A Sketch-based Architecture for Mining Frequent Items and Itemsets from Distributed Data Streams

Eugenio Cesario, Antonio Grillo, Carlo Mastroianni
ICAR-CNR
Rende (CS), ITALY
Email: {cesario,grillo,mastroianni}@icar.cnr.it

Domenico Talia
University of Calabria, ICAR-CNR
Rende (CS), ITALY
Email: talia@deis.unical.it

***Abstract*—This paper presents the design and the implementation of an architecture for the analysis of data streams in distributed environments. In particular, data stream analysis has been carried out for the computation of items and itemsets that exceed a frequency threshold. The mining approach is hybrid, that is, frequent items are calculated with a single pass, using a sketch algorithm, while frequent itemsets are calculated by a further multi-pass analysis. The architecture combines parallel and distributed processing to keep the pace with the rate of distributed data streams. In order to keep computation close to data, miners are distributed among the domains where data streams are generated. The paper also reports the experimental results obtained with a prototype of the architecture, tested on a Grid composed of two domains handling two different data streams.**

***Keywords*- distributed data mining; frequent items; frequent itemsets; Grids; stream mining**

I. Introduction

Mining data streams is a very important research topic and has recently attracted a lot of attention, because in many cases data is generated by external sources so rapidly that it may become impossible to store it and analyze it offline. Moreover, in some cases streams of data must be analyzed in real time to provide information about trends, outlier values or regularities that must be signaled as soon as possible. The need for online computation is a notable challenge with respect to classical data mining algorithms [1], [2]. Important application fields for stream mining are as diverse as financial applications, network monitoring, security problems, telecommunication networks, Web applications, sensor networks, analysis of atmospheric data, etc.

A further difficulty occurs when streams are distributed, and mining models must be derived not only for the data of a single stream, but for the integration of multiple and heterogeneous data streams [3]. This scenario can occur in all the application domains mentioned before. For example, in a Content Distribution Network, user requests delivered to a Web system can be forwarded to any of several servers located in different and possibly distant places, in order to serve requests more efficiently and balance the load. In such a context, the analysis of user requests, for example to discover frequent patterns, must be performed with the inspection of data streams detected by different servers. Another notable application field is the analysis of packets processed by the routers of an IP network. In any distributed scenario, it is essential that miners are located as close to data sources as possible, in order to limit the overhead of data communication. When there is the need for performing multiple passes on data, the presence of data cachers can help, provided that they are also distributed appropriately.

Sometimes the rate of a single data stream can be so fast that a single computing node can have difficulties to keep the pace with the generation of data. In these cases, it can be useful to sample the data stream instead of processing all the data [4], but of course this can lower the accuracy of derived models, depending on the sampling frequency and the adopted algorithm. A different, or complementary, solution is to partition a data stream among a set of miners, so that each miner processes only a fraction of data. This solution can be achieved by parallelizing the computation over the nodes of a cluster, or can also be implemented by exploiting the multiple CPUs/GPUs offered by modern multicore and manycore machines.

Two important and recurrent problems regarding the analysis of data streams are the computation of *frequent items* and *frequent itemsets* from transactional datasets. The first problem is very popular both for its simplicity and because it is often used as a subroutine for more complex problems. The goal is to find, in a sequence of items, those whose frequency exceeds a specified threshold. When the items are generated in the form of *transactions* — sets of distinct items — it is also useful to discover frequent sets of items. A *k-itemset*, i.e., a set of *k* distinct items, is said to be frequent if those items concurrently appear in a specified fraction of transactions. The discovery of frequent itemsets is essential to cope with many data mining problems, such as the computation of association rules, classification models, data clusters, etc. This task can be severely time consuming, since the number of candidates is combinatorial with their allowed size. The technique usually adopted is to first discover frequent items, and then build candidate itemsets incrementally, exploiting the *Apriori* property [5], which states that an *i*-itemset can be frequent only if all of its subsets are also frequent. While there are some proposals in the literature to mine frequent itemsets in a single pass, it is recognized that in the general case, in which the generation rate is fast, it is very difficult to solve the problem without allowing multiple passes on the data stream [6].

The architecture we designed and present in this paper addresses the issues mentioned above by exploiting the following main features:

- the architecture combines the parallel and distributed paradigms, the first to keep the pace with the rate of a single data stream, by using multiple miners (processors or cores), the second to cope with the distributed nature of data streams. Miners are distributed among the domains where data streams are generated, in order to keep computation close to data.
- the computation of frequent items is performed through *sketch* algorithms. These algorithms maintain a matrix of counters, and each item of the input stream is associated with a set of counters, one for each row of the table, through hash functions. The statistical analysis of counter values allows item frequencies to be estimated with the desired accuracy. Sketch algorithms compute a linear projection of the input: thanks to this property, sketches of data can be computed separately for different stream sources, and can then be integrated to produce the overall sketch [7].
- the approach is *hybrid*, meaning that frequent items are calculated online, with a single pass, while frequent itemsets are calculated by a further multi-pass analysis. This kind of approach allows important information to be derived on the fly without imposing too strict time constraints on more complex tasks, such as the extraction of frequent k-itemsets, as this could excessively lower the accuracy of models.
- to support the mentioned hybrid approach, the architecture exploits the presence of *data cachers* on which recent data can be stored. In particular, miners can turn to data cachers to retrieve the statistics about frequent items and use them to identify frequent *sets* of items. To avoid excessive communication overhead, data cachers are distributed and placed close to stream sources and miners.

To the best of our knowledge, this is one of the first attempts to combine the four mentioned characteristics. In particular, we are not aware of attempts to combine the parallel and distributed paradigms in stream mining, nor of implemented systems that adopt the hybrid single-pass/multi-pass approach, though this kind of strategy is suggested and fostered in the recent literature [8]. The major advantages of the proposed architecture are its scalability and flexibility. Indeed, the architecture can efficiently exploit the presence of multiple miners, and can be adapted to the requirements of specific scenarios: for example, the use of parallel miners can be avoided when a single miner can keep the pace of a single stream, and the use of data cachers is not necessary if mining frequent itemsets is not required or if the stream rate is so slow that they can be computed on the fly.

Beyond presenting the architecture, we describe a prototype that implements it and discuss a set of experiments performed in a Grid environment composed of two domains handling two data streams. In this scenario, we computed frequent items and itemsets for two well known datasets, *Kosarak* and *WebDocs*, and we analyzed the processing time when changing the number of miners available in each domain and the rate of the data streams.

The rest of the paper is structured as follows: Section II summarizes the main issues regarding the mining of data streams and illustrates the most common algorithms for the computation of frequent items and itemsets; Section III describes the proposed parallel/distributed architecture for mining data streams and discusses the adopted hybrid approach; Section IV describes the prototype and the testbed scenario, and reports the related results; Section V discusses related work and Section VI concludes the paper and gives hints for future research avenues.

II. Mining Frequent Items and Frequent Itemsets in Distributed Data Streams

Data stream analysis is often performed with randomized and approximated algorithms, since exact and deterministic algorithms would require too much computing time or memory space. Accordingly, data mining algorithms for data stream analysis are generally evaluated with respect to three metrics [7]:

- The **processing time** of the operations that update the data structures and the mining models after the arrival of a new stream item;
- The **storage space** used by the algorithm;
- The **accuracy** of the approximated algorithm, in general specified through two parameters set by the user: the accuracy parameter ϵ and the failure probability δ, which means that the estimation error is at most ϵ with probability $(1 - \delta)$. Of course, processing time and storage size strongly depend on these parameters.

One of the most important issues is the discovery of *frequent items* [9], consisting of identifying the items whose frequency in a stream exceeds a specified fraction σ of the overall stream size. This problem has a huge number of applications in a variety of scenarios: frequent items can be the most popular destinations of IP packets, the most frequent queries submitted to a search engine, the most common values observed by sensors in a wireless environment, etc. Moreover, frequent items are often used as the basis for more complex analysis processes.

The problem is formalized as follows [7]:

PROBLEM STATEMENT. *Given a stream S of n items $e_1, ..., e_n$, where the frequency of an item i is $f_i = |\{j|e_j = i\}|$, and a frequency threshold σ, the ϵ-approximate-frequent-items problem consists of finding the set F of items such that: $F = \{i|f_i \geq (\sigma - \epsilon)n\}$*

Two basic categories of algorithms can be used to solve this problem: the *Counter-Based* and the *Sketch-Based* algorithms. The algorithms of the first type maintain counters for a subset of elements, and counters are updated every time one of these elements is observed in the stream. If the observed element has no associated counter, the algorithm must choose whether to ignore the element or replace an existing counter with a counter for the new item. At the end of the first pass, frequent items will surely be among those associated with counters, but the inverse is not true, which requires at least a second pass to verify which counters actually correspond to frequent items. Some of the most used *Counter-Based* algorithms are the *SpaceSaving* algorithm [10] and the *LossyCounting* algorithm [11].

Conversely, *Sketch-Based* algorithms [9] do not monitor a subset of elements but provide, with a given accuracy, an estimation of the frequency for all stream elements using a matrix of counters C with d rows and w columns. A set of d hash functions $h_1, ..., h_d$ are chosen among a family of *pairwise-independent* functions, and are associated to the different matrix rows. Each item i observed in the stream is mapped, for each row r, to the matrix element $C[r, h_r(i)]$. This counter is then modified depending on the specific sketch algorithm: in *CountMin* [12], at the arrival of a new item i, the counter is incremented as follows (see Figure 1):

$$for\ r \in [1, d] \rightarrow C[r, h_r(i)] += 1$$

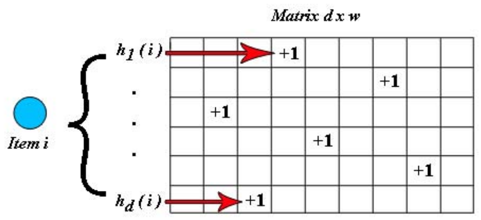


Figure 1. **CountMin Algorithm. A new item is associated, for each row, to a different entry - computed with a hash function - which is then incremented.**

The number of counters in a row, w, is lower than the number of elements, so there are conflicts, because several distinct elements will be mapped by a hash function to the same counter. However, different elements are in conflict for different rows, which enables the adoption of statistical techniques to estimate the actual frequencies of elements. In *CountMin*, collisions always cause extra increments of counters, therefore the best estimation for the frequency f_i of element i is the minimum value of the counters associated to i:

$$f_i = min_r\{C[r, h_r(i)]\}$$

Of course, the accuracy of sketch algorithms increases with the size of the matrix, since a larger matrix reduces the frequency of collisions of different elements on the same counter. In *CountMin*, setting $d = \lceil \ln \frac{1}{\delta} \rceil$ and $w = \lceil \frac{e}{\epsilon} \rceil$ ensures that f_i, in a stream with n elements, has error at most ϵn with probability of at least $1 - \delta$. The spatial complexity is $O(\frac{e}{\epsilon} \ln \frac{1}{\delta})$ while the time for update is $O(\ln \frac{1}{\delta})$.

More details about *CountMin* can be found in [12]. Here it is worth recalling that this algorithm, as all the sketch algorithms, has the important property that the sketch is a linear projection of the input. This means that the overall sketch of multiple streams can be computed by adding the sketches of the single streams. Thus is the main reason why we decided to adopt *CountMin*: in a distributed architecture, it would be prohibitive to transmit source data to a central processing node, while the mere transmission of sketch summaries allows overhead communications to be drastically reduced.

The computation of frequent itemsets can either be performed directly, or by exploiting the statistics of frequent items. The direct computation in a single pass is feasible only if the stream rate is moderate, due to the large number of candidate frequent itemsets. For example, in [6] a hybrid approach is used: first, a counter-based algorithm computes the candidate 2-itemsets, then a second pass is necessary to eliminate the *false* candidates, finally the *Apriori* property is exploited to find the frequent i-itemsets, for $i > 2$. The merit of hybrid approaches is that they try to combine the best of single-pass and multiple-pass algorithms [8], and can be particularly efficient in a distributed scenario. In our architecture, frequent items are computed on the fly with *CountMin*, and the results, stored in distributed data cachers, are used to compute frequent itemsets.

III. A HYBRID MULTI-DOMAIN ARCHITECTURE

The stream mining architecture presented in this paper aims at solving the problem of computing frequent items and frequent itemsets from distributed data streams, exploiting a hybrid single-pass/multiple-pass strategy. It is assumed that stream sources, though belonging to different domains, are homogenous, so that it is useful to extract knowledge from their union. Typical cases are the analysis of the traffic experienced by several routers of a wide area network, or the analysis of client requests forwarded to multiple web servers. Miner nodes are located close to the streams, so that data transmitted between different domains only consists of models (sketches), not raw data.

The architecture, depicted in Figure 2, includes the following components:

- **Data Streams (DS)**, located in different domains.

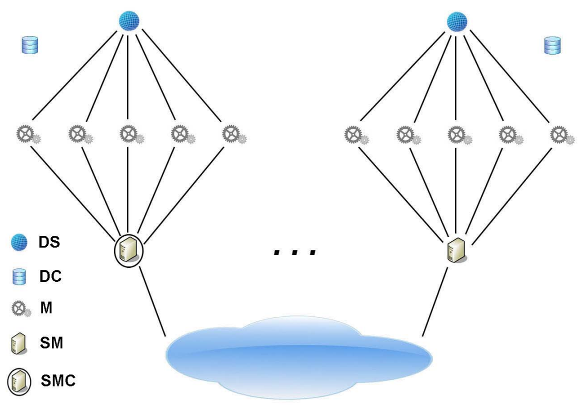


Figure 2. **Distributed architecture for data stream mining.**

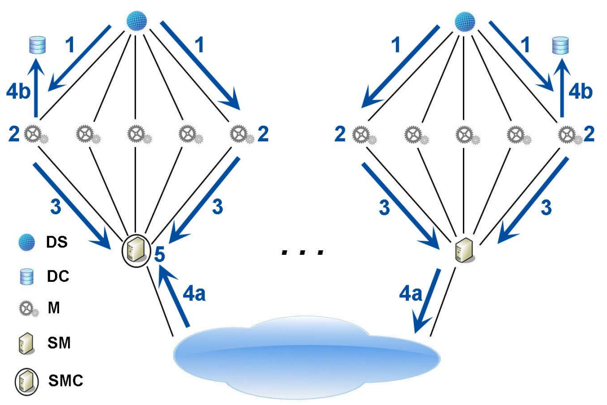


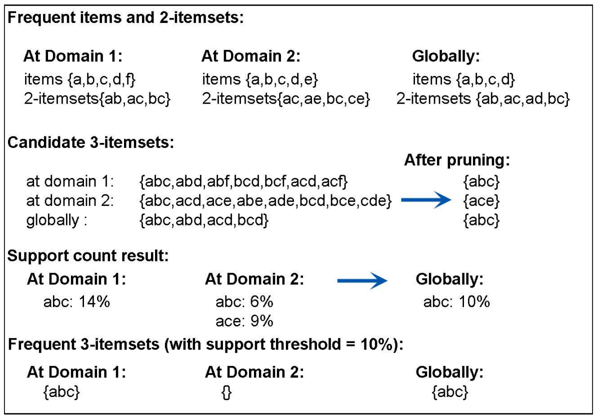
Figure 3. **Schema of the algorithm for mining frequent items.**

- **Miners (M)**. They are placed close to the respective Data Streams, and perform two basic mining tasks: the computation of sketches for the discovery of frequent items, and the computation of the support count of candidate frequent itemsets. If a single Miner is unable to keep the pace of the local DS, the stream items can be partitioned and forwarded to a set of Miners, which operate in parallel. Each Miner computes the sketch only for the data it receives, and then forwards the results to the local Stream Manager. Parallel Miners can be associated to the nodes of a cluster or to the cores of a manycore machine.
- **Stream Managers (SM):** in each domain, the Stream Manager collects the sketches computed by local miners, and derives the sketch for the local DS. Moreover, each SM cooperates with the Stream Manager Coordinator to compute global statistics, valid for the union of all the Data Streams.
- **Stream Managers Coordinator (SMC):** this node collects mining models from different domains and computes overall statistics regarding frequent items and frequent itemsets. The SMC can coincide with one of the Stream Managers, and can be chosen with an election algorithm. In the figure, the SM of the domain on the left also takes the role of SMC.
- **Data Cachers (DC)** are essential to enable the hybrid strategy. Each Data Cacher stores the statistics about frequent items discovered in the local domain. These results are then re-used by Miners to discover frequent itemsets composed of increasing numbers of items.

The algorithm for the computation of frequent items, outlined in Figure 3, is performed continuously, for every new block of data that is generated by the data streams. A *block* is defined here as the set of transactions that are generated in a time interval P. If the generation rate is too fast to be sustained by a single Miner, a filter is used to partition the block into as many mini-blocks as the number of available miners (step 1 in the figure). Each Miner computes the sketch related to the received mini-block (step 2) and transmits it to the SM (step 3), which overlaps the sketches, thanks to the linearity property of sketch algorithms, and extracts the frequent items for the local domain. Then, two concurrent operations are executed: every SM sends the local sketch to the SMC (step 4a), and the Miners send the most recent blocks of transactions to the local Data Cacher (step 4b). The last operation is necessary to later compute the frequent itemsets. At step 5, the SMC aggregates the sketches received by SMs and identifies the items that are frequent for the union of data streams. Frequent items are computed for a *window* containing the most recent W blocks. This can be done easily thanks to the linearity of the sketch algorithm: at the arrival of a new block, the sketch of this block is added to the current sketch of the window, while the sketch of the least recent block is subtracted. The window-based approach is common because most interesting results are generally related to recent data [13].

Sketch-based algorithms are only capable of computing *frequent items*. To discover *frequent itemsets*, it is necessary to perform multiple passes on the data. Candidate *k*-itemsets are constructed starting from frequent *(k–1)*-itemsets. More specifically, at the first step candidate 2-itemsets are all the possible pairs of frequent items: Miners must compute the support for these pairs to determine which of them are frequent. In the following steps, a candidate *k*-itemset is obtained by adding any frequent item to the frequent *(k–1)*-itemsets. Thanks to the Apriori property, candidates can be pruned by checking if all the *k–1* subsets are frequent: a *k*-itemset can be frequent only if all the subsets are frequent.

The approach allows us to compute both itemsets that are frequent for a single domain and those that are frequent for the union of distributed streams. Figure 4 shows an example of how frequent 3-itemsets are computed. The top part of the figure reports items and 2-itemsets that are frequent for the two considered domains and for the whole system. The candidate 3-itemsets, computed by the two SMs and by the SMC, are then reported, before and after the pruning based on the Apriori property. In the bottom part, the figure reports the support counts computed for the two domains and for the whole system. Finally, the SMs check which candidates exceed the specified threshold (in this case, set to 10%): notice that the {abc} itemset is frequent globally though it is locally frequent in only one of the two domains. In general, it can happen that an itemset occurs frequently for a single domain and infrequently globally, or vice versa: therefore, it is necessary to separately perform the two kinds of computations.


Frequent items and 2-itemsets:

At Domain 1:	**At Domain 2:**	**Globally:**
items {a,b,c,d,f}	items {a,b,c,d,e}	items {a,b,c,d}
2-itemsets{ab,ac,bc}	2-itemsets{ac,ae,bc,ce}	2-itemsets {ab,ac,ad,bc}

Candidate 3-itemsets:

		After pruning:
at domain 1:	{abc,abd,abf,bcd,bcf,acd,acf}	{abc}
at domain 2:	{abc,acd,ace,abe,ade,bcd,bce,cde} →	{ace}
globally :	{abc,abd,acd,bcd}	{abc}

Support count result:

At Domain 1:	**At Domain 2:** →	**Globally:**
abc: 14%	abc: 6%	abc: 10%
	ace: 9%	

Frequent 3-itemsets (with support threshold = 10%):

At Domain 1:	**At Domain 2:**	**Globally:**
{abc}	{}	{abc}

Figure 4. **Example of the computation of frequent 3-itemsets.**

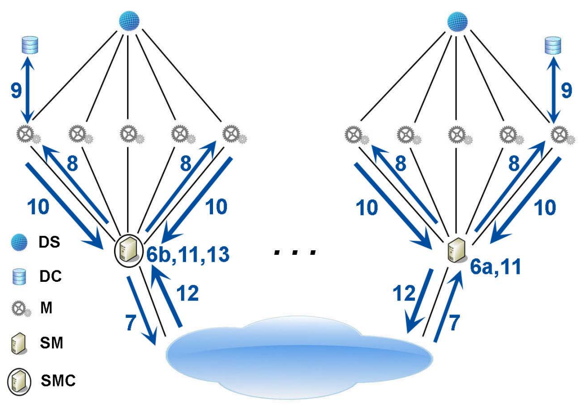


Figure 5. **Schema of the algorithm for mining frequent itemsets.**

The schema of the algorithm for mining frequent itemsets is illustrated in Figure 5. It assumes that the steps indicated in Figure 3 have already been performed. At step 6, every SM builds the candidate k-itemsets for the local domain (6a), and the SMC also builds the global candidate k-itemsets (6b). The SMC sends the global candidates to the SMs for the computation of their support at the different domains (step 7). The SMs send both local and global candidates to the Miners (step 8), which turn to the Data Cacher to retrieve the transactions included in the current window (step 9)[1], compute the support count for all the candidates, and transmit the results to the local SM (step 10). The SM aggregates the support counts received by Miners and selects the k-itemsets that are frequent in the local domain (step 11). Analogously, the SMs send the SMC the support counts of the global candidates (step 12), and the SMC computes the itemsets that are frequent over the whole system (step 13). The algorithm restarts from step 6 to find frequent itemsets with increasing numbers of items. The cycle stops either when the maximum allowed size of itemsets is reached or when no frequent itemset was found in the last iteration.

IV. Prototype and Performance Evaluation

The architecture described in the previous section was implemented starting from the *Mining@Home* system. *Mining@Home*, a Java-based framework partly inspired by the Public Computing paradigm, was adopted to perform several classes of data mining computations, among which the analysis of astronomical data to search for gravitational waves [14], and the discovery of closed frequent itemsets with parallel algorithms [15]. The main features of the stream mining prototype inherited from *Mining@Home*, are the *pull* approach (Miners are assigned jobs on the basis of their availability) and the adoption of Data Cachers to store reusable data. Moreover, some important modifications were necessary to adapt the framework to the stream mining scenario: for example, the selection of the Miners that are the most appropriate to perform the mining tasks is subject to vicinity constraints, because in a streaming environment it is very important that the analysis of data be performed close to the data source. Another notable modification is the adoption of the hybrid approach for the single-pass computation of frequent items and the multi-pass computation of frequent itemsets.

Experiments were performed on the ICAR-CNR Grid. We used two clusters, connected by a router to test a scenario with two domains and two data streams. The first cluster has eight Cpu Intel Xeon E5520 nodes with four 2.27 GHz processors and 24 GB Ram; the second cluster has eight Intel Itanium nodes with two 1.5 GHz CPU and 4 GB Ram. The Miners and the Stream Managers were installed on the nodes of the clusters while the Data Streams and the Data Cachers were put on different nodes, external to the clusters. All the nodes run Linux, and the software components are written in Java.

To assess the prototype, we used the transactional datasets published by the *FIMI Reposity* [16]. Some of these datasets are originated by data streams, so they are appropriate for our analysis. In particular:

[1]Miners may have the ability to store some transactions in their own memory. In this case, they only ask the Data Cacher those transactions that could not be stored locally.

- The *"kosarak"* dataset contains a list of *click-streams* generated by users of an online portal. The analysis of user visits can be useful to identify the most popular sections of the portal, the preferences and requirements of users, etc.;
- the *"webDocs"* dataset is generated from a set of Web pages. Each page, after the application of a filtering algorithm, is represented with a set of significant words included in it. The analysis of most frequent words, or sets of words, can be useful to devise caching policies, indexing techniques, etc.

Basic information about the two datasets is summarized below:

Dataset	**MB**	**No. of tuples**	**No. of distinct items**	**Size of tuples (no. of items)**		
				min	**med**	**max**
kosarak	30.5	990002	41270	1	8	2498
webDocs	1413	1692082	5267656	1	177	71472

The parameters used to assess the prototype are listed below:

- P: the time interval that delimits a block of data. This interval determines the average number of transactions generated within a block, denoted as N_t, and the average size of a block in bytes, B;
- N_M: the number of available miners per domain, assuming that this number is the same for the two domains;
- F_{CPU}: the fraction of CPU time dedicated by miners to mining jobs. In the experiments this parameter is tuned using the program *"cpulimit"* [2].
- S: the support threshold used to determine frequent items and itemsets;
- W: the number of blocks contained in the sliding window;
- C_M: the capacity of the miner buffer, expressed in bytes. In the experiments, the buffer size is set so as to contain the most recent block of data.
- ϵ and δ, the accuracy parameters of the sketch algorithm, which are both set to 0.01.
- the maximum size of candidate itemsets, set to 5 for Kosarak and to 8 for Webdocs.

The main performance index assessed during the experiments is the *average execution time*, i.e., the time necessary to compute frequent items and frequent itemsets at the arrival of a new block of data. If this value is not longer than the time interval P, it means that the system is able to keep the pace with data production.

A. *Experiments with the dataset* Kosarak

The experiments were executed assuming a time period P equal to $15s$. The generation rates were compatible with the Web sites of Wikipedia, Microsoft and Ebay, as estimated using the Web site *http://www.webtraffic24.com*.

[2]http://cpulimit.sourceforge.net.

These rates correspond, respectively, to values of N_t equal to about 20,000, 15,000 and 10,000 transactions per block. The generation rates were equally partitioned between the two domains. Notice that these are very high generation rates, and allowed the prototype to be tested in challenging conditions. 30% of the CPU time was dedicated by miners to the mining jobs, the frequency threshold S was set to 0.02, and the window size was set to 5 blocks.

Figure 6 reports the average execution time experienced for the computation of frequent items exclusively (I), and for the computation of both frequent items and itemsets (I+IS), vs. the number of miners per domain N_M. The figure shows that the processing time decreases as the number of miners increases, which confirms that the architecture is scalable. Scalability is ensured by two main factors: the linearity property of the sketch algorithm, and the placement of data cachers close to the miners. The dashed line, corresponding to an execution time equal to the time period P (15 seconds), is used to check if the system is stable: when the index is lower than the dashed line, the system is able to keep pace with the generation of stream data. This condition is always verified when the system is only asked to compute frequent items. On the other hand, the computation of frequent itemsets is much more time consuming, and results show that a single miner per domain is not sufficient: depending on the generation rate, two, three or four miners are needed to keep the processing time below the period length P.

Figure 7 shows the average execution time measured when setting the value of N_t to 15,000 and varying the support threshold S. This parameter does not influence the time to execute the frequent items algorithm, which is executed in a single pass: therefore the figure reports the results regarding the combined computation of frequent items and frequent itemsets (I+IS). As expected, a lower value of the threshold leads to an increase of the processing time, since the number of frequent itemsets, computed at each step of the algorithm (see Figure 5), is larger.

B. *Experiments with the dataset* Webdocs

A second set of experiments was performed taking the dataset *Webdocs* as input of the data stream. As the dataset contains representative words of Web pages filtered by a search engine, the data rate was set to typical values registered by servers of Google and Altavista, again using the site *http://www.webtraffic24.com* to do the estimation. The considered values for N_t were 500, 1500 and 3000 transactions, with the time period P set to 15 seconds. A single transaction contains on average many more items than the Kosarak dataset, so the size of a block in bytes is larger: for example, with $N_t = 1500$, the value of B is about 750 KBytes, while it was about 150 Kbytes with Kosarak. As for the previous set of tests, the percentage of the CPU time dedicated by miners to the mining jobs was set to 30%, the

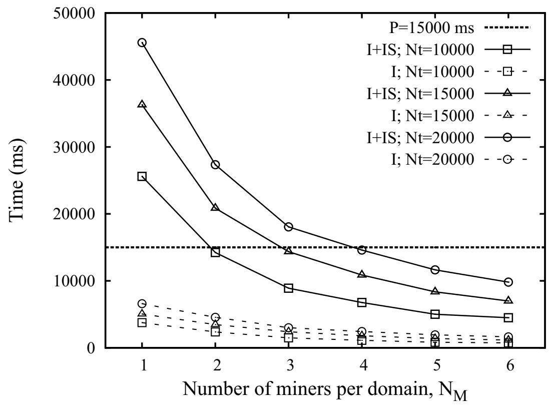


Figure 6. Analysis of *Kosarak*: average execution time for the computation of frequent items (I) and itemsets (IS), vs. the number of miners per domain, for different values of the number of transactions per block, N_t.

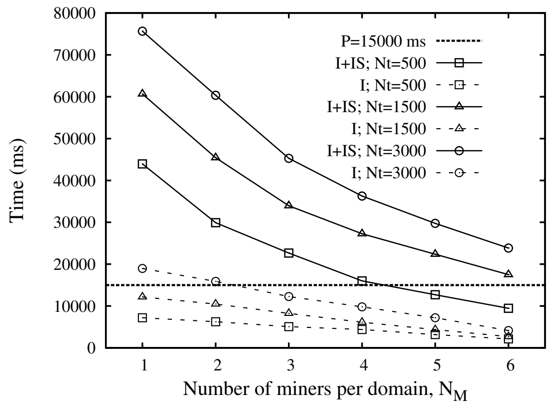


Figure 8. Analysis of *Webdocs*: average execution time for the computation of frequent items (I) and itemsets (IS), vs. the number of miners per domain, for different values of the number of transactions per block, N_t.

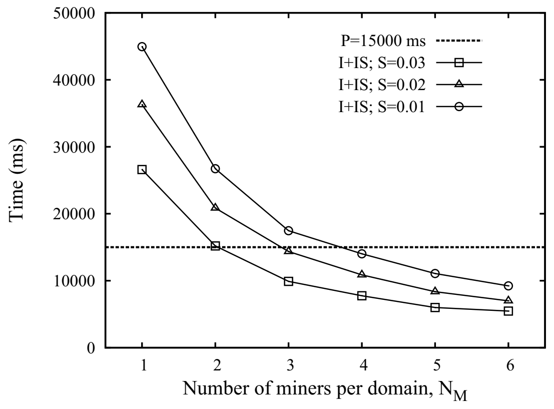


Figure 7. Analysis of *Kosarak*: average execution time for the computation of frequent items and itemsets vs. the number of miners per domain, for different values of the threshold S. The value of N_t is set to 15,000.

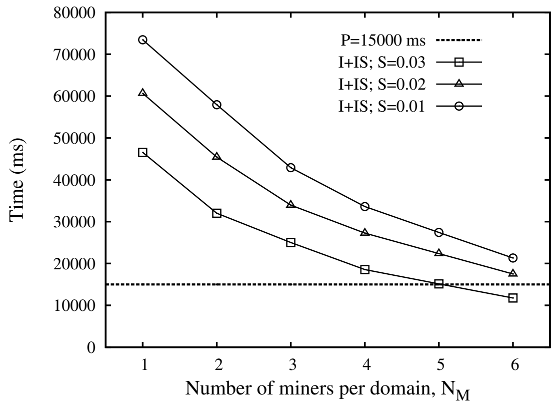


Figure 9. Analysis of *Webdocs*: average execution time for the computation of frequent items and itemsets vs. the number of miners per domain, for different values of the threshold S. The value of N_t is set to 1500.

frequency threshold S to 0.02 (except when this parameter was varied) and the window size to 5 blocks.

Figure 8 shows the average time needed to compute frequent items and itemsets after the arrival of a new block. The architecture has no problems computing frequent items before the arrival of the next block but, owing to the higher complexity of the dataset (in particular, the larger number of items per transaction), the computation of frequent itemsets is more challenging. To perform this task, six miners per domain are sufficient if the data rate is equal to 500 transactions per time period, but results show that more miners are needed in the case of higher generation rates. It should be considered that a centralized architecture would have few chances to keep pace with data, which means that the computation of frequent itemsets would have to be done completely offline, while the architecture proposed here can achieve the objective by using an appropriate degree of parallelism.

Figure 9 focuses on the case in which the transaction rate is set to 1500 transactions per time period, and illustrates the impact of the threshold value S. Reported results show that the system is stable (i.e., the processing time is shorter than the time period) only when the threshold is as high as 0.03 and at least 5 miners per domain are available.

V. Related Work

The analysis of data streams has recently attracted a lot of attention owing to the wide range of applications for which it can be extremely useful. Important challenges arise from the necessity of performing most computation with a single pass on stream data, because of limitations in time and memory space. Stream mining algorithms deal with problems as diverse as clustering and classification of data streams, change detection, stream cube analysis, indexing, forecasting, etc [17].

For many important application domains previously mentioned in this paper, a major need is to identify frequent patterns in data streams, either single frequent elements or frequent sets of items in transactional databases. A rich survey of algorithms for discovering frequent items is provided by Cormode and Hadjieleftheriou [7]. In their paper, discussion focuses on the two main classes of algorithms for finding frequent items. Counter-based algorithms have their foundation on some techniques proposed in the early 80s to solve the *Majority* problem [18], i.e., the problem of finding a majority element in a stream, using a single counter. Variants of this algorithm were devised, sometimes decades later, to discover items whose frequencies exceed any given threshold. LossyCounting is perhaps the most popular algorithm of this type [11]. The second class of algorithms compute a *sketch*, i.e., a linear projection of the input, and provide an approximated estimation of item frequencies using limited computing and memory resources. Popular algorithms of this kind are CountSketch [19] and CountMin [12], and the latter is adopted in this paper. Advantages and limitations of sketch algorithms are discussed in [20]. Important advantages are the notable space efficiency (required space is logarithmic in the number of distinct items), the possibility of naturally dealing with negative updates and item deletions, and the linear property, which allows sketches of multiple streams to be computed by overlapping the sketches of single streams. The main limitation is the underlying assumption that the domain size of the data stream is large, but this is true in many significant domains.

Even if modern single-pass algorithms are extremely sophisticated and powerful, multi-pass algorithms are still necessary either when the stream rate is too rapid, or when the problem is inherently related to the execution of multiple passes, which is the case, for example, of the frequent itemsets problem. Single-pass algorithms can be forced to check the frequency of 2- or 3-itemsets, but this approach cannot be generalized easily, as the number of candidate k-itemsets is combinatorial, and it can become very large when increasing the value of k [6]. Therefore, a very promising avenue could be to devise hybrid approaches, which try to combine the best of single- and multiple-pass algorithms. A strategy of this kind, discussed in [8], is adopted in the mining architecture presented in this paper.

The analysis of streams is even more challenging when data is produced by different sources spread in a distributed environment. A thorough discussion of the approaches currently used to mine multiple data streams can be found in [21]. The paper distinguishes between the *centralized* model, under which streams are directed to a central location before they are mined, and the *distributed* model, in which distributed computing nodes perform part of the computation close to the data, and send to a central site only the models, not the data. Of course, the distributed approach has notable advantages in terms of degree of parallelism and scalability. An interesting approach for the continuous tracking of complex queries over collections of distributed streams is presented in [3]. To reduce the communication overhead, the adopted strategy combines two technical solutions: (i) remote sites only communicate to the coordinator concise summary information on local streams (in the form of sketches); (ii) even such communications are avoided when the behavior of local streams remains reasonably stable, or predictable: updates of sketches are only transmitted when a certain amount of change is observed locally. The success of this strategy depends on the level of approximation on the results that is tolerated. A similar approach is adopted in [22]: here stream data is sent to the central processor after being filtered at remote data sources. The filters adapt to changing conditions to minimize stream rates while guaranteeing that the central processor still receives the updates necessary to provide answers of adequate precision.

VI. Conclusions

In recent years, the progress in digital data production and pervasive computing technology have made it possible to produce and store large streams of data. Data mining techniques became vital to analyze such large and continuous streams of data for detecting regularities and outlier values in them. In particular, when data production is massive and/or distributed, decentralized architectures and algorithms are needed for its analysis.

The distributed stream mining system presented in this paper is a contribution in the field and it aims at solving the problem of computing frequent items and frequent itemsets from distributed data streams by exploiting a hybrid single-pass/multiple-pass strategy. We assumed that stream sources, though belonging to different domains, are homogenous, so that it is useful to extract knowledge from their union. Beyond presenting the system architecture, we described a prototype that implements it and discussed a set of experiments performed in a real Grid environment. The experimental results confirm that the approach is scalable and can manage large data production by using an appropriate number of miners in the distributed architecture.

References

[1] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy, "Mining data streams: A review," *ACM SIGMOD Record*, vol. Vol. 34, no. 1, 2005.

[2] C. C. Aggarwal, *Data Streams: models and algorithms*. Springer, 2007.

[3] G. Cormode and M. Garofalakis, "Approximate continuous querying over distributed streams," *ACM Transactions on Database Systems*, vol. Vol. 33, no. 2, 2008.

[4] G. Cormode, S. Muthukrishnan, K. Yi, and Q. Zhang, "Optimal sampling from distributed streams," *ACM Principles of Database Systems (PODS)*, 2010.

[5] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo, "Fast discovery of association rules," pp. 307–328, 1996.

[6] R. Jin and G. Agrawal, "An algorithm for in-core frequent itemset mining on streaming data," in *5th IEEE International Conference on Data Mining ICDM*, Houston, Texas, USA, 2005, pp. 210–217.

[7] G. Cormode and M. Hadjieleftheriou, "Finding the frequent items in streams of data," *Communications of the ACM*, vol. 52, no. 10, pp. 97–105, 2009.

[8] A. Wright, "Data streaming 2.0," *Communications of ACM (CACM)*, vol. Vol. 53, no. 4, 2010.

[9] G. Cormode and M. Hadjieleftheriou, "Finding frequent items in data streams," in *International Conference on Very Large Data Bases*, 2008.

[10] A. Metwally, D. Agrawal, and A. Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *International Conference on Database Theory*, 2005.

[11] G. Manku and R. Motwani, "Approximate frequency counts over data streams," in *International Conference on Very Large Data Bases*, 2002.

[12] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *J. Algorithms*, vol. Vol. 55, 2005.

[13] M. Datar, A. Gionis, P. Indyk, and R. Motwani, "Maintaining stream statistics over sliding windows," *SIAM Journal on Computing (SIAMCOMP)*, vol. Vol. 31, no. 6, 2002.

[14] C. Mastroianni, P. Cozza, D. Talia, I. Kelley, and I. Taylor, "A scalable super-peer approach for public scientific computation," *Future Generation Computer Systems*, vol. 25, no. 3, pp. 213–223, March 2009.

[15] C. Lucchese, C. Mastroianni, S. Orlando, and D. Talia, "Mining@home: toward a public resource computing framework for distributed data mining," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 5, pp. 658–682, April 2009.

[16] "Frequent itemset mining dataset repository," available at http://fimi.cs.helsinki.fi.

[17] C. Aggarwal, "An introduction to data streams," in *Data Streams: Models and Algorithms*, C. Aggarwal, Ed. Springer, 2007, pp. 1–8.

[18] M. Fischer and S. Salzburg, "Finding a majority among n votes: solution to problem 81-5," *J. Algorithms*, vol. 3, no. 4, pp. 376–379, 1982.

[19] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, 2002.

[20] C. C. Aggarwal and P. S. Yu, "A survey of synopsis construction in data streams," in *Data Streams: Models and Algorithms*, C. Aggarwal, Ed. Springer, 2007, pp. 169–207.

[21] A. G. Srinivasan Parthasarathy and M. E. Otey, "A survey of distributed mining of data streams," in *Data Streams: Models and Algorithms*, C. Aggarwal, Ed. Springer, 2007, pp. 289–307.

[22] C. Olston, J. Jiang, and J. Widom, "Adaptive filters for continuous queries over distributed data streams," in *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, San Diego, California, 2003.