Annals of the University of Craiova, Mathematics and Computer Science Series
Volume 38(3), 2011, Pages 66–75
ISSN: 1223-6934, Online 2246-9958

# Data stream management systems: a response to large scale scientific data requirements

Sabina Surdu

Abstract. Exact sciences research communities are dealing with data sets that are expected to reach exascale sizes in the years to come. Analysing such data sets becomes a tedious task, when using classical data processing paradigms. An increasing number of domains from exact sciences, like radio astronomy or financing, are handling data streams, which are sequences of values produced over time by data sources. Data streams require new processing approaches and a number of prototypes have already been implemented. As an important application of informatics in exact sciences, we consider the case of Data Stream Management Systems, as a response to increasingly large scale data in a great number of fields. We aim at highlighting the main challenges in data stream processing. Based on identified difficulties, we present a preliminary set of five principles, SCIPE, that we plan to use in further research work. The final purpose would be to design and develop a dedicated Scientific Data Stream Management System.

*2010 Mathematics Subject Classification.* Primary 68P15; Secondary 68N01.
*Key words and phrases.* data streams, continuous processing, exact sciences, extremely large databases, data stream management systems.

## 1. Introduction

Nowadays scientific databases are reaching petascale sizes and in the years to come these repositories are expected to exceed exascale dimensions. The research community is constantly seeking for new data models and data processing paradigms for these extremely large databases. Sustained efforts are made in niches like data storage and distribution, data mining analysis or query optimization.

But in domains like radio or optical astronomy data arrives in continuous streams from an ever increasing number of celestial bodies and is captured by field-specific telescopes. Seismometers perpetually monitor continuous streams of seismic waves to study seismic activity, in an attempt to predict earthquakes. Radiation monitoring devices scan sequences of environment radiation data, in order to emit alerts when the ration level surpasses a given threshold. In quantum physics streams of information from the atomic world describe the activity of elementary particles. Financial data also takes the form of data streams. Weather satellites continuously stream collected data in order to monitor the planet's climate. Telephone calls also generate streams of phone records. And the examples can go on.

As we can see, in a large number of scientific domains data is presented as an arbitrary number of continuous streams. A data stream is a continuous sequence of data, provided by a data source as time goes by. Storing all this data is impossible,

Received July 02, 2011. Revision received August 20, 2011.
This paper is supported by the Sectoral Operational Programme Human Resources Development (SOP HRD) 2007 - 2013, co-financed by the European Social Fund, under the contract number POSDRU/6/1.5/S/3.

with the current technology. Whereas storage capabilities are limited, data streams are practically infinite. Only part of this data needs to be stored for future querying purposes, which are domain dependent. However, a large number of scientific applications that need to handle data streams require that a considerable fraction of the data should be processed on the fly and then discarded or stored for a short period of time.

Consider a financial application that must alert its subscribers whenever a stock is below a given value more than twice in the last 3 minutes. At any given time instant, a processing system should only store the data that arrived on the financial stream in the last three minutes. Processing is performed on this three minutes window of data and then the data is discarded, as new information arrives on the stream.

As an important application of informatics in exact sciences, we consider the case of Data Stream Management Systems, as a response to increasingly large scale data in a great number of scientific domains.

Data Stream Management Systems (DSMSs) are dedicated systems that process streams of continuous data. If traditional Database Management Systems (DBMSs) handle stored and finite data relations, DSMSs cope with transient data streams, which come from a various number of data sources, cannot be entirely stored and are virtually unlimited in size. DSMSs process data streams by executing continuous queries, which perpetually run over their continuous input data. A number of prototype DSMSs have been implemented. We will refer to a selection of these in our paper and we will also mention a key benchmark, whose experiments show that a DSMS outperforms a DBMS in data stream processing.

We intend to show how a dedicated DSMS can serve the needs of applications in a number of domains from exact sciences. We will touch processing paradigms that are still in their infancy, like load shedding and operator scheduling, in a continuous data processing context. We will show how a DSMS can provide a scientific application with the data analyses it needs, in a continuous flow processing paradigm. We propose a set of principles which can guide the design of what we call a Scientific Data Stream Management System (Sci-DSMS). Starting from related work in the field of data stream processing, we take into consideration general goals in exact sciences data processing and draw the general directions than can be followed in the process of designing a Sci-DSMS.

In the rest of the paper, we use the term *scientific* to refer to aspects from exact sciences, *i.e.* domains that use precise and rigorous methods to express quantifiable results.

This paper is organized as follows. Section 2 provides an introduction to the data stream processing paradigm. The main theoretical aspects are revealed, placing them in a comparative light with traditional data processing approaches. Section 3 highlights the key features of DSMSs. In Section 4 we take a brief look at large scale scientific databases. Section 5 discusses the data model and performance issues in specific DSMSs. In Section 6 we present the set of principles that should guide the design of a Sci-DSMS. Section 7 concludes on the results of our work and provides future research directions.

## 2. Data stream processing

**2.1. Data streams.** Data streams take the form of sequences of values, which are provided over time by data sources. In radio astronomy the data sources could be the

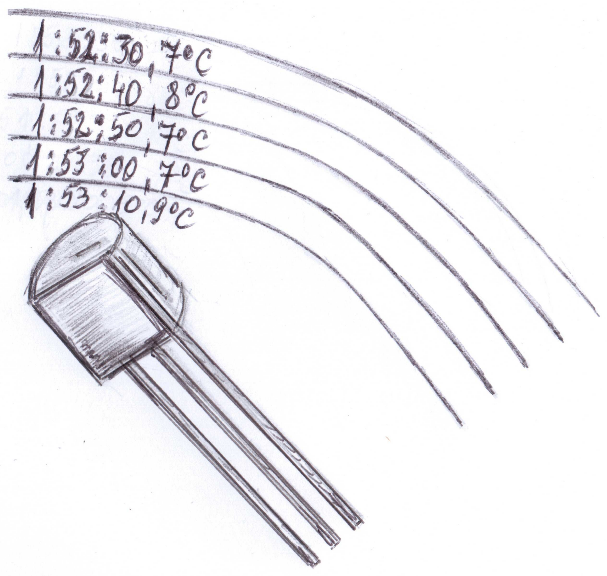

FIGURE 1. Temperature tuples from a data stream provided by a temperature sensor

planets and stars that emanate radio waves. Weather satellites sensors can also be viewed as data sources. Although the real data source is the weather, we will make the assumption that the data source is the device that produces data in a digital manner. Temperature sensors for instance provide streams of temperature notifications. In financial services, a stock exchange is a data source that provides data to stock ticker feeds providers, which act as a data source in turn and pass data along to end-users.

As we can see, data streams are omnipresent. In recent years, research community has started to propose data models and querying capabilities that can cope with data streams. The main difference between data streams and data from traditional DBMSs is the fact that the latter doesn't have the built-in notion of time. For data streams time is an essential feature, as data sources produce their elements as time goes by. Each element on a stream is associated with a time-based value, indicating when the element was produced, *i.e.* a timestamp.

In our examples data is structured, resembling records in traditional relations from DBMSs. Data that arrives on streams can also take the form of images for instance. In this paper we will only consider systems that process streams of structured data records, encompassing fields with basic data types like strings, numbers, etc. We chose to stick to this palette of systems because most users are familiar with relational databases and the querying languages they use. In the rest of the paper we will refer to a unitary piece of data that arrives on a stream with the terms tuple, record or element. FIGURE 1. shows five tuples of temperature notifications, produced by a temperature sensor which emits a temperature tuple every ten seconds.

**2.2. Query evaluation.** Query evaluation on data streams is usually performed in a window-based manner. Data streams are infinite and hence cannot be stored, as system resources are limited. Consider the case of a temperature sensor from a refrigeration container. One may be interested to know when the temperature in the container exceeds a given threshold for more than ten consecutive time instants in the last three minutes. In this case, even if all the data on the stream of temperatures could be stored, this approach would be highly ineffective. One should only store a window of data that arrived on the stream in the previous three minutes and perform desired computation on this fraction of data. As new tuples appear on the stream, the data window slides over them, refreshing itself, hence the name sliding window.

Traditional DBMSs perform one-shot queries on their input data [3]. These queries are executed against stored, finite data relations and they complete in a finite amount of time. Systems that handle data streams on the other hand execute continuous queries over them. A continuous query runs over time and updates its results based on new input data it receives. In the previous example, a continuous query could run perpetually over the temperature notifications stream and compute the result (which is itself a stream) on windows of three minutes data.

One can control the rhythm at which to execute continuous queries. [15] introduces a novel way of categorizing continuous queries. Change-based continuous queries execute whenever a new element appears on the data stream. In our example, this means the sliding window refreshes with every newly arrived element on the stream. Timer-based continuous queries are executed at specified time instants, *i.e.* every 10 seconds. For our example, this means at every 10 seconds the sliding window contains all the elements that arrived on the stream in the last three minutes and the continuous query is executed against this window. This way system resources as memory and CPU are saved, at the expense of some (bounded) inaccuracy in results (for instance, one could miss a window of data when the temperature exceeds the threshold for more than ten consecutive time instants in the last three minutes).

Based on our example, we can state that a continuous query resembles a view or a condition statement from a trigger. From this perspective, one could add a great number of triggers to a DBMS and perform continuous processing in a traditional, although enhanced context. [1] however shows that a classical DBMS doesn't scale well past a certain number of triggers, whereas a monitoring application could easily track hundreds of streams with a great number of running continuous queries.

## 3. Data Stream Management Systems

DSMSs are dedicated systems, built around stream processing engines, that process data streams using continuous queries. These systems emerged to serve the needs of monitoring data-intensive applications, that scan streams of data in order to produce relevant results.

A key benchmark for comparing the performances of DSMSs relative to each other and to traditional DBMSs is the Linear Road [6]. In three hours simulation experiments, the benchmark clearly showed that a DSMS can outperform a DBMS by at least a factor of five, when processing high load data streams and executing continuous and one-shot queries.

DSMSs perform better than DBMSs in the context of data streams and continuous queries because they address fundamental demands of monitoring applications. We enumerate the most important ones.

Continuous queries provide results in a real time manner, with a low latency requirement. In a variable tolling system [24] that computes highway tolls based on dynamic factors such as accident proximity or traffic congestion, a driver must be alerted in real time whenever a new toll is issued for his or her car. Providing this answer later in the future would be of no use.

Data streams are processed on the fly and then discarded. Once an element has been seen and processed by the processing engine, it can no longer be retrieved. Although some history over the data streams can be retained, in the end it is discarded.

The number of data sources can dramatically increase. So can the stream tuple rate, *i.e.* the rhythm at which tuples arrive on the stream. The system can get overloaded and may no longer be able to provide query results in a timely manner. Various strategies are used to deal with high tuple rate, which we will later describe in this paper.

Another processing paradigm difference is that the stream processing engine (SPE) must react to arriving data, as opposed to traditional query processing engines that orchestrate their data movement [14]. The SPE is not in complete control over the input data any more.

If DBMSs can rely on all their available data to compute exact queries answers, DSMSs can only compute approximate answers, based on the window of data their queries consider.

## 4. Large scale scientific databases

Recent years have brought an explosion of data sets, both in their volume and their complexity. [16] shows that in 2007 the world's storage capacity was 290 exabytes (1 exabyte = $10^{18}$ bytes). Data sets of petascale sizes are encountered not only in the industry, but also in scientific domains.

The Extremely Large Dabases community [25] has started a series of workshops in order to address the challenges faced by both academic and industrial users, when dealing with ever increasing data sets. Representatives from oil and gas domains, radio astronomy, medicine, bioinformatics, finance, geoscience or high-energy physics complained over the last years about current lack of solutions to problems arising from extremely large data sets, like statistical analysis tools or approximation and sampling techniques that cannot cope with nowadays volumes of data [11].

Whereas some challenges remain domain-specific, others arise in the majority of the considered scientific domains. In radio astronomy, finance or marketing the number of data sources and tuple rate can be incredibly large. Thousands of billions of celestial bodies emanate radio waves every time instant, transaction records (whether from a marketer or a credit card processor point of view) arrive on streams at an ever increasing rate. Analysing scientific data from these sources can prove to be a tedious task. For domains like these we consider the opportunity of designing dedicated scientific data stream management systems, whose design and development is performed having general scientific goals in mind.

## 5. Data model

**5.1. Aurora, STREAM, Medusa and Borealis.** We will exemplify on four known DSMSs what we consider to be the core current challenges in nowadays stream processing. In the following section we will explore these challenges from a Scientific DSMS point of view.

Aurora is a DSMS which allows specifying queries in a visual manner [10]. Using a boxes-and-arrows paradigm, Aurora displays queries as networks of operators, connected by data streams. On a query diagram, the boxes represent operators and the arcs that connect the boxes are flowing streams of data.

STREAM is a general-purpose DSMS [4], which builds queries using a designated SQL-like query language, CQL (Continuous Query Language [5]). Queries written in CQL are transformed into physical query plans, composed of operators, inter-operator queues and synopses. Operators perform the query processing, queues buffer input and output data stream tuples and synopses store operators state, when required.

Medusa is a distributed DSMS that expresses its queries using the query diagrams from Aurora [23]. Medusa actually uses Aurora as a single node processing engine.

Borealis is a distributed second generation DSMS, which uses Aurora's stream processing techniques and Medusa's distributed functionalities [2]. Therefore, Borealis uses as well the boxed-and-arrows paradigm to express its queries.

**5.2. Data stream model.** The above mentioned DSMSs consider a data stream $S$ to be a sequence of tuple-timestamp pairs $(s, t)$ that appear over time, where $s$ is the tuple that arrives on stream $S$ and $t$ is the timestamp when $s$ appeared on $S$. The tuples conform to the stream schema of named attributes (similar to records that follow a relation schema in a relational DBMS).

Apart from continuous queries, these systems also allow one-shot queries, expressed against data sets equivalents. STREAM for instance contains the relation data type [17]. Dealing with finite stored relations is out of the scope of our paper.

An interesting improvement over this model can be observed on Borealis. Whereas all the other three systems allow only tuple insertion on their streams, Borealis encompasses a technique of correcting previously arrived records on streams [12], enabling stream tuples delete and update operations [18].

Borealis also allows its queries specification to be changed at runtime, in a non-disruptive manner. This means the user can change some parameters of the query, *i.e.* selected fields, or even the query operators while the query network is running.

**5.3. Improving performance.** When data stream rates are very high or the stream schema has tens of attributes, the DSMS uses certain strategies in order to improve performance. Otherwise, tuples can accumulate at various places in the system, compromising its ability to provide results in a timely manner.

**5.3.1.** *Load shedding.* Load shedding is the action taken by a DSMS when the tuple rate exceeds its abilities to provide desired results in a real time manner.

Aurora developed two types of load-shedding algorithms: Random-LoadShedding and Semantic-LoadShedding [22]. For both approaches, Aurora and its related systems Medusa and Borealis use the notion of Quality of Service.

Quality of Service encompasses parameters that can refer to the latency bound allowed for a DSMS queries or input tuples that are important and should not be subject to load shedding during overload [19].

In Aurora every output tuple is associated with multiple QoS functions. Examples of these include dropped tuple-based (indicates the amount of dropped input tuples), time-based (specifies the latency requirement for query results) and value-based (indicates which output values are more important than others) [21]. The random load shedding algorithm drops elements in a random manner at strategic points in the query diagram. The semantic load shedding algorithm takes a superior approach. It takes into account value-based QoS, if such data is available, when choosing which tuples to drop, so that less important tuples are more likely to get dropped than more important tuples. This way the semantics of the application is impacted in a minimal manner.

Both Aurora and Borealis use a Window Drop operator for their aggregation queries [20]. Previosuly described algorithms extract and drop tuples from within a window. This operator has the advantage of maintaining windows integrity, during load shedding, by disallowing tuples that get dropped to start new windows.

STREAM's load shedding approach works best with aggregation queries [9]. The system inserts load shedder operators on certain places in the query operator tree and assigns them a probability value. Every tuple that flows through a load shedder is discarded with the load shedder's assigned probability.

**5.3.2.** *Operator scheduling.* In order to save memory and CPU, DSMSs use various operator scheduling strategies, that lead to less intermediate state used and less performed computations, while still achieving desired results.

Aurora performs a train superbox scheduling approach, which batches multiple input tuples into trains and pushes them through multiple boxes in the query diagram[13]. This way, the box call overhead is avoided, as is the cost of spilling data to disk. So does Borealis. Borealis however can also schedule operators in a way that maximizes the chances to produce significant data values as output (using QoS metrics on both input and output values).

STREAM uses a FIFO approach to operator scheduling when the data arrival rate is uniform [8] and a dedicated Chain algorithm when the tuple rate is erratic [7], in order to reduce intermediate state.

## 6. A Scientific DSMS - design principles

The first aspect that must be considered when dealing with data streams from exact sciences is their size. Some of the large scientific data sets described in Section 4 are constructed by storing and archiving arriving elements on streams. We believe that, in the case of some scientific domains, scientific structured data that arrives on streams fits into the data model presented in Section 5. Instead of receiving the input streams and archive or store the elements for later use, an on the fly processing technique that encompasses both summarization and results computation should be designed. We chose this to be the first from our set of principles oriented towards a Sci-DSMS design. This principle is domain and application dependent, *i.e.* in some fields or applications there might be a need to access each of previously arrived data elements on a stream, hence summarization may not be able to answer the results of such queries.

As an alternative to the above approach, if individual elements storage is necessary, one could store tuples up to a certain moment in the past. Past that moment, all tuples should be either summarized or discarded. This is our second principle.

The third principle is inspired from Borealis. When designing a Sci-DSMS one should take into account the possibility to correct previously arrived data on input streams. Revision processing is a great enhancement of Borealis and saves the daunting task of dealing with incorrect query results, due to incorrect input data.

The load shedding strategies used by the described systems can also be successfully applied in a designed Sci-DSMS. Load shedding is necessary in a Sci-DSMS, where tuple rate can be erratic and data can appear in high loads. The best load shedding technique to use is however domain dependent. Astronomical data could be less interesting in certain periods of time for instance, whereas for the same time intervals financial data could arise the greatest interest. However, we do believe that, no matter the chosen approach, load shedding algorithms should take into account QoS information, if available, in order to perform a semantic load shedding.

Last but not least, one should consider the interaction with a Sci-DSMS from the scientist point of view. Researchers from different domains are not necessarily database experts. Generally speaking, a monitoring application can be the interface between the researcher and a DSMS. But we embark on designing a Sci-DSMS which attempts to answer most of the needs of a researcher. Therefore, we aim at eliminating the interface and let the scientist work directly on the Sci-DSMS. We still maintain the idea of customizing the Sci-DSMS depending on the scientist's domain, but this will be the subject of future work. Hence, we aim to design a system where data and query specification are performed in a user-friendly manner, that is a visual programming language combined with an SQL-like declarative query interface (as the reader may have noticed, the sources of inspiration are Aurora's GUI and STREAM's CQL).

We concisely present our SCIentific data stream processing PrinciplEs - SCIPE:

1. Whenever possible: process, then summarize or discard a data tuple. This has a great impact on storage, whereas still keeping the tuple value for subsequent querying, in a summarized manner, if necessary.

2. If individual elements storage is necessary, store only tuples from the recent past and discard or summarize old elements. The effect from principle 1 is obtained following this principle as well.

3. Design a revision processing enabled system.

4. Shed load in a domain-dependent manner, using QoS information, when available.

5. Construct queries in a user-friendly manner, combining visual and declarative, SQL-like languages.

## 7. Conclusion and future directions

In this paper we provided an overview of data streams and the query processing that accompanies them. We discussed theoretical and concrete, implementation-related aspects of DSMSs. We showed the identified needs from exact sciences research communities who are dealing with extremely large data sets. We identified the intersection of current state-of-the-art in data stream processing with exact sciences data processing demands. We presented a preliminary guide to designing a dedicated Sci-DSMS, composed of a set of principles - SCIPE.

Future work includes three research directions. SCIPE is in its preliminary stage. We intend to develop a formalization of our Sci-DSMS oriented set of designing principles. Subsequently, we want to identify additional common requirements in exact sciences data processing, that can add value to our SCIPE set. Once these steps are performed, we intend to provide an implemented prototype of a Sci-DSMS, that can handle streams of data in exact sciences in a uniform manner.

## References

[1] D.J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul and S. Zdonik, Aurora: a new model and architecture for data stream management, *The VLDB Journal* **12** (2003), no. 2, 120–139.

[2] D.J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.H. Hwang, W. Lindner, A.S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing and S. Zdonik, The Design of the Borealis Stream Processing Engine, *Proceedings of the 2005 Conference on Innovative Data Systems Research (CIDR)* (2005).

[3] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava and J. Widom, *STREAM: The Stanford Data Stream Management System*, Technical Report, Stanford InfoLab, 2004.

[4] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, R. Motwani, I. Nishizawa, U. Srivastava, D. Thomas, R. Varma and J. Widom, STREAM: The Stanford Stream Data Manager, *IEEE Data Engineering Bulletin* **26** (2003), no. 1, 19–26.

[5] A. Arasu, S. Babu and J. Widom, The CQL Continuous Query Language: Semantic Foundations and Query Execution, *The VLDB Journal* **15** (2006), no. 2, 121–142.

[6] A. Arasu, M. Cherniack, E. Galvez, D. Maier, A.S. Maskey, E. Ryvkina, M. Stonebraker and R. Tibbetts, Linear Road: A Stream Data Management Benchmark, *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB '04)* (2004), 480–491.

[7] B. Babcock, S. Babu, M. Datar and R. Motwani, Chain: Operator Scheduling for Memory Minimization in Data Stream Systems, *Proceedings of Special Interest Group on Management of Data Conference 2003 (SIGMOD'03)* (2003), 253–264.

[8] B. Babcock, S. Babu, M. Datar, R. Motwani and D. Thomas, Operator Scheduling in Data Stream Systems, *The VLDB Journal* **13** (2004), no. 4, 333–353.

[9] B. Babcock, M. Datar and R. Motwani, Load Shedding for Aggregation Queries over Data Streams, *Proceedings of the 20th International Conference on Data Engineering (ICDE 2004)* (2004), 350–361.

[10] H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, E.F. Galvez, J. Salz, M. Stonebraker, N. Tatbul, R. Tibbetts and S.B. Zdonik, Retrospective on Aurora, *The VLDB Journal* **13** (2004), no. 4, 370–383.

[11] J. Becla, K.T. Lim and D.L. Wang, Report From The 3rd Workshop On Extremely Large Databases, *Data Science Journal* **9** (2010), 1–16.

[12] D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul and S. Zdonik, Monitoring Streams - A New Class of Data Management Applications, *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB '02)* (2002), 215–226.

[13] D. Carney, U. Cetintemel, A. Rasin, S. Zdonik, M. Cherniack and M. Stonebraker, Operator Scheduling in a Data Stream Manager, *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB '03)* (2003), 838–849.

[14] S. Chandrasekaran, O. Cooper, A. Deshpande, M.J. Franklin, J.M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss and M. Shah, TelegraphCQ: Continuous Dataflow Processing for an Uncertain World, *Proceedings of the 2003 Conference on Innovative Data Systems Research (CIDR)* (2003).

[15] J. Chen, D.J. DeWitt, F. Tian and Y. Wang, NiagaraCQ: A Scalable Continuous Query System for Internet Databases, *Proceedings of Special Interest Group on Management of Data Conference 2000 (SIGMOD'00)* (2000), 379–390.

[16] M. Hilbert and P. Lopez, The World's Technological Capacity to Store, Communicate, and Compute Information, *Science Express* **332** (2011), no. 6025, 60–65.

[17] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein and R. Varma, Query Processing, Resource Management, and Approximation in a Data Stream Management System, *Proceedings of the 2003 Conference on Innovative Data Systems Research (CIDR)* (2003).

[18] E. Ryvkina, A.S. Maskey, M. Cherniack and S. Zdonik, Revision Processing in a Stream Processing Engine: A High-Level Design, *Proceedings of the 22nd International Conference on Data Engineering (ICDE 2006)* (2006).

[19] S. Schmidt, *Quality-of-Service-Aware Data Stream Processing*, Dresden University of Technology, Department of Computer Science, Ph.D. Thesis, 2007.

[20] N. Tatbul and S. Zdonik, Window-aware Load Shedding for Aggregation Queries over Data Streams, *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB '06)* (2006), 799–810.

[21] N. Tatbul, QoS-Driven Load Shedding on Data Streams, *EDBT '02 Proceedings of the Workshops XMLDM, MDDE, and YRWS on XML-Based Data Management and Multimedia Engineering-Revised Papers* (2002), 566–576.

[22] N. Tatbul, U. Cetintemel, S. Zdonik, M. Cherniack and M. Stonebraker, Load Shedding in a Data Stream Manager, *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB '03)* (2003), 309–320.

[23] S.B. Zdonik, M. Stonebraker, M. Cherniack, U. Cetintemel, M. Balazinska and H. Balakrishnan, The Aurora and Medusa Projects, *IEEE Data Engineering Bulletin* **26** (2003), no. 1, 3–10.

[24] Congestion Pricing: A Primer, URL: `http://www.ops.fhwa.dot.gov/publications/congestionpricing/congestionpricing.pdf`.

[25] XLDB, URL: http://www-conf.slac.stanford.edu/xldb/.

(Sabina Surdu) Faculty of Mathematics and Computer Science, Babes-Bolyai University, 1 Mihail Kogalniceanu Street, Cluj-Napoca, RO-400084, Romania
*E-mail address*: `surdusabina@yahoo.com`