Chapter 17

Mining distributed data streams on Content Delivery Networks

Eugenio Cesario, Carlo Mastroianni and Domenico Talia

17.1 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 off-line. 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. Important application fields for stream mining are as diverse as financial applications, network monitoring, security problems, telecommunication networks, Web applications, Content Delivery Networks, sensor networks, analysis of atmospheric data, etc.

The mining data stream process becomes more difficult when streams are distributed, because mining models must be derived not only for the data of a single stream, but for the integration of multiple and heterogeneous data streams. This scenario can occur in all the application domains mentioned before and specifically in a Content Delivery Network. In this context, 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 among the servers. The analysis of user requests, for example to discover frequent patterns, must be performed with the inspection of the data streams detected by different servers. The discovery of popular items can suggest the data and the Web pages that are more interesting (i.e., frequently requested) for the users: such data can be pre-fetched in the Web cache of servers, in order to serve future demands more efficiently. In order to improve the efficiency and scalability of the whole process, the Content Delivery Network can be hosted on and exploit the facilities of a Cloud infrastructure.

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, 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. In this chapter we elaborate a distributed architecture, firstly introduced in [4], for mining data streams generated from multiple and heterogeneous data sources, with specific focus on the case of Content Delivery Networks.

More in detail, the architecture exploits 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, 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;
- the approach is hybrid, meaning that frequent items are calculated online, with a single pass, while frequent itemsets are calculated as a background activity by a further 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.

The major advantages of the presented 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. Such behavior can be naturally obtained if the miners are running on Cloud servers, because the number of active machines is dynamically adapted and scaled to the computational load.

Beyond presenting the architecture, we describe an implemented prototype and discuss a set of experiments performed in a distributed environment composed of two domains each one handling a data stream.

17.2 Background/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 [1].

For many important application domains, 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 [4]. Some of these algorithms, for example CountSketch [5], 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. Advantages and limitations of sketch algorithms are discussed in [2]. 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, however this assumption holds 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 [5]. A very promising avenue could be to devise hybrid approaches, which try to combine the best of single- and multiple-pass algorithms [22] . A strategy of this kind 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, as happens in a Content Delivery

Network. 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 [7]. 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 [19]: 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. In [17], the problem of finding frequent items in the union of multiple distributed streams is tackled by setting a hierarchical communication topology in which streams are the leaves, a central processor is the root, and intermediate nodes can compress data flowing from the leaves to the root. The amount of compression is dynamically adapted to make the tolerated error (difference from estimation and actual frequency of items) follow a precision gradient:

the error must be very low at nodes close to the sources, but it can gradually increase as the communication hierarchy is climbed. The objective of this strategy is to minimize load on the central node while providing acceptable error guarantees on answers.

Stormy [15] is a distributed multi-tenant streaming service designed to run on Cloud infrastructures. Like traditional data stream processing engines (SPEs), it executes continuous queries against ongoing streams of incoming data and eventually forwards the results to a designated target. To achieve good scalability, the system exploits distributed hash tables (DHT) for an efficient distribution of the queries across all nodes. Moreover, it guarantees good level of fault tolerance by replicating query executions on several nodes, avoiding in this way a single point-of-failure. The system combines standard techniques of Cloud Computing with stream processing to guarantee elasticity, scalability, and fault tolerance.

The *Prism* system [9], the Portal Infrastructure for Streaming Media, is a Content Delivery Network architecture for distributing, storing, and delivering high quality streaming media over the Prism IP network infrastructure. A first service developed on this architecture was the Prism-based stored-TV (STV) service, aimed at allowing users to select content based on the program's name. It is composed of three main modules: *live sources*, *portals* and *clients*. *Live sources* receive content from a content provider, encode and packetize it, and then stream it into the Prism IP network. *Portals* receive multimedia content from live sources and other portals, and transmit it to Prism clients. Portals can store and archive live content, thus allowing it to be viewed on demand, and provide functions such as fast-forward and rewind.. *Clients* receive content from a portal and display it to end users. They are connected to the backbone using broadband access.

17.3 A Hybrid Multi-Domain Architecture

This section presents the stream mining architecture that aims at solving the problem of computing frequent items and frequent itemsets from distributed data streams, 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. 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 of a Content Delivery Network. 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 includes the following components, depicted in Figure 17.1:

- **Data Streams (DS)**, located in different domains.
- **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 a high speed computer network, 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 Figure 17.1, the SM of the domain on the left also takes the role of SMC.
- **Data Cachers (DC)** are essential to enable the hybrid strategy and the computation of frequent itemsets, when this is needed. 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, also outlined in Figure 17.1, is performed continuously, for each new block of data generated by the data streams. A block is defined here as the set of transactions that are generated in a time interval P. The algorithm includes the following steps, also shown in the figure:

1. a filter is used to partition the block into as many mini-blocks as the number of available Miners;
2. each Miner computes the sketch related to the received mini-block;
3. the Miner transmits the sketch to the SM, which overlaps the sketches, thanks to the linearity property of sketch algorithms, and extracts the frequent items for the local domain;

4. 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 only needed when frequent itemsets are to be computed, otherwise it can be skipped;
5. the SMC aggregates the sketches received by SMs and identifies the items that are frequent for the union of data streams.

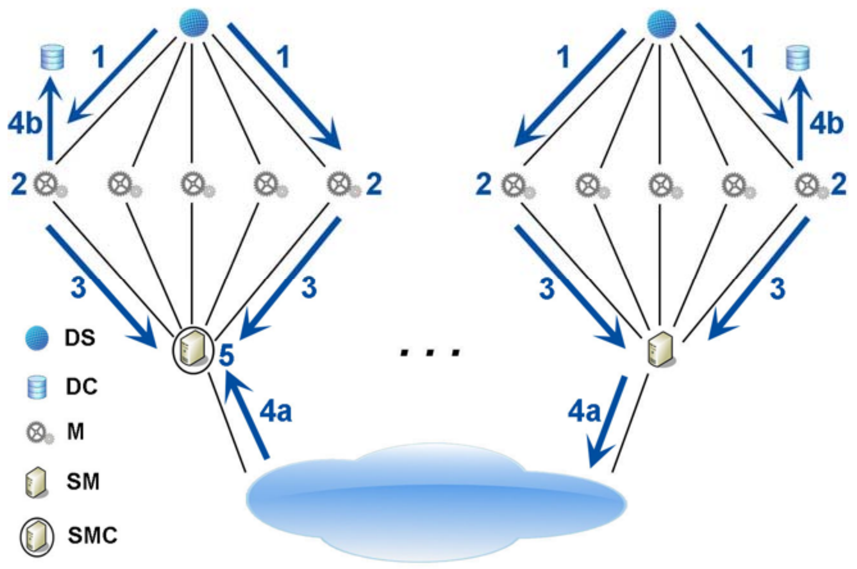
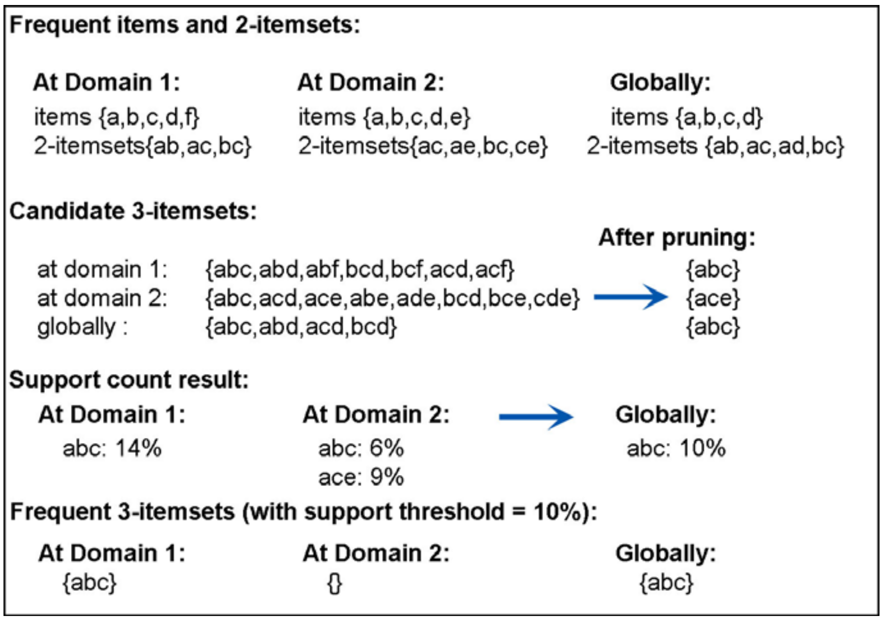


Figure 17.1: Architecture and schema of the algorithm for mining frequent items.

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 [10].

Sketch-based algorithms are only capable of computing frequent items. To discover frequent itemsets, it is necessary to perform multiple passes on 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 17.2 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 17.2: Example of the computation of frequent 3-itemsets.

The schema of the algorithm for mining frequent itemsets is illustrated in Figure 17.3, which assumes that the steps indicated in Figure 17.1 have already been performed.

The successive steps are:

6. each SM builds the candidate k-itemsets for the local domain (6a), and the SMC also builds the global candidate k-itemsets (6b);
7. the SMC sends the global candidates to the SMs for the computation of their support at the different domains;
8. SMs send both local and global candidates to the Miners;
9. Miners turn to the Data Cacher to retrieve the transactions included in the current window (this operation is performed only at the first iteration of the algorithm). Then, the Miners compute the support count for all the candidates, using the window-aware technique presented in [11];
10. Miners transmit the results to the local SM;

11. the SM aggregates the support counts received by Miners and selects the k-itemsets that are frequent in the local domain;
12. analogously, the SMs send the SMC the support counts of the global candidates;
13. the SMC computes the itemsets that are frequent over the whole system. At this point, 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.

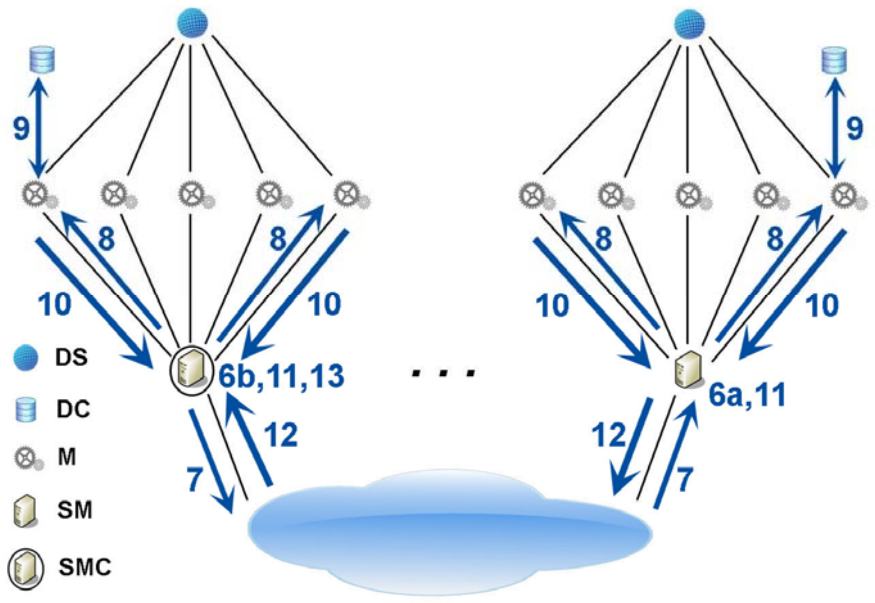


Figure 17.3: Schema of the algorithm for mining frequent itemsets.

17.4 A Prototype for Stream Mining in a CDN

The architecture described in the previous section was implemented starting from *Mining@Home*, a Java-based framework partly inspired by the Public Computing paradigm, which was adopted for several classes of data mining computations, among which the analysis of astronomical data to search for gravitational waves [18], and the discovery of closed frequent itemsets with parallel algorithms [16]. 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 is 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 distribution infrastructure. We used two networks, connected by a router to test a scenario with two domains and two data streams. Each network has a cluster: the first cluster has twelve Cpu Intel Xeon E5520 nodes with four 2.27 GHz processors and 24 GB RAM; the second cluster has twelve 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 Sources and the Data Cachers were put on different nodes, external to the clusters. The average inter-domain transfer rate measured in the experiments was 197*KB/s*, while the average intra-domain transfer rates were 918*KB/s* and 942*KB/s* in the two networks. All the nodes run Linux, and the software components are written in Java.

To assess the prototype, we used “webDocs”, a stream dataset published by the *FIMI Repository* [11]. The dataset is generated from a set of distributed Web hosts belonging to a Content Delivery Network. Each Web 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. Some basic information about the dataset is summarized below:

Dataset	**MB**	**No. of tuples**	**No. of distinct items**	**Size of tuples (no. of items)**		
				Min	Med	Max
webDocs	1413	1692082	5267656	1	177	71472

The parameters used to assess the prototype are listed below:

- N_t is the average number of transactions generated within a block.
- N_M is the number of available miners per domain, assuming that this number is the same for the two domains;

As the webDocs 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 interval P set to 15 seconds. It is assumed that the cache of a miner can contain one complete block of data.

The support threshold used to determine frequent items and itemsets is set to 0.02. The size of the sliding window, i.e., the number of consecutive blocks of data on which computation is performed, is set to 5 blocks. The accuracy parameters of the sketch algorithm, ϵ and δ [11], are both set to 0.01. Another peculiar parameter of the sketch algorithm, maximum size of candidate itemsets, is set to 8.

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.

Figures 17.4 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 execution time is calculated from the time at which a block of data is ready to be delivered from the Data Sources to the time at which the Stream Miner Controller terminates the computation. The dashed line corresponds to the time period P, and it is shown to easily check in which cases the system is stable. Results, for three different values of N_t, show that a single miner per domain is not sufficient: depending on the generation rate, at least 4, 6 or 8 miners are needed to keep the processing time below the period length P. In general, the processing time decreases as the number of miners increases, which is a sign of the good scalability of the architecture. Scalable behavior is ensured by two main factors: the linearity property of the sketch algorithm, and the placement of Data Cachers close to the miners.

In this evaluation the case of N_M=1 corresponds to a non-parallel architecture in which the mining computation is performed on a single node. Therefore the results in Figure 17.4 can also be seen as a comparison between a parallel and a sequential solution. It is worth noting that in this scenario any centralized architecture would have few chances to keep pace with data, which means that the computation of frequent itemsets would have to be done offline, while the architecture presented here can achieve the goal of online processing by using an appropriate degree of parallelism.

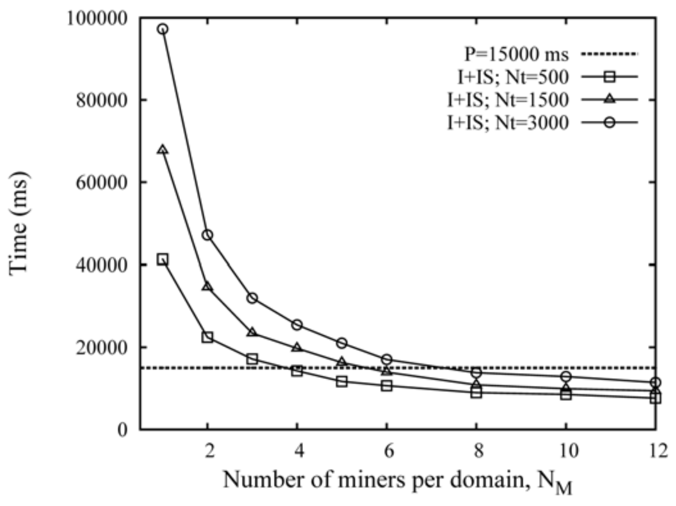


Fig. 17.4. Analysis of webDocs streams: average execution time for the computation of frequent items and itemsets (I+IS), vs. the number of miners per domain, for different values of the number of transactions per block, Nt.

To better assess the system behavior, it is useful to analyze the data traffic involved in the computation. Figure 17.5 shows the amount of data transmitted over the network at the generation of a new block of data, for different numbers of miners per domain. In these experiments, N_t was set to 15000. The first three groups of bars show the overall amount of data transferred between nodes of type A to nodes of type B in a single domain, denoted as A→B. For example, DS→M is the amount of data transmitted by the Data Source to the Miners of a single domain at every time period. The values of DS→M and M→DC are equal since each Miner sends to the local DC the data received from the DS. The fourth group reports DT_{Domain}, the overall amount of data transferred within a single domain, computed as the sum of the contributions shown in the first three groups (the contribution of the first group is considered twice). The contribution SM→SMC is the amount of data transferred between the Stream Managers and the

Stream Manager Controller. Finally, the last group of bars reports the amount of data transferred over the whole network, DT_{Net}. This is computed as the term DT_{Domain} times the number of domains (in this case 2) plus the term SM→SMC. It is interesting to notice that the contribution DC→M decreases when the number of miners per domain increases, and becomes null when N_M is equal or greater than 5. This can be explained by considering that each miner must compute the frequency of items and itemsets over a window of 5 time periods, and possesses a cache that can contain one complete block of data. If there is only one miner per domain, this miner can store one block and must retrieve the remaining four blocks from the local Data Cacher. As the number of miners increases, each Miner needs to request less data from the Data Cacher, and needs no data when N_M equals or exceeds the window size.

Conversely, the component M→SM slightly increases with the number of miners, because every miner sends the local SM a set of results (the sketch and the support counts of itemsets) having approximately the same size. However, as the contribution of DC→M has a larger weight than M→SM, the overall amount of data exchanged within a domain decreases as N_M increases from 0 to 5. For larger values of NM, DT_{Domain} starts to increase, because M→SM slightly increases and DC→M gives no contribution. A similar trend is observed for the values of DT_{Net}. Therefore, not only the presence of multiple miners allows the computation to be distributed, but also leads to a decrease of the overall amount of transferred data.

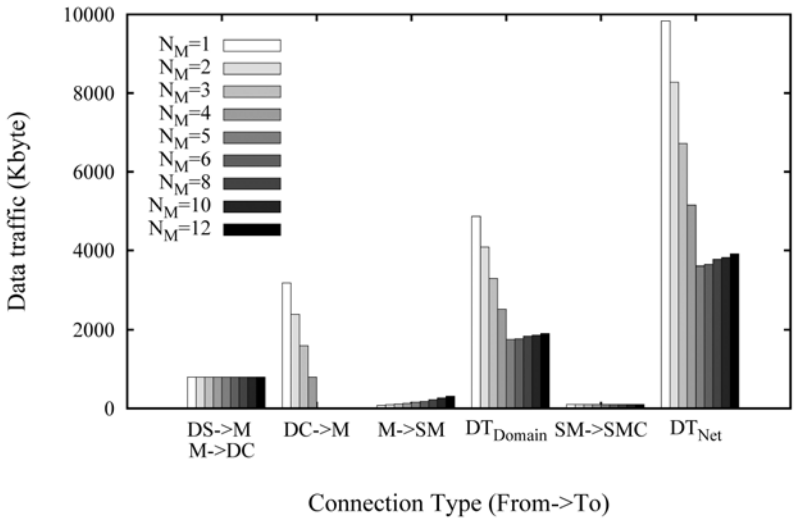


Fig. 17.5 Data traffic per each block of data generated by the Data Streams. In this experiments, Nt=1500.

A study has been performed in order to evaluate the efficiency of the approach. In particular, the efficiency analysis was performed in accordance with the study of parallel architectures presented in [13]. Specifically, we extracted the overall computation time T_C, i.e., the sum of the computation times measured on the different miners, and the overall overhead time T_O, defined as the sum of all the times spent in other activities, which practically coincide with the transfer times. The efficiency of the computation can be defined as the fraction of time that the miners actually devote to computation with respect to the sum of computation and overhead time: $E = T_C \backslash (T_C + T_O)$. The Figure 17.6 reports efficiency values obtained in our experiments. It is observed that efficiency increases as the number of miners increases up to 5, i.e., the window size. This effect is induced by the use of caching, as explained in [13]. Specifically, when N_M increases up to the window size, each miner needs to request less data from the Data Cacher, because more data can be stored in the local cache: this leads to a higher efficiency. For larger values of N_M, this effect does not hold anymore and the efficiency slightly decreases, being still very high. Moreover, it is noticed that the efficiency increases with the rate of data streams. This means that the distributed architecture is increasingly convenient when the problem size increases, which is a another sign of good scalability properties.

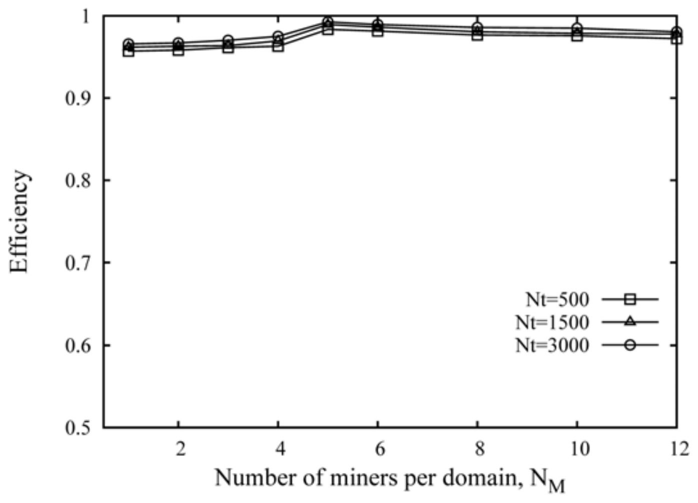


Fig. 17.6 Efficiency vs. the number of miners per domain, for different values of the number of transactions per block, Nt.

17.5 Visionary thoughts for practitioners

A walk through CDN’s evolution, discussed in [20], reveals a smooth transition from pre-evolution period to the current generation of CDNs. Two of the identified evolutionary avenues are: a more massive deployment of servers with improved caching techniques, and a major boost for delivering streaming rich media content: video, audio and associated data. The analysis of streaming data is essential to improve the efficiency and effectiveness of CDN architectures and the value of CDN services to the user.

Data mining algorithms can provide an efficient way to perform this analysis, but they must be adapted to the distributed and heterogeneous environment that is peculiar to modern CDN solutions. The architecture and prototype presented in this chapter are, to the best of our knowledge, one of the first attempts to devise a stream mining solution that offer the following characteristics, which can be particularly profitable in the context of Content Delivery Networks: parallel/distributed architecture, use of sketch algorithms, hybrid online/offline approach, and use of distributed data cachers. 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 [22].

The major advantages of the presented 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 when mining frequent itemsets is not required or when the stream rate is so slow that they can be computed on the fly.

17.6 Future Research Directions

The accurate analysis of stream data is becoming increasingly important as more and more companies are exporting not only their public Web pages but also their data and applications to the Cloud. The success of Cloud has a strong impact on the building and usage of Content Delivery Networks, and future research activities should investigate the advantages that can derive from the integration of Cloud and CDN. Indeed, the services offered by main CDN players, such as Akamai and Mirror Image, are priced out of reach for all but the largest enterprise customers. An alternative approach to content delivery is presented in [3] and consists of leveraging the infrastructure provided by 'Storage Cloud' providers, who offer internet accessible data storage and delivery at a much lower cost. The devised architecture, MetaCDN, exploits Cloud storage facilities and creates an integrated overlay network that provides a low cost, high performance CDN for content creators. In this or similar architectures the analysis of stream data coming from distributed and possibly heterogeneous Cloud facilities is of much interest.

On the other hand, Cloud resources can be used to help the analysis of distributed data streams. For example, modern commercial analytics tools such as Gomez, Keynote and Cedexis, produce large amounts of stream data that is used to drive Web platform analysis and optimize Web business performance. Cloud environments provide efficient computing infrastructures and offer effective support to the analysis of stream data and the implementation of data mining and knowledge discovery systems. Cloud infrastructures can be efficiently exploited to archive, analyze and mine large distributed data sets. An interesting approach is described in [14]. The authors present a combined stream processing system that, as the input stream rate varies, adaptively balances workload between a dedicated local stream processor and a Cloud stream processor. The approach only utilizes Cloud machines when the local stream processor becomes overloaded.

17.7 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 or outlier values in them. In particular, when data production is massive and/or distributed, decentralized architectures and algorithms are needed for its analysis. Content Delivery Networks have recently experienced a gradual but continuous evolution from the publication of static and semi-static content to the provisioning of multimedia data streams. Therefore, CDNs are a privileged scenario for the application of distributed stream mining techniques.

The architecture 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. Beyond presenting the system architecture, we described a prototype that implements it and discussed a set of experiments performed in the ICAR-CNR distributed infrastructure. 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] C. Aggarwal, *An introduction to data streams*. In: C. Aggarwal (ed.) Data Streams: Models and Algorithms, pp. 1–8. Springer (2007)

[2] C. Aggarwal, P.S. Yu. *A survey of synopsis construction in data streams.* In: C. Aggarwal (ed.) Data Streams: Models and Algorithms, pp. 169–207. Springer (2007)

[3] J. Broberg, R. Buyya, and Z. Tari. 2009. *MetaCDN: Harnessing 'Storage Clouds' for high performance content delivery*. J. Netw. Comput. Appl. 32, 5 (September 2009), 1012-1022.

[4] E. Cesario, A. Grillo, C. Mastroianni, D. Talia: *A sketch-based architecture for mining frequent items and itemsets from distributed data streams*. In: Proc. of the 11th IEEE/ACM International Symposium

on Cluster, Cloud and Grid Computing (CCGrid 2011), pp. 245–253. Newport Beach, CA, USA (2011)

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

[6] J. Cheng, Y. Ke, W. Ng.: *A survey on algorithms for mining frequent itemsets over data streams*. Knowl. Inf. Syst. 16, 1–27 (2008)

[7] G. Cormode, M. Garofalakis.: *Approximate continuous querying over distributed streams*. ACM Transactions on Database Systems Vol. 33(2) (2008)

[8] G. Cormode, M. Hadjieleftheriou, *Finding the frequent items in streams of data*. Communications of the ACM 52(10), 97–105 (2009)

[9] C.D. Cranor, M. Green, C. Kalmanek, D. Shur, S. Sibal, J.E. Van der Merwe, and C.J. Sreenan, *Enhanced Streaming Services in a Content Distribution Network*, IEEE Internet Computing, 5(4): 66-75, 2001.

[10] M. Datar, A. Gionis, P. Indyk, R. Motwani: *Maintaining stream statistics over sliding windows*. SIAM Journal on Computing (SIAMCOMP) Vol. 31(6) (2002)

[11] Frequent itemset mining dataset repository. Available at http://fimi.cs.helsinki.fi

[12] C. Giannella, J. Han, J. Pei, X. Yan, P.S. Yu: *Mining frequent patterns in data streams at multiple time granularities*. In: H. Kargupta, A. Joshi, K. Sivakumar, Y. Yesha (eds.) Data Mining: Next Generation Challenges and Future Directions, chap. 3, pp. 191–210. MIT Press, Menlo Park, USA (2004)

[13] A.Y. Grama, A. Gupta, V. Kumar: *Isoefficiency: Measuring the scalability of parallel algorithms and architectures*. IEEE Concurrency Vol. 1(3) (1993)

[14] W. Kleiminger, E. Kalyvianaki, and P. Pietzuch, *Balancing load in stream processing with the cloud*. In Proceedings of the 2011 IEEE 27th International Conference on Data Engineering Workshops (ICDEW '11). IEEE Computer Society, Washington, DC, USA, 16-21.

[15] S. Loesing, M. Hentschel, T. Kraska, and D. Kossmann. 2012. *Stormy: an elastic and highly available streaming service in the cloud*. In Proceedings of the 2012 Joint EDBT/ICDT Workshops (EDBT-ICDT '12), 55-60.

[16] C. Lucchese, C., Mastroianni, S. Orlando, D. Talia.: *Mining@home: toward a public resource computing framework for distributed data mining*. Concurrency and Computation: Practice and Experience 22(5), 658–682 (2009)

[17] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, C. Olston: *Finding (recently) frequent items in distributed data streams*. In: ICDE '05: Proceedings of the 21st International Conference on Data Engineering, pp. 767–778. Tokyo, Japan (2005)

[18] C. Mastroianni, P. Cozza, D. Talia, I Kelley, I. Taylor: *A scalable super-peer approach for public scientific computation*. Future Generation Computer Systems 25(3), 213–223 (2009)

[19] C. Olston, J. Jiang, 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)

[20] M. Pathan. Ongoing Trends and Future Directions in Content Delivery Networks (CDNs), July 2011. Available from: http://amkpathan.wordpress.com/article/ongoing-trends-and-future-directions-in-3uxfz2buz8z1w-2/.

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

[22] A. Wright.: Data streaming 2.0. Communications of ACM (CACM) Vol. 53(4) (2010)