*Mining Massive Data Sets for Security*
*F. Fogelman-Soulié et al. (Eds.)*
*IOS Press, 2008*
*© 2008 IOS Press. All rights reserved.*
*doi:10.3233/978-1-58603-898-4-89*

# Data stream management and mining

Georges HEBRAIL
*TELECOM ParisTech, CNRS LTCI*
*Paris, France*

**Abstract.** This paper provides an introduction to the field of data stream management and mining. The increase of data production in operational information systems prevents from monitoring these systems with the old paradigm of storing data before analyzing it. New approaches have been developed recently to process data 'on the fly' considering they are produced in the form of structured data streams. These approaches cover both querying and mining data.

**Keywords.** Data processing, data streams, querying data streams, mining data streams.

## Introduction

Human activity is nowadays massively supported by computerized systems. These systems handle data to achieve their operational goals and it is often of great interest to query and mine such data with a different goal: the supervision of the system. The supervision process is often difficult (or impossible) to run because the amount of data to analyze is too large to be stored in a database before being processed, due in particular to its historical dimension.

This problem has recently been studied intensively, mainly by researchers from the database field. A new model of data management has been defined to handle "data streams" which are infinite sequences of structured records arriving continuously in real time. This model is supported by newly designed data processing systems called "*Data Stream Management Systems*". These systems can connect to one or several stream sources and are able to process "*continuous queries*" applied both to streams and standard data tables. These queries are qualified as continuous because they stay active for a long time while streaming data are transient. The key feature of these systems is that data produced by streams are not stored permanently but processed 'on the fly'. Note that this is the opposite of standard database systems where data are permanent and queries are transient. Such continuous queries are used typically either to produce alarms when some events occur or to build aggregated historical data from raw data produced by input streams.

As data stored in data bases and warehouses are processed by mining algorithms, it is interesting to mine data streams, i.e. to apply data mining algorithms directly to streams instead of storing them beforehand in a database. This second problem has also been studied and new data mining algorithms have been developed to be applicable directly to streams. These new algorithms process data streams 'on the fly' and can provide results either based on a portion of the stream or on the whole stream already seen. Portions of streams are defined by fixed or sliding windows.

Several papers, tutorials and books have defined the concept of data streams (see for instance [1], [2], [3], [4], [5]). We recall here the definition of a data stream given in [3]:

*"A data stream is a real-time, continuous, ordered (implicitly by arrival time or explicitly by timestamp) sequence of items. It is impossible to control the order in which items arrive, nor is it feasible to locally store a stream in its entirety."*

The streams considered here are streams of items (or elements) in the form of structured data and should not be confused with streams of audio and video data. Real-time means here also that data streams produce massive volumes of data arriving at a very high rate. Table 1 shows an example of a data stream representing electric power consumption measured by many communicating meters in households (see [6]).

**Table 1.** Example of a data stream describing electric power metering

| Timestamp | Meter | Active P (kW) | Reactive P (kVAR) | U (V) | I (A) |
|---|---|---|---|---|---|
| … | … | … | … | … | … |
| 16/12/2006-17:26:12 | 86456422 | 5,374 | 0,498 | 233,29 | 23 |
| 16/12/2006-17:26:32 | 64575861 | 5,388 | 0,502 | 233,74 | 23 |
| 16/12/2006-17:26:11 | 89764531 | 3,666 | 0,528 | 235,68 | 15,8 |
| 16/12/2006-17:29:28 | 25647898 | 3,52 | 0,522 | 235,02 | 15 |
| … |  | … | … | … | … |

This paper provides an introduction to the data stream management and mining field. First, the main applications which motivated these developments are presented (telecommunications, computer networks, stock market, security, …) and the new concepts related to data streams are introduced (structure of a stream, timestamps, time windows, …). A second part introduces the main concepts related to Data Stream Management Systems from a user point of view. The third part presents some results about the adaptation of data mining algorithms to the case of streams. Finally, some solutions to summarize data streams are briefly presented since they can be useful either to process streams of data arriving at a very high rate or to keep track of past data without storing them entirely.

## 1. Applications of data stream processing

Data stream processing covers two main aspects: (1) processing queries defined on one or several data streams, possibly also including static data; (2) performing data mining tasks directly on streams, usually on a unique stream. For both of these aspects, the following requirements should be met:

- Processing is done in real-time, following the arrival rate in the streams and not crashing if too many items arrive
- Each item of the stream is processed only once (one-pass processing)

- Some temporary storage of items of the streams may be considered but with a bounded size

We divide the applications of data stream processing in two categories:

- **Real-time monitoring (supervision) of Information Systems**. Whatever the sector of activity, current Information Systems (IS) manage more and more data for supporting human activity. The supervision of these systems (for instance supervision of a telecommunication network) requires the analysis of more and more data. It is not possible anymore to adopt the 'store then process' strategy which has been used until now by maintaining large data warehouses. There is a strong need for real-time supervision applied on detailed data instead of batch supervision applied on aggregate data. This can only done by processing data on the fly which is the approach developed by data stream processing.

- **Generic software for operational systems managing streaming data**. There are many operational Information Systems for which the main activity is to process streams of data arriving at a high rate. It is the case for instance in finance where computerized systems assist traders by analyzing automatically the evolution of the stock market. Such systems are today developed without any generic tool to process streams, exactly as first Information Systems were developed using files to store data before data base technology was available. Data stream processing solutions will provide generic software to process queries on streams, analyze streams with mining algorithms and broadcast resulting streams.

Supervision of computer networks and telecommunication networks has been one of the first applications motivating research on data streams. Supervision of computer networks consists in analyzing IP traffic on routers in order to detect either technical problems on the network (congestion due to undersized equipment) or usage problems (unauthorized intrusion and activity). These applications fall into the first category of real-time monitoring of IS. The size of data to be analysed is huge since the utmost detailed item to be analyzed is an IP packet. Typical rate is several tens of thousands of records per second to be processed when raw detailed data is already sampled by 1/100. Typical queries on streams for these applications are the following: *find the 100 most frequent couples of IP addresses (@sender, @destination) on router R1, how many different couples (@sender, @destination) seen on R1 but not R2, and this during the last hour ?*

Finance is today the domain where first commercial data stream management systems (see [7] and [8] for instance) are used in an industrial way. The main application in finance is to support trading decisions by analyzing price and sales volume of stocks over time. These applications fall into the second category of operational systems whose first goal is to manage streams of data. Typical queries on streams for these applications are the following: *find stocks whose price has increased by 3% in the last hour and the volume of sales by 20% in the last 15 minutes.*

An interesting other (artificial) application is the “Linear Road Benchmark” (see [9]). The “Linear Road Benchmark” is a simulator of road traffic which has been designed to compare the performance of different Data Stream Management Systems. It simulates road traffic on an artificial highway network. Vehicles transmit continuously their position and speed: all data are received by the Data Stream Management System which is in charge of supervising traffic by computing for each vehicle a toll price depending on the heaviness of the traffic: high traffic on a highway portion leads to a high price. Computed toll are sent in real-time to vehicles so that they can adapt their route to pay less. This system – though only imaginary – is interesting because it falls in both of the categories described above: the result of a real-time supervision is directly re-injected in the operational system.

Other applications are described in [3] and concern web log analysis and sensor networks.

## 2. Models for data streams

### 2.1. Structure of data streams

As recalled in the introduction, a data stream is an infinite sequence of items (elements). Usually, items are represented by a record structured into fields (i.e. a tuple), all items from the same stream having the same structure. Each item is associated with a *timestamp*, either by a date field in the record (this is referred to *explicit* timestamping) or by a timestamp given when an item enters the system (this is referred to *implicit* timestamping). Still related to the representation of time in streams, the timestamp can be either *physical* (i.e. a date) or *logical* (i.e. an integer which numbers the items). Implicit timestamping ensures that items arrive in the order of timestamps while it is not true for explicit timestamping. Table 1 showed an example of explicit physical timestamp. Table 2 shows an example of data stream figuring IP sessions described by implicit logical timestamps.

**Table 2.** Example of a data stream describing IP sessions

| Timestamp | Source | Destination | Duration | Bytes | Protocol |
|---|---|---|---|---|---|
| … | … | … | … | … | … |
| 12342 | 10.1.0.2 | 16.2.3.7 | 12 | 20K | http |
| 12343 | 18.6.7.1 | 12.4.0.3 | 16 | 24K | http |
| 12344 | 12.4.3.8 | 14.8.7.4 | 26 | 58K | http |
| 12345 | 19.7.1.2 | 16.5.5.8 | 18 | 80K | ftp |
| … | … | … | … | … | … |

### 2.2. Model of data streams

Following [4], the contents of a stream describe observed values which have to be related to one or several underlying signals: the *model* of the stream defines this relationship. An observation is defined by a couple $(t_i, m_i)$ where $t_i$ is a timestamp and

$m_i$ is a tuple of observed values. For instance in Table 2, $t_i=i$ since the timestamp is logical and $m_{12342}$*=(10.1.0.2, 16.2.3.7, 12, 20K, http).*

There are several possibilities to define *one* underlying signal from observations. For instance in the stream of Table 2, one may be interested in:

- the number of bytes transmitted at each timestamp. This is called the *time series model* where each observation gives directly the value of the underlying signal.
- the total number of bytes going through the router since the beginning of the stream. This is called the *cash register model* where each observation gives a positive increment to add to the previous value of the signal to obtain the new value. When increments can be negative the model is called the *turnstile model.*

Another possibility is to define *several* signals from a single stream. This is the case when the stream contains information related to several objects: one signal per object may be extracted from the stream. For instance in Table 1, one may define one signal per meter and in Table 2, one may define one signal per @sender IP address or per couple (@sender, @destination). Again the model can be the time series, the cash register or the turnstile model for each underlying signal.

Depending on the model of the stream, the transformation of observations to the defined underlying signals may be easy or difficult. For instance, the transformation requires large computations when there is a very large number of underlying signals (for instance in Table 2 if there is one signal per @sender IP address).

## *2.3. Windows on data streams*

In many applications, the end-user is not interested in the contents of the whole stream since its beginning but only in a portion of it, defined by a *window* on the stream. For instance, a typical query may be: *find the 100 most frequent @sender IP address during the last hour*. There are several ways to define windows on a stream:

- The window can be defined *logically* (in terms of number of elements, ex: the last 1000 elements) or *physically* (in terms of time duration, ex: the last 30 minutes).
- The window can be either a *fixed window* (defined by fixed bounds, ex: September 2007), a *sliding window* (defined by moving bounds, ex: last 30 minutes), or a *landmark window* (one fixed and one moving bound, ex: since September, 1st, 2007).

Finally, the end-user has to define the *refreshment rate* for the production of the results of queries/data mining tasks (ex: *give, every 10 minutes, the 100 most frequent @sender IP address during the last hour*). Again the refreshment rate can be defined either logically or physically.

## 3. Data stream management systems

Data Base Management Systems (DBMS) are software packages which help to manage data stored on disk. A model is available to define structures of data (usually the relational model) and a query language enables to define/modify structures and to insert/modify/retrieve data in the defined structures. The most popular query language is the SQL query language. A DBMS also offers other services like transaction processing, concurrency control, access permission, backup, recovery, … It is important to note that in a DBMS data is stored permanently and that queries are transient.

Though there is not yet a stable definition of a Data Stream Management System (DSMS), DSMSs can be seen as extensions of DBMSs which also support data streams. The structure of a data stream is defined in the same way as a table in a relational database (i.e. list of fields/attributes associated with a data type), but there is no associated disk storage with a stream. A data stream is connected to a source where data arrives continuously (for instance a TCP/IP port) and there is no control on the rate of arrival in the source. If the DSMS is not able to read all data in the source, the unread data is lost.

### *3.1. Continuous queries*

The key feature of a DSMS is that it extends the query language to streams. The semantics of a query has to be redefined when it is applied to streams. The concept of *continuous query* has been introduced to do so. The result of a query on a stream can be either another stream or the maintenance of a permanent table which is updated from the contents of the streams. Thus queries are permanent and process data 'on the fly' when it arrives: this explains why queries on streams are called continuous.

Let us consider the following stream describing the orders from a department store, sent to this stream as there are created:

```
ORDERS (DATE, ID_ORDER, ID_CUSTOMER, ID_DEPT, TOTAL_AMOUNT)
```

The following query filters orders with a large amount:

```
SELECT DATE, ID_CUSTOMER, ID_DEPT, TOTAL_AMOUNT
FROM ORDERS
WHERE TOTAL_AMOUNT > 10000;
```

This query can be processed 'on the fly' (without storing the whole stream) if its result is output as another stream.

Let us now consider the following query computing the sum of sales per department:

```
SELECT ID_DEPT, SUM(TOTAL_AMOUNT)
FROM ORDERS
GROUP BY ID_DEPT;
```

The result of this query is typically a permanent table which is updated from the contents of the stream. Note that the stream does not need to be stored since the update of the result can be done incrementally. It is also possible to generate another stream from the permanent updated table by sending to this new stream at the refreshment rate either the whole table, the inserted tuples or the deleted tuples between the refreshment timestamps.

These two queries can be extended easily to join the stream to a permanent table describing customers, for instance for the last query:

```
SELECT ORDER.ID_DEPT, CUSTOMER.COUNTRY, SUM(TOTAL_AMOUNT)
FROM ORDERS, CUSTOMER
WHERE ORDERS.ID_CUSTOMERS=CUSTOMER.ID_CUSTOMER
GROUP BY ID_DEPT, CUSTOMER.COUNTRY;
```

Again, this query can be evaluated without storing the stream, by joining on the fly the tuples from the stream to the ones of the permanent table and maintaining incrementally the aggregates. This is true when the customer information is available in the permanent table at the time an order corresponding to the customer arrives in the stream.

But things are not always so easy. There are many cases where it is not possible to produce the result of the query without storing the whole stream. We define a second stream containing the different bills associated with orders, several bills being possibly associated with one order and arriving at different times:

```
BILLS (DATE, ID_BILL, ID_ORDER, BILL_AMOUNT)
```

Let us consider the following query which is also an example illustrating that joins of several streams can be handled by a DSMS:

```
SELECT ORDERS.ID_ORDER, TOTAL_AMOUNT - SUM(BILL_AMOUNT)
FROM ORDERS, BILLS
WHERE ORDERS.ID_ORDER = BILL.ID_ORDER
GROUP BY ORDERS.ID_ORDER, TOTAL_AMOUNT
HAVING TOTAL_AMOUNT - SUM(BILL_AMOUNT) != 0;
```

This query lists the orders for which the total amount has not been covered exactly by bills. To process this query, it is not necessary to keep in memory the elements of the BILLS stream once they have been joined to their corresponding ORDERS element. But it is necessary to keep all elements of the ORDERS stream: indeed, since the number of bills associated with an order is not known, the system always has to expect a new bill to appear for each arrived order. Such queries are called ‘blocking queries’ because they cannot be evaluated ‘on the fly’. There are several solutions to this problem and the most common one is to define windows on the streams. The user specifies on each stream a window on which the query applies, for instance:

```
SELECT ORDERS.ID_ORDER, TOTAL_AMOUNT - SUM(BILL_AMOUNT)
FROM ORDERS [LAST 30 DAYS], BILLS [LAST DAY]
WHERE ORDERS.ID_ORDER = BILL.ID_ORDER
GROUP BY ORDERS.ID_ORDER, TOTAL_AMOUNT
HAVING TOTAL_AMOUNT - SUM(BILL_AMOUNT) != 0;
```

This new query is restricted to the production of the result considering only the orders of the last 30 days and the bills of the last day. The query is then 'unblocked' and can be evaluated with a limited temporary storage of the contents of the streams (last 30 days for orders and at the most last day for bills). It is interesting to note that the end-user himself specifies how to limit the scope of the query by accepting that bills not issued 30 days after their orders will never be considered to produce a result in this query.

## *3.2. Main features of a DSMS*

As mentioned before, a DSMS provides a data model to handle both permanent tables and streams. It enables the user to define continuous queries which apply to both permanent tables and streams and produce either new streams or update some other permanent tables. There are two main approaches to provide such a facility to the user: (1) the definition of an extension of the SQL language (see for instance the STREAM [10] and TelegraphCQ [11] systems), (2) the definition of operators applicable to streams which are combined to produce the desired result (see for instance the Aurora system [12]).

When designing a DSMS, several problems have to be solved, the most challenging ones being the following:

- Defining for a set of continuous queries an optimized execution plan which reduces temporary storage and allows shared temporary storage between the queries. Note that the typical use of a DSMS (see the Linear Road benchmark in [9]) leads to the definition of several queries producing intermediate streams which are then used by several other queries. Such execution plans have to be dynamic because the rate of arrival in streams may evolve and new queries may be defined by the user. Interesting solutions have been studied in the STREAM project (see [10]).
- Being able to face important variations in rates of arrival in input streams. As seen before, data arriving in data streams are either processed immediately or lost. DSMSs must ensure that there is no crash in this case and that the performance deteriorates in a controlled way. The main approach – called *load shedding* - is to place random sampling operators so that all elements of the streams are not processed when the system is overloaded. The placement of these operators is optimized dynamically to maximize a function of quality of service (see [12], [13] for instance).

## *3.3. Main existing DSMSs*

Existing DSMSs fall into two main categories: (1) general purpose DSMSs, (2) specialized DSMSs dedicated to a particular domain.

General-purpose DSMSs

In this category, the most famous ones are the following research prototypes: STREAM ([10]), TelegraphCQ ([11]), and Aurora-Medusa-Borealis ([12]). STREAM is the result of a project at Stanford University: a new system has been built from scratch and has introduced many new concepts and solutions related to the management of data streams. In particular, the CQL language has been defined as an extension of the SQL language to handle continuous queries with windowing facilities. Though a demo is available on-line and the software can be downloaded, it is not really usable in practice (see [6]). TelegraphCQ has been developed at the University of Berkeley as an extension to the PostgreSQL system to support streams and continuous queries defined on streams. New continuous queries can be added dynamically to the system and it is usable for applications where there is no need to process very fast streams. The Aurora system has been developed at Brown University, MIT and Brandeis. In this system, queries are defined by combining operators which take streams as input and produce a new stream as output. A lot of interesting work has been done on load shedding by defining and optimizing a function of quality of service when the system becomes overloaded. The follow-up projects Medusa and Borealis focused on a distributed version of Aurora.

There are also a few commercial general-purpose DSMSs. The Aurora project led to the Streambase software (see [8]). The TelegraphCQ project led to the Aminsight/Truviso software (see [7]).

Specialized DSMSs

Specialized DSMSs have been developed to solve data stream problems from a particular domain of application. In these systems, specific operations of the domain are optimized. The most popular systems are Gigascope [14] (network monitoring), Hancock [15] (analysis of telecommunication calls), NiagaraCQ [16] (large number of queries on web contents), Statstream [17] (correlations between a large number of sensors) and Traderbot [18] (analysis of stock market).

A complete survey of DSMSs which was up to date in 2004 can be found in [19]. It also includes work on sensor networks which is related to data stream management.

## 4. Data stream mining

Data stream mining can be defined as *'applying data mining algorithms to one or several data streams'*, with the same constraints as DSMSs: (1) one-pass, (2) limited memory and CPU. From a user point of view, the data mining task can apply either to the whole stream since its beginning or to a part of it defined by a window. Depending on the type of window, the algorithms are different:

- Whole stream: algorithms have to be incremental. This is the case for instance for neural network algorithms which are incremental in nature. Some nonincremental algorithms can be modified to do so. This has been done for instance for decision trees (see below).

- Sliding window: algorithms have to be incremental and have the ability to forget the past of the stream though the corresponding elements are not accessible anymore. This is the case for additive methods like PCA (see below).
- Any past portion of the stream not decided in advance: algorithms have to be incremental and some summaries of the past must be kept in limited memory. Summaries of data streams can be done by several means; a solution based on micro-clustering is described below.

We do not cover here all stream mining algorithms but only illustrate these three cases by one example of each.

## 4.1. Building decision trees on data streams

Some early work has been done by Domingos and Hulten to build decision trees incrementally from data streams (see [20]). The idea is that it is not necessary to wait for all examples to decide of a split in the tree. A statistical test is performed to decide when a split can be done. The statistical test can be done by keeping track of appropriate statistics which can be computed incrementally in bounded space.

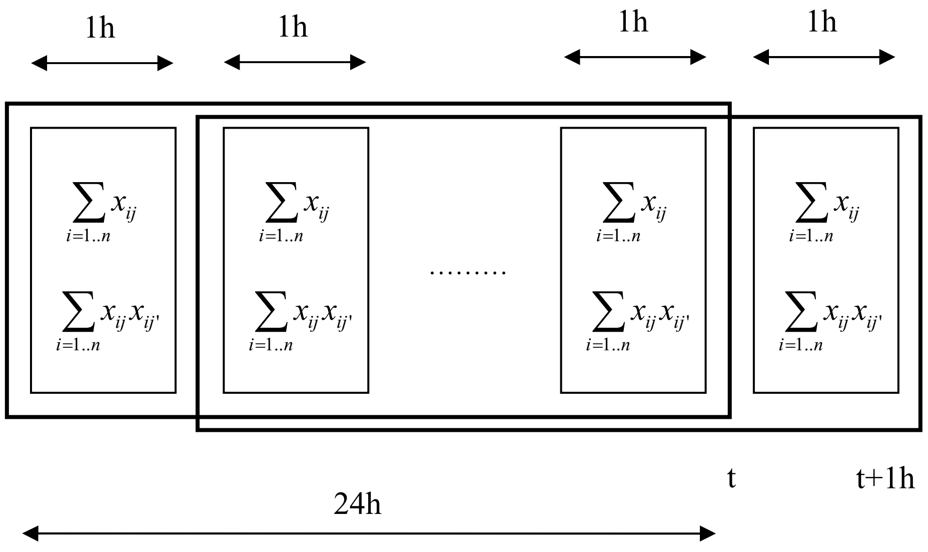

**Figure 1.** PCA on a sliding window

### 4.2. Performing PCA incrementally on a sliding window

Principal Component Analysis (PCA) is performed by singular value decomposition (*svd*) of the covariance (correlation) matrix between variables. It is easy to prove that the covariance matrix can be computed incrementally. Moreover, if the sliding window is for instance 24 hours and the refreshment rate is 1 hour, it is possible to maintain 25 covariance matrices as shown in Figure 1. At the end of each hour period, the covariance matrix can be computed from the last 24 ones and the *svd* is performed. This solution is applicable to any additive or pseudo-additive data mining method.

### 4.3. Performing clustering on any past portion of the stream

Several clustering algorithms have been developed to be applicable to data streams (see for instance [21]). The *Clustream* algorithm (see [22]) allows to perform clustering on any past portion of a stream where all fields are numerical. This algorithm works in two steps: (1) a clustering is performed on-line to build and maintain a large number of micro-clusters whose contents follow the contents of the stream, (2) a clustering of the micro-clusters is done off-line on demand to produce the final clustering.

The key concept of this approach is that micro-clusters are defined in such a manner that it is possible to ‘substract’ micro-clusters at date T1 from micro-clusters at date T2 to obtain micro-clusters describing the stream between T1 and T2. This is done by storing for each micro-cluster additive statistics in a structure called *Cluster Feature Vector CFV* (sum of values and sum of squares for every field, including the timestamp field). Snapshots of the micro-clusters are taken regularly and compacted logarithmically as micro-clusters get older: this approach is called *tilted time windowing* and ensures that the summary of the whole stream is stored with bounded memory (see Figure 2). The drawback is that there is less detail for older data than for recent data but this is often acceptable in applications.

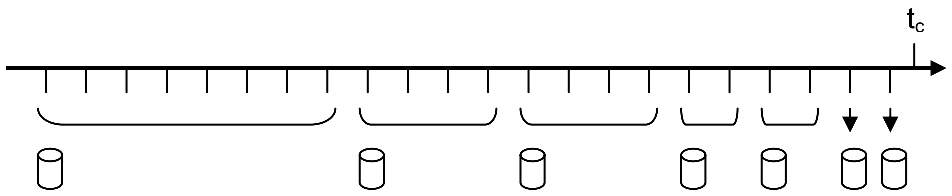

**Figure 2.** *Tilted time* structure: the size of storage decreases logarithmically when data gets older

## 5. Synopses structures

A strong constraint both in querying and mining data streams is to process data ‘on the fly’, facing variations in input streams. For many applications, it is nevertheless necessary to keep track of information related to a maximum number of elements of the stream but in bounded space. As seen in the previous sections, a common way to do so is to restrict the scope of the query (or the mining task) by windowing. In many applications, this solution is not efficient enough and much work has been done to summarize the contents of data streams in order to produce an approximate but controlled result.

Synopses structures can be used both for answering queries and applying data mining algorithms to streams. Many synopses structures have been studied. We present here a few examples of them.

We can distinguish between temporal and memory management approaches. The most famous approach to temporal management is the concept of *tilted time windows* (introduced in [23]) which compacts historical data by forgetting details in a logarithmic way with time, thus keeping a summary of the past with bounded memory. This approach of temporal management can be combined to any memory management approach. For instance, Clustream combines tilted time windows and micro-clustering.

As for memory management, many approaches have been studied, mainly: random samples, histograms and quantiles, micro-clusters and sketches. Since Clustream is an example of synopsis in the form of micro-clusters, we only develop below two approaches: random sampling and sketches.

## Summarizing by random sampling

On-line random sampling from data streams is a problem because the size of the sample has to be bounded and the size of the data set from which elements are selected is not known when sampling begins, i.e. at the beginning of the stream. Some algorithms exist to overcome this problem: the *reservoir* algorithm (see [24]) maintains a fixed size uniform random sample from the beginning of the stream to current time. Some elements are selected randomly from the stream with a decreasing probability and replace randomly elements already in the sample. The decreasing probability ensures that the sample is uniform over the period.

Some extensions and new approaches have been developed to extract random samples from a sliding window defined on the stream (see [25]). In this case, the problem is that elements of the sample have to be removed and replaced by new ones when they expire from the window, under the constraint of still keeping uniform the sample.

A combination of random sampling and tilted time windowing is described in the paper of Baptiste Csernel in this book.

## Summarizing with sketches

Sketches are data structures dedicated to a specialized task which can be updated incrementally to maintain in small space summarized information. They are often based on the use of hash functions which project randomly stream data on the small structure. Approximate answers to queries can be obtained from sketches with probabilistic bounds on errors. Since data streams produce data with a high rate, the probabilistic bounds are quite tight. Popular sketches are the Flajolet sketch which counts the number of distinct objects appearing in a stream (see [26]) and the count sketch algorithm (see [27]) which maintains the $k$ most frequent items in a stream. These two approaches provide the result with storage much smaller than the number of distinct items appearing in the stream.

## Conclusion

This paper has introduced the main concepts of data stream management and mining. The two main directions are the development of Data Stream Management Systems (DSMS) which enable to query one or several streams, and the development of data stream algorithms which enable to mine data streams without storing them entirely. Synopses are basic tools which enable to process streams using a smaller storage space: they can be used both in query and mining tasks.

As for perspectives on DSMSs, there is some current work on event processing systems which provide a more convenient querying language for streams describing events (see for instance the Cayuga [28] and SASE [29] projects). The first commercial DSMSs appear on the market and are used mainly in financial applications.

As for perspectives on data stream mining, one challenge is to build summaries of the whole stream in order to be able to apply data mining tasks to any past portion of the stream. Another challenge is to apply data mining algorithms to several streams, without joining them but keeping a summary of each stream which preserves the relationship between the streams. The paper of Baptiste Csernel in this book describes an approach to do so.

## References

[1] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom, *Models and Issues in data stream systems*, PODS'2002, 2002.

[2] M. Garofalakis, J. Gehrke, R. Rastogi, *Querying and Mining Data Streams: You Only Get One Look. A tutorial*, Tutorial SIGMOD'02, Juin 2002.

[3] L. Golab, M.T. Özsu, *Issues in Data Stream Management*, Canada. SIGMOD Record, Vol. 32, No. 2, June 2003.

[4] S. Muthukrishnan, *Data streams: algorithms and applications*, In Foundations and Trends in Theoretical Computer Science, Volume 1, Issue 2, August 2005.

[5] C.C. Aggarwal, *Data streams: models and algorithms*, Springer, 2007.

[6] T. Abdessalem, R. Chiky, G. Hébrail, J.L. Vitti, *Using Data Stream Management Systems to analyze Electric Power Consumption Data*, International Workshop on Data Stream Analysis (WDSA),. Caserta (Italie), March 2007.

[7] Amalgamated Insight/Truviso, http://www.truviso.com

[8] Streambase software, http://www.streambase.com

[9] A. Arasu, M. Cherniack, E. Galvez, D. Maier, A.S. Maskey, E. Ryvkina, M. Stonebraker, R. Tibbetts, *Linear Road: A Stream Data Management Benchmark*, Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004. http://www.cs.brandeis.edu/~linearroad/

[10] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, J. Widom. *STREAM: The Stanford Data STREAM Management System*, Department of Computer Science, Stanford University. Mars 2004. Available at: http://www-db.stanford.edu/stream

[11] S. Chandrasekaran, O. Cooper, A. Deshpande, M.J. Franklin, J.M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, M. Shah, *TelegraphCQ: Continuous Dataflow Processing for an Uncertain World*, CIDR 2003. http://telegraph.cs.berkeley.edu/telegraphcq/v2.1/

[12] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, S. Zdonik, *Aurora: A New Model and Architecture for Data Stream Management*, In VLDB Journal (12)2: 120-139, August 2003.

[13] B. Babcock, M. Datar, R. Motwani, *Load Shedding for Aggregation Queries over Data Streams*, 2004. Available at:http://www-db.stanford.edu/stream

[14] T. Johnson, C.D. Cranor, O. Spatscheck, *Gigascope: a Stream Database for Network Application*, ACM SIGMOD International Conference on Management of Data, June 9-12, 2003.

[15] C. Cortes, K. Fisher, D. Pregibon, A. Rogers, *Hancock: a language for extracting signatures from data streams*, 2000 KDD Conference, pages 9–17, 2000.

[16] J. Chen, D. J. DeWitt, F. Tian, Y. Wang, *NiagaraCQ: A Scalable Continuous Query System for Internet Databases*, SIGMOD Conference 2000: 379-390.

[17] Y. Zhu, D. Shasha, *StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time,* VLDB, Hong Kong, China, August, 2002.

[18] Traderbot home page, http://www.traderbot.com (closed since July 2007).

[19] V. Goebel, T. Plagemann, *Data Stream Management Systems - Applications, Concepts, and Systems*, Tutorial MIPS'2004, 2004.

[20] P. Domingos, G. Hulten, *Mining High-Speed Data Streams*, In Proceedings of the 6th ACM SIGKDD conference, Boston, USA, pages 71-80, 2000.

[21] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan, *Clustering data streams*, In IEEE Symposium on Foundations of Computer Science. IEEE, 2000.

[22] C.C. Aggarwal, J. Han, J. Wang, and P.S. Yu, *A framework for clustering evolving data streams*, In Proceedings of the 29th VLDB Conference, Berlin, Germany, 2003.

[23] Y. Chen, G. Dong, J. Han, B. Wah, J. Wang, *Multidimensional regression analysis of time-series data streams*, VLDB Conference, 2002.

[24] J. Vitter, *Random sampling with a reservoir*, ACM Trans. Math. Softw., 11(1) :37–57, 1985.

[25] B. Babcock, M. Datar, R. Motwani, *Sampling from a moving window over streaming data*, In Proceedings of the thirteenth annual ACM-SIAM SODA, 2002.

[26] P. Flajolet, *Probabilistic Counting Algorithms for Data Base Applications*, In Journal of Computer and System Sciences, Volume 32, Issue 2, page 182-209, Sept.1985

[27] M. Charikar, K. Chen, M. Farach-Colton, *Finding frequent items in data streams*, Theor. Comput. Sci., 312(1) :3–15, 2004.

[28] L. Brenna, A. Demers, J. Gehrke, M. Hong, J. Ossher, B. Panda, M. Riedewald, M. Thatte, W. White, *Cayuga: A High-Performance Event Processing Engine*, ACM-SIGMOD 2007.

[29] D. Gyllstrom, E. Wu, H.J. Chae, Y. Diao, P. Stahlberg, G. Anderson. *SASE: Complex Event Processing over Streams*. Demo in CIDR 2007, Asilomar, CA, January 2007.