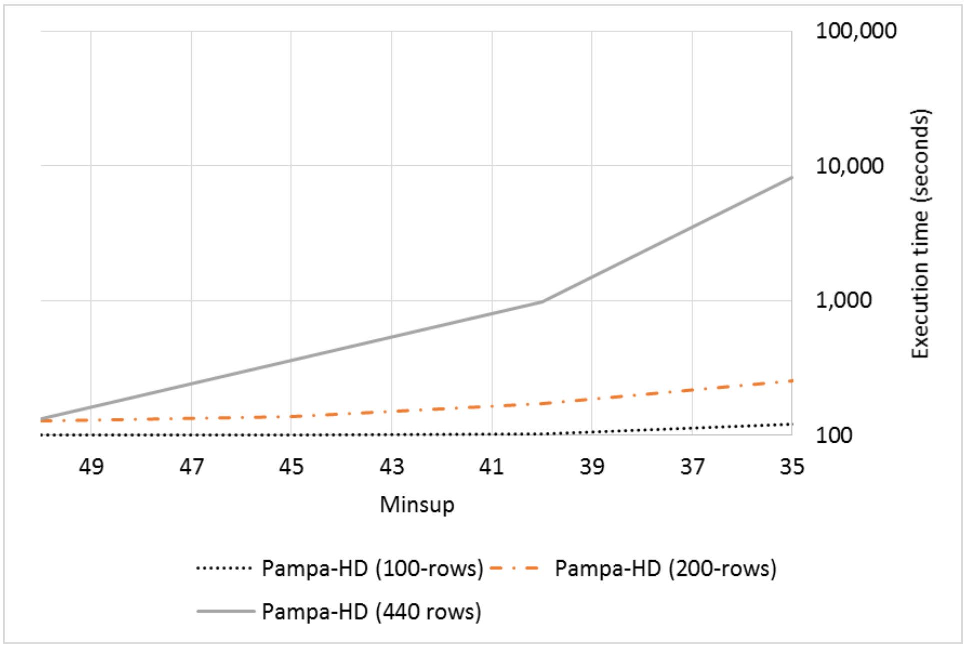

Figure 7: Execution times for different version of the PEMS-SF for PaMPa-HD.

1.5. Scalability

1.6. Communication Cost

1.7. Load Balance

2. Related work

Questo tutto vecchio, rifrasare? 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 [**?**] is a bottom up approach: itemsets are extended one item at a time and their frequency is tested against the dataset. FP-growth [**?**], instead, is based on an FP-tree transposition of the transactional dataset and a recursive divide-and-conquer approach. These techniques explore the 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 [? ?].

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 [?], based on the MapReduce paradigm [?], and Apache Spark [?]. Parallel FP-growth [?] 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. BigFIM and DistEclat [?] are two recent methods to extract frequent itemsets. DistEclat represents a distributed implementation of the Eclat algorithm [?] 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. Carpenter [?], which inspired this work, 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 **??**. This work is based on the distributed implementation PaMPa-HD [**?**]. The original algorithm assumes a slightly different architecture, assuming an additional independet synchronization job at each iteration. As already described in Section**??**, 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.

3. Applications

Questo tutto vecchio, rifrasare? Lo eliminiamo? Since PaMPa-HD is able to process extremely high-dimensional datasets we believe it is suitable for many application (scientific) domains. The first example is bioinformatics: 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 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 [**?**], [**?**]. Datasets, in which the transactions represent flows and the item are flows attributes, are already a very promising application domain for data mining techniques [**?**],[**?**], [**?**].

4. 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 shos its scalability and its performance in coping with real datasets characterized by up to 8 millions different items and, above all, an average number of items per transaction over a hundred thousands, on a small commodity cluster of 5 nodes. PaMPa-HD outperforms state-of-the-art algorithms, by showing a better scalability than all the state of the art distributed approaches such as PFP, DistEclat and BigFIM. Further developments of the framework can be related to the introduction of new pruning rules related to specific use cases inside the algorithm. This pruning, so far related to the post processing phase, would avoid the processing of useless data.

Acknowledgement

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