

Conceptual Modelling of Computations On Data Streams

Anita Dani

University Of Wollongong In Dubai,
Dubai, U.A.E.

AnitaDani@uowdubai.ac.ae

Janusz Getta

University Of Wollongong, Wollongong, Australia
jrg@uow.edu.au

Abstract :

This paper proposes a new symbolic language for the conceptual modelling of computations on data streams. We consider a class of algorithms related to the evaluation of mathematical operators on data streams. A vector model is defined to represent the sliding windows. A graph abstraction is used to model the algorithms. The notation is general enough to be used for visualisation and optimisation of a wide class of data stream processing applications.

1 Introduction

A data stream is a continuous flow of data from a source to a destination. We can think of a data stream as an unlimited sequence of tuples D_1, D_2, \dots arriving at the regular or irregular intervals. The rate of incoming data can be very high, variable, and unpredictable. As discussed in B. Babcock et al (2002) the major applications of data stream processing systems include the analysis of phone calls by callers to dialed parties, analysis of data flow in the computer networks, analysis of data obtained from the sensor networks, e.g. changes of temperature at a weather station, and variety of financial applications.

Data stream applications are in many aspects different from traditional database applications. The most important characteristic of data stream is that the underlying data are changing while the user applications are static and continuously repeated. Even though traditional databases can manage large volumes of data set, they are unable to effectively process the intensive sequences of updates. This is due to the high arrival frequencies and irregular intervals of data streams. In the applications where the real time responses are of the prime importance, e.g. in stock market applications, speed of processing is a vital issue. In the applications where interpretation of data is mostly qualitative,

accuracy of the answers is less important than the processing speed.

Even a stream is defined as an infinite sequence of data items, the computations are always performed on a finite subset of the stream. A concept of *window* is the most popular data structure in data streams processing. It captures a finite and relevant to the present time slot subset of an infinite stream. A timestamp associated with each tuple in a stream allows us to broadly categorise windows as time windows and data windows. Time window is the most suitable model for applications where the recent data are significant. As presented by Koudas N. and Guha S. (2003), a time window can be agglomerative, sliding or shifting window. Due to the frequent changes in underlying dataset, simple computation such as average becomes a blocking operation on a sliding window.

A window over a data stream can be treated as a sequence, a list, or a set of elements. This work introduces a vector model and graph abstraction for the algorithms on sliding windows. This is based on a similar concept presented by Chen G. and Kotz D.(2002). Our approach facilitates the study of algorithms at more generic level. Different mining applications apply different mathematical operators on sliding windows. We present a short summary of operators, which are commonly used and their applications in the next section. Evaluation of these operators are straightforward and do not need any special data structures. Efficiency of an algorithm on a sliding window is always related to the number of passes through the window elements. It is important to develop the algorithms that can trade off between the memory requirements and the running time and at the same time produce exact answers. The lower-level operators introduced in the paper can model any typical mathematical operator used for mining. It is possible to detect, which values can be stored and re-used from the properties of the mathematical operator.

An algorithm can be considered as *incremental* or *windowed algorithm* if it avoids re-processing of the current elements within the window. The paper provides the guidelines for verification whether an algorithm is incremental. If an algorithm is not incremental we show how to optimise it and how to estimate the amount of extra memory resources required to make the algorithm incremental.

Section 2 includes a brief summary of the earlier research on data streams. Definitions, notation and symbols are

provided in section 3. Summary of mathematical operators on a data stream and a motivating example are presented in section 4. Section 5 describes the vector model of a window and the lower-level operators on the window. Section 6 illustrates model of a data stream operator using the notation of section 5. Section 7 includes the criteria that determine incremental nature of the algorithm and rules to optimise the algorithm. Section 8 is about conclusion and future direction.

2 Related work

A comparative study of the issues in databases and data streams is presented by Babcock (2002) et al. The complex and continuous queries on data streams with the limited resources promote the research on efficient query processing. L.Qiao et al(2003) and Koudas and Guha (2001) developed single pass or incremental algorithms to evaluate aggregate queries over data streams. Study of space requirements for the single and multiple pass algorithms is presented by Rauch ,Heninger, Raghavan and Rajgopalan (1998). Business and financial related applications require the efficient algorithms for joining data streams and joining a window on a stream with a relational table. Das et al (2003) and Golab and Tamer (2003) developed a new algorithm to join data streams. Datar et al (2002) proposed the efficient algorithms to compute the stream statistics over the sliding windows of bits. The analysis of memory requirements for Min/Max algorithms on sliding windows is presented by Qiao , Chen, Li, Agarwal and Abbadi (2003). Arasu et al[8] presented a class of queries over multiple data streams which can be computed using bounded memory. In the presence of efficient query processing algorithms, the next challenge is to build efficient query plans. Telegraph CQ group (S. Krishnamurthy et al 2003) considered optimisation of query processing. Research groups such as STREAM (S.Group 2003), Aurora (Zdonik S. et al (2003), and Telegraph CQ (S. Krishnamurthy et al 2003) developed the prototypes of complete Data Stream Management Systems. Comparative study of these works is presented by Koudas and Srivastava (2003). The use of punctuations in processing block operators is presented by Tucker et al (2002).

The Aurora system uses the box and arrow diagrams to indicate the primitive operators and data flow. Our definition of a primitive operator is slightly different from the one introduced in Aurora. We distinguish two types of algorithm based on the characteristic of these operators. Even though Avg and Max are the aggregate operators, we exploit the difference between their evaluations. Avg is evaluated by *processing* the values whereas Max is evaluated by *searching* for a value. Our lower-level operators can model these distinctly.

The research done so far has been at ad-hoc level. We have studied some of the mathematical operators on data streams, which are used for mining purpose. Our research is focused on a visual notation, which can model all such operators in a consistent manner. The graph abstraction can reveal the details at a granular level and at the same time can detect the common characteristics of the algorithm. This will help to build semantics for a class of operators on sliding window.

3 Definitions

Data stream is an infinite sequence of tuples where each tuple has a timestamp automatically added at the recording time. *Window* is a finite and well defined sub-sequence of a data stream. *Sliding window* is the mechanism of forming overlapping sub-sequences of tuples, which are relevant to the application, at the pre-determined instances. *Time window* is the window of tuples such that time-stamp of each tuple lies within a given interval of time. The number of tuples within the window is fixed if the rate of data arrival is constant. In general, the number of tuples within the time-window may not be fixed. *Data window* is a fixed size window of tuples. The window slides after it fills up to its capacity. If the rate of arrival is constant, then the window slides at the regular intervals.

4 Motivating Example

We summarize here some of the common mathematical operators.

Operator	Mathematical expression	Area of application
L_p norm (p finite)	$(\sum x_i^p)^{1/p}$	Digital Signal Processing, clustering applications
L_∞	Max x_i (i=1,2..n)	Statistics, financial applications
Weighted Average	$\sum k_i x_i$	Time series, financial applications
Discrete Fourier Transform	$\sum x_i e^{-2j\pi k / n}$	Digital Signal Processing, Image processing
Time Decaying sums	$V_g(T) = \sum f(t) g(T-t)$	IP routing - RED protocol, Internet gateway selection, Usage statistics of phone customers

Table 1: Summary of Mathematical operators on Data Streams

Whenever the new elements are appended to the window and the oldest elements expire the operator should re-evaluate itself on the new window. We say that evaluation of an operator is *blocking* if its computations always require the access to all elements in a window. The evaluation of an operator is *incremental* if its computations reuse the result of the most recent evaluation of the operations, the tuples removed and added to the window. The operators like L_p norm for finite p can be evaluated incrementally whereas the operators like weighted average cannot be evaluated incrementally in a general case. The operators like Max or Min can be evaluated incrementally for certain values of new data elements. The graphical notation explained in the next section overview a class of incremental operators.

5 Conceptual model

5.1 Modelling a window as a vector

We define an instance of a sliding window as a vector $X = \langle x_1, x_2, x_3, \dots, x_n \rangle$ obtained from the stream of tuples $\langle x_i, t_i \rangle$ ordered by a time stamp attribute t_i .

The vector is completely described by the following three attributes-

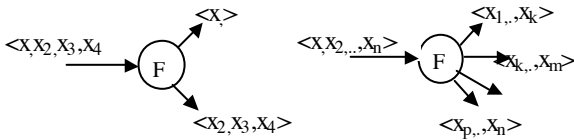
- Dimension of the vector (Number of elements in the window)
- Order of the elements within the window
- Values of the elements within the window.

5.2 Lower-level operators

This section introduces a concept of *lower-level operators* on a window modelled as a vector of values. The lower-level operators change the values of the attributes listed above. The actions associated with each of the lower-level operators are always triggered by the arrival of a new vector on one of the inputs. We categorize the lower-level operators on the basis of the attributes changed by the operators

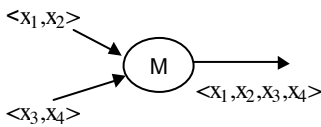
5.2.1 The operators that change dimensions

Filter (denoted by F)



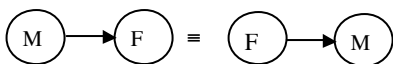
In general, a filter splits a given vector of dimension n into a finite number of sub-vectors of smaller dimensions. The operator F allows for the replication of elements within the sub-vectors.

Merge (denoted by M)



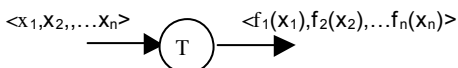
The operator merges the elements from two vectors to form a new vector whose dimension is equal to the sum of dimensions of the input vectors. We assume that the second vector is appended to the first one. This implies that the operator M is not commutative. Moreover, the operator M will not eliminate duplicates.

Note that



5.2.2 The operators that change values

Transform (denoted by T)



Where $f_i : \mathcal{R} \rightarrow \mathcal{R}$ is some mathematical function $\forall i = 1, 2, \dots, n$. In general $f_i \neq f_j$.

Note that the functions operate individually on every element of the vector. The transform operator operating on a vector of dimension n will be a vector of dimension n such as $\langle f_1, f_2, f_3, \dots, f_n \rangle$ at all instances.

The above representation is well defined when at least one of the following conditions is satisfied. The dimension of the transform vector is greater than or equal to the dimension of the window vector.

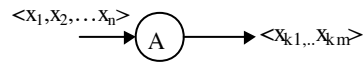
When the dimension of the window vector is unpredictable, the transform vector can be augmented through known definition. This is possible if it is known that $f_{j+k} = f_j \forall k = 1, 2, 3, \dots$

We call a vector as *uniform* if $f_i = f_j$ for all i and j . The transform vector operating on fixed size window may or may not be uniform. But *non-uniform* vectors cannot operate on variable sized window.

The variable sized windows are appropriate when the sliding of window is an instance based window. There can be multiple inputs at the same instance. Non-uniform definition for f_i 's leads to ambiguity. In some cases, the dimension of the data vector may be more than the dimension of the transform vector. This may result into missing values in the output vector.

Aggregate (denoted by A)

Aggregate operator outputs the aggregate of the x_i 's when applied to a vector $\langle x_1, x_2, x_3, \dots, x_n \rangle$. This operator finds or searches for a value or a set of values satisfying the given condition. For example, an algorithm that finds the maximum value in a window or an algorithm that finds first even number in the window belongs to a class of aggregate operators.

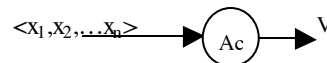


$P(x_{ki})$ is true for all ki 's for given

Note that A will not output aggregate as sum or avg, for which we are introducing another operator.

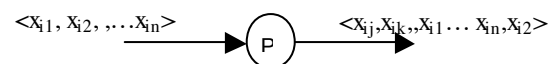
Accumulator (denoted by Ac)

Accumulator computes $V = \oplus x_i$ where \oplus is a mathematical operator on all arguments x_i . The operator always outputs a single value computed from the contents of a given vector of dimension n . For example, if \oplus represents the sum of all x_i 's, then $V = \sum x_i \forall i = 1, 2, \dots, n$



5.2.3 The operator that changes positions

Permutation (denoted by P)



This operator will change the position of elements within the vector.

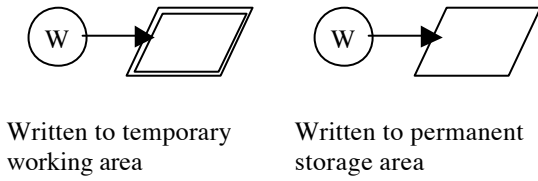
5.2.4 The operators required for streaming

Reader (denoted by S)

Let S be the source of tuples. Reading of a new element is indicated as the outgoing branch from S. It does not have an input. Reading of a new element triggers a slide of a window and the following operations. Presence of the distinct sources indicates a distributed environment.

Writer (denoted by W)

Writer operator is needed in order to store the temporary results and reuse them on sliding windows applications. The operator produces no new outputs. The result of the previous operation may be written to working area, intermediate storage area or permanent storage. We make this distinction in the view of the architecture proposed by Babu S. and Widom J. (2001).



6 Modelling stream operators using lower-level operators

The operators listed in the section 3 can be modelled by a directed graph where the lower-level operators are represented as nodes and arrows indicate the flows of data.

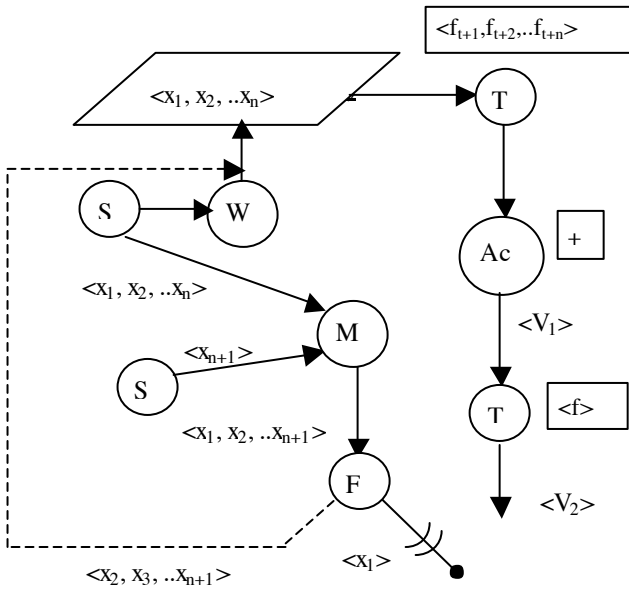


Fig (1)

The graph in Fig (1) can model all operators evaluated in the following way. Let $X = \langle x_1, x_2, x_3, \dots, x_n \rangle$ be a window observed at the instance $t=0$, Vector X is passed through the Transform operator and then the output vector $\langle f_1(x_1), f_2(x_2), \dots, f_n(x_n) \rangle$ is passed through the accumulator. The output of the accumulation operator is a single dimensional vector which is transformed to produce the final result at instance $t=0$. At the next instance $t=1$,

element x_{n+1} enters the window. This triggers the Merge operation of $\langle x_{n+1} \rangle$ with X. The window is refreshed through the Filter operator, which removes the oldest element from the window. The resulting window vector has same dimension as the original window vector.

The operations of T, Ac and T are repeated on this vector. At every instance when the window moves, the operator T operates on a vector of dimension n. The algorithm abstracted by the Fig(1) is not efficient (or optimised). At every repetition, the operator Ac (+) in this case, takes input vector of dimension n. Even though only two values within the vector have changed, the algorithm reads all n values including the n-1 values, which are not changed. This is due to the fact that the result of every iteration is not recorded. The result of each iteration can be stored and reused for next computation. This avoids re-reading of elements at every movement. At every movement, two sub-vectors have impact on the next result. One is the sub-vector that enters the window, and the other is the sub-vector that leaves the window. Note here that both these vectors will have same dimension.

Impact of the sub-vector that leaves the window is negative whereas that of the other sub-vector is positive. In order to have incremental algorithm, it is required to undo the contribution of the sub-vector that is leaving the window. The process of 'undoing' the effect can be achieved if we can revert the operation of Accumulator. Another important factor is the definition of transformation vector. If the transform vector is uniform then the impact of the new sub-vector can be computed. After initialisation, for each successive movement we can provide the information of the previous result, the new and the old sub-vector. The operator Ac will operate on a sub-vector formed by $\langle T(x_1), T(x_{n+1}), V \rangle$

Let $W_0 = \langle x_1, x_2, \dots, x_n \rangle$ be the window observed at instance $t=0$ and $W_1 = \langle x_2, x_3, \dots, x_{n+1} \rangle$ be the window observed at the next instance $t=1$. We present here two operators on the window at two successive instances.

Example 1: Let $\theta = \sum x_i^2$ $\theta(W_0) = x_1^2 + \dots + x_n^2$ and $\theta(W_1) = x_2^2 + \dots + x_{n+1}^2 = \theta(W_0) - x_1^2 + x_{n+1}^2$

Evaluation of the operator requires squaring of n numbers and adding them at every movement of the window. For window of size n, it requires n multiplication and 1 addition. When the window moves, only element leaves the window and one element enters the window. Remaining n-1 elements have not changed, but the operator requires squaring of these numbers. $\theta(W_1)$ can be computed without processing the elements x_2, \dots, x_n , which have not expired from the window as

$$\theta(W_1) = \theta(W_0) - x_1^2 + x_{n+1}^2$$

This requires squaring of only two numbers, x_1 (element leaving the window) and x_{n+1} (element entering into the window). Thus the evaluation of the operator is optimised by storing and re-using the previous results.

Note here : The transformation vector $\langle f_1, \dots, f_n \rangle$ is uniform where $f_i(x) = x^2$ for every i.

Example 2 : Weighted average $\sum k_i x_j$ for every window element x_j . The elements within the window are given weights related to their position.

In this case $f_i(x) = k_i \cdot x$.

$\theta(W_0) = k_1x_1 + k_2x_2 + \dots + k_nx_n$ $\theta(W_1) = k_1x_2 + k_2x_3 + \dots + k_nx_{n+1}$
 In general, $k_i \neq k_j$ then computation is not incremental.

7 From non-incremental to incremental

To summarize, we can say here that given any algorithm on a sliding window, it is possible to implement it incrementally, if

- The transformation applied on the vector at any instance is uniform for every element of the vector.
- The process of accumulation is reversible.

When the algorithm is incremental we can perform the following steps in order to optimise the graph.

1. Store the results of initial window (basic window) in the form of a vector. Let this be the first instance t_0 .
2. At the next instance the window slides as a set of new elements arrive. For the sake of simplicity, let us assume that only one element enters the window. At this stage, following actions will be performed in parallel.

(2A) After sliding of the window, two vectors are filtered out; one consists of the oldest element from the window. In this case, it is $\langle x_1 \rangle$. The second vector is the vector after removing x_1 from the original vector. In this case it is $\langle x_2, x_3, \dots, x_n \rangle$. Pass the vector $\langle x_1 \rangle$ through the operator T.

(2B) Merge the new element x_{n+1} with vector $\langle x_2, x_3, \dots, x_n \rangle$. The resultant vector will replace the old vector after passing through the W operator. Vectors $\langle x_{n+1} \rangle$ will pass through the operator T.

3. The output of 2(A) will pass through another operator T, which is inverse operation of Ac.
4. The output of (3) and (2C) will be merged with the contents of W from step (1).
5. Output of (4) will pass through the operator Ac.

Steps (1) to (5) will form a loop, which will be repeated for each movement of the window. This is represented by --- on the graph. (Fig (2))

8 Conclusion and future direction

Low-level operators are identified and symbolic notation is introduced. The symbolic notation can represent all operators on data streams as well as single user and distributed environment. The graph abstraction can be used to build conceptual model for computation of mathematical operators on data stream, in particular to suit the sliding window model. Sufficient conditions are established to check if the operator can be evaluated without re-processing the un-expired elements.

The operators, which require accumulation through arithmetic operation can be made incremental. Since the evaluation is independent of data values. If the graph consists of the lower level operator aggregate, A, then it is not possible to optimise the graph completely, unless the evaluation is supported by additional memory, since the operation of evaluating aggregates is not reversible. Moreover the evaluation of these operators is data dependent unlike the accumulator. To make the

evaluation of these operators efficient we have to provide additional memory resources such as storing sorted list of values as it is presented by Qiao et al (2003).

As extension to the current work, we are going explore aggregate operators and operators on two or more data streams.

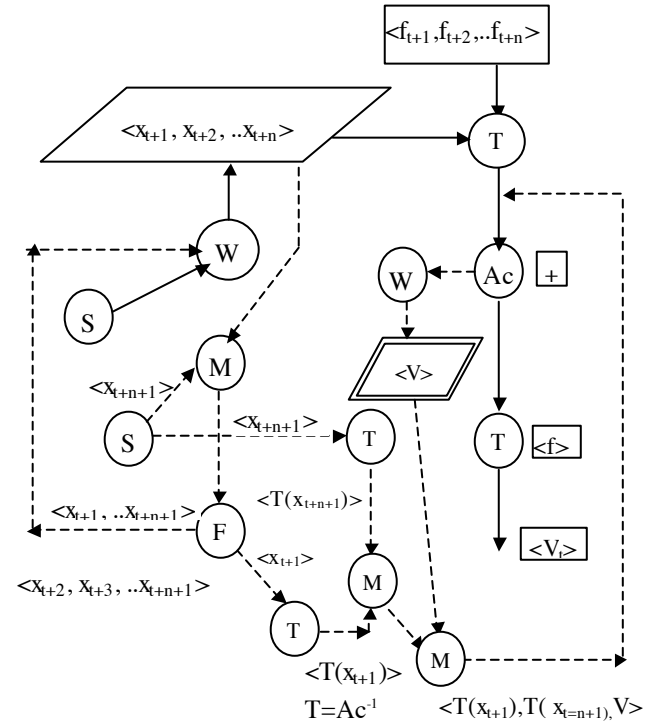


Fig (2) ¹

9 References

Arasu, Babcock, B., Babu, S., McAlister, J. and Widom, J. (2002): Characterizing Memory Requirements for Queries over Continuous Data Streams. *Proc. ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, Madison, Wisconsin, USA, 221-232, ACM Press.

Babcock, B., Babu, S., Datar, M., Motwani, R. and Widom, J. (2002): Models and Issues in Data Stream Systems. *Proc of the twenty-first ACM SIGMOD-SIGACT-SIGART*, Madison, Wisconsin, USA, 1-16, ACM Press.

Babu S. and Widom J.(2001): Continuous Queries Over Data Streams, *ACM SIGMOD Record archive* Volume 30, Issue 3, 109-120, ACM Press.

Chen, G. and Kotz, D. (2002): Context Aggregation and Dissemination in Ubiquitous Computing Systems, *Proc. Fourth IEEE Workshop on Mobile Computing Systems and Applications*, Callicoon, New York, 105-114, IEEE Computer Society Press.

- : Permanent storage,
- : Working storage
- : Initial Window
- : Successive instances

- Cranor, C., Johnson, T., Spatscheck, O. and Shkapenyuk V. (2003): The Gigascope Stream Database, *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 27-32, IEEE Computer Society Press.
- Das, A., Gehrke J. and Riedwald M. (2003): Approximate Join Processing Over Data Streams. *Proc. ACM SIGMOD International Conference of Management of Data*, San Diego, California, USA, 40-51, ACM Press.
- Datar, M., Gionis, A., Indyk, P. and Motwani, R. (2002): Maintaining Stream Statistics Over Sliding Windows. *Proc. Of 13th Annual ACM-SIAM symposium on Discrete Algorithms*, San Francisco, USA, 645-644, Society for Industrial and Applied Mathematics.
- Golab, L. and Tamer, M. (2003): Processing Sliding Window Multi-Joins in Continuous Queries over Data Streams *Proceeding of the 29th VLDB Conference*, Berlin, Germany, available at <http://db.uwaterloo.ca/~ddbms/publications/stream/vldb03.pdf> Accessed 12/11/04.
- Guha S. and Koudas N. (2001): Approximating a Data Stream for Querying and Estimation Algorithms and Performance Evaluation. *Proc. 18th International Conference on Data Engineering (ICDE'02)*, San Jose, CA, USA, 567-576, IEEE Computer Society.
- Koudas N. and Srivastava D. (2003): Data Stream Query Processing, *Proceedings of 29th International Conference on Very Large Data Bases VLDB* Berlin, Germany, available at <http://www.vldb.informatik.hu-berlin.de/ressources/vldb-2003-Tutorial T3.pdf> Accessed 12/11/04.
- Krishnamurthy S., Chandrasekaran S., Cooper O., Deshpande A., Franklin M., Hellerstein J.M., Hing W., Maden S., Reiss F. and Shah M. (2003): Telegraph CQ An Architectural Status Report. *IEEE Data Engineering Bulletin* 26.
- Qiao, L., Chen, S., Li, HG., Agarwal, D. and Abbadi, E.E. (2003): Efficient Computation for Max/Min Queries on Sliding Windows. *Unpublished Manuscript*.
- Rauch, M., Henzinger, Raghavan, P. and Rajgopalan, S. (1998): Computing on Data Streams. Available at <http://gatekeeper.research.compaq.com/pub/DEC/SRC/technical-notes/SRC-1998-011.pdf> Accessed 10/11/04.
- Stream Group (2003): STREAM The Stanford Stream Data Manager, *IEEE Data Engineering Bulletin* 26.
- Tucker P., Maier D., Sheard T., and Fegaras L. (2002): Enhancing relational operators for querying over punctuated data streams. manuscript, Available at <http://www.cse.ogi.edu/dot/niagara/pstream/punctuating.pdf>. Accessed 12/11/04.
- Zdonik S., Stonebreaker M., Cherniack M., Cetintemel U., Balazinska M., and Balakrishnan H. (2003): The Aurora And Medusa Projects, *Bulletin of the Technical Committee on Data Engineering*, IEEE Computer Society.