2006

A system of operations on sliding windows

Anita Dani
University of Wollongong in Dubai

Janusz Getta
University of Wollongong, jrg@uow.edu.au

A System Of Operations On Sliding Windows

Anita Dani [1], Janusz Getta [2]
[1] *College Of IT, University Of Wollongong in Dubai*
U.A.E., AnitaDani@uowdubai.ac.ae
[2] *School of Computer Science and Software Engineering,University Of Wollongong*
Australia, jrg@uow.edu.au

Abstract

One of the distinguishing characteristics of a data stream system is 'a continuous query operating on dynamic data' as opposed to 'static data processed by instantaneous query' in DBMS.

In this paper, we propose a system of operators to process a continuous query on sliding windows over streaming data. We use the pre-defined notion of sliding windows over a data stream and provide an abstraction for a system of operations that supports evaluation of a continuous query on sliding windows. The model consists of a basic data window, a set of transition operations, a set of processing operations and the operation of binding number of streams together before processing.

1. Introduction

Data streams can be observed in various fields such as network monitoring and sensor-based monitoring [1]. Applications of advanced sensor and radio frequency identifier (RFIDs) technologies require implementation of specialized software for processing long sequences of data over long periods of time. This kind of software system is commonly known as data stream processing system [3] or sensor database system [4].

An interesting research problem is the construction of a universal system of operations to process continuous queries on sliding windows over data streams similar to the system of relational algebra operations. Any system of this kind should cover all stages of data stream processing, i.e. formation of sliding windows, iterative processing of user applications, processing of many streams within one applications.

In this paper we present an abstract model for the data window and basic operations. The model consists of three layers of operations required for processing a continuous query. First layer is for data- formation, second layer is for data-binding and the third layer is for data-processing. This allows processing of multiple streams in the same way as that of a single stream. Since the model separates the layer of window formation from the layer of processing, optimization can be done in parallel as well as jointly. Our model can express processing of queries involving single or multiples streams in a consistent manner.

○

In the section 2 we present a summary of related work. Section 3 describes computations on data streams. Section 4 presents informal description of the proposed model. Section 5 presents the proposed model formally. Section 6 shows equivalence of the proposed system with the relational algebra operators. Section 7 is summary,conclusion and future direction.

2. Related Work

The prototypes of data stream processing systems implemented in the last few years either consider a data stream as a sequence of tuples [5], [6]or as a sequence of numeric values [7], [8] or even as a sequence of XML documents [9]. The type of data items in a stream determines the operations that should be implemented to process the user applications. A couple of formal models of data stream processing and operations on data streams have been proposed so far. The most common approach considers a data stream as a sequence of tuples and a single state of a sliding window over a data stream as a relational table. Then, an extended SQL can be used to express queries over data streams and extended relational algebra can be used as an implementation language [10]. Another approach proposes a system of very basic operations on single data items extracted from a data stream and single sliding window on another stream or the results of intermediate computations [13]. In this model, a user application is represented by a set of path expressions where each expression is a sequence of elementary operations and describes the processing of a single data item taken from data stream and the contents of static sliding windows over the other data streams. This model also considers a data stream as a sequence of tuples. A denotational semantics for the applications processing data streams has been proposed in [11]. This approach defines semantics for a generic continuous query language [12]. The query language considers two types of data objects: streams and relational tables and provides the operations that transform streams into tables and reverse. An interesting conceptualization of data stream processing has been proposed in [2]. This approach treats the fixed states of sliding windows as fixed size vectors and provides a system of operations on vectors in order to represent a single state of computations on data streams. Incremental computation of a typical blocking operator results in a real time evaluation of queries over sliding windows.

1-4244-0716-8/06/$20.00 ©2006 IEEE.

A sufficient condition to achieve incremental evaluation of mathematical operators is presented in [14]. This work does not consider the effect of the sliding mechanism on evaluation of a mathematical operator. A broader class of windows is introduced in [16], called as predicate-windows. In this paper, authors have defined predicate-windows using a predicate on any attribute of data items in the stream. Our model can represent the mechanism of forming a predicate-window as one of the types of window-formation mechanism. A formal framework for expressing windows in continuous queries over data streams is presented in [15]. The framework presented in this paper does not model processing of multiple streams. The model presented in [19] shows only two layers of operations, where as in this paper, we propose that the layer of data binding is equally important to process queries on multiple streams in the same way as the queries on a single stream.

In the next section we characterize basic difference between instances of a data stream and instances of a database.

3. Stream Instances and windows

A data stream is potentially infinite collection of data items. Data items arrive into the stream continuously and hence the size of the stream increases with time. It is not possible to process an infinite stream in finite period of time. Due to potentially infinite size of the stream a query is evaluated after forming finite subsets of the data stream. Each subset is called as an *instance* of the stream. Processing of data stream requires processing of infinite number of rapidly changing instances. These instances change only by insert operation and thus each instance is a superset of all instances seen before. In contrast, there are only a finite number of instances of a database. The instances in a database, are changed due to insert, update and delete operation. There is no obvious containment relation between any two instances. Each instance remains static for a longer period which allows the DBMS to create and apply techniques such as indexing to improve performance. The contents of the database instance do not change while a query is running. In order to process each stream instance, a data window must be formed from the instance of the stream before processing.

Thus processing of a query on a data stream is done in two stages, the stage of forming the data set and the stage of processing the well formed data set. In queries involving non-blocking operators data formation and data processing can be one at the same time [20]. Example of such application is: *Convert the stock prices from one currency to another for a particular stock;* Where as in a query involving some blocking operator output can not be produced until the entire input is received. In such cases, the data set must be formed before evaluating the result.

For a given query, all of the data items may not qualify for processing. Thus we form a precise subset of the stream instance, which is called as the *window*. At any given time-instance t, the window is formed by applying some predicate on the stream instance observed at t. The window definition is included in the where clause of the query. In case of time-window, the predicate is based on the time-stamp attribute, in case of data-window, this predicate is based on the count of data items in the window. We enhance this generalization by allowing the attribute in the predicate-windows to be a function of any attribute as well as a function of the window formed from the previous stream instance. The predicate for forming a new window from the previous one is the parameter of window formation.

The movement of data items in and out of the window and computation of result on the window are modeled by different operations, which we call as low-level transformations.

4. Modeling computations

The rule of forming a new data window from an existing one is modeled by a predicate. The validity of data items at any given instance is defined by applying this predicate.

A. Three layers of computations

We identify here that any application where the query is continuous and data are dynamic, the following three steps are performed in an infinite loop:

- Process the current data set to produce the required output.
- Form a new data window from the current data window by adding and or removing certain data items.
- In case of applications involving multiple streams, there is an additional operation required. This operation binds multiple streams into a single stream.

Thus we note here three layers of processing a continuous query on sliding windows:

- *Data-formation* layer: This layer of computation is responsible for reshuffling the contents based on a certain rule.
- *Data-binding* layer: This layer is required only in the applications involving multiple streams. This operation may be applied for data formation as well as for preparing data for processing.
- *Functional* layer: This layer of computation is responsible for processing the current data set statically.

Each layer is represented by sets of certain basic transformations. In addition to the basic transformations data-formation layer is controlled by rules of adding or removing data items and predefined instances of query evaluation. We model the *functional* layer as a set of output transformations. The *data-formation* layer is modeled as a set of transition transformations. The *data-binding* layer consists of one single operation that operates on k-distinct windows and results into a single window of k-tuples. [tuple as used in mathematical terms and not as used in relational algebra terms].

1) Data-formation layer: Forming a new sliding window from the existing data set requires adding new data items and removing old data items.

The rules of removing data items can be classified into two distinct categories as *intrinsic* or *extrinsic*.

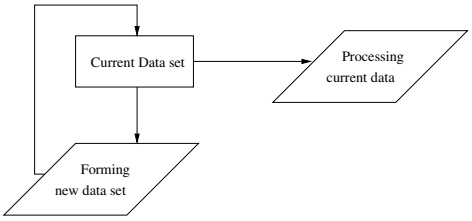

Fig. 1: Extrinsic rule of data removal

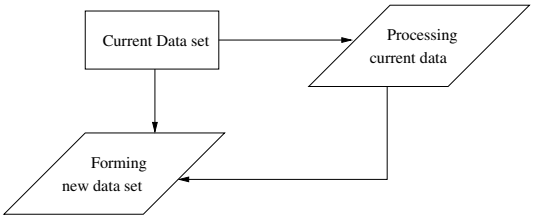

Fig. 2: Intrinsic rule of data removal

Extrinsic rules of data removal are based on a *key-qualifier* that is a property of the individual data items. Extrinsic rules do not modify the value of the key-qualifier during processing. (Fig-1) Example is removing data items with the least time-stamp. The attribute time-stamp is a property of individual data items and does not change during processing. An advantage of having extrinsic rules for data formation is that the layer of data formation can form the window required for the future computation while the current window elements are being processed.

Intrinsic rules of data removal from the current data window are based on a *key-qualifier* that is a property of the current data window. Intrinsic rules modify the value of the key-qualifier during processing. (Fig-2) Value of the qualifier such as the capacity of the window may change during processing layer of computation. The data formation layer can not form the window required for future computation until the processing of the current window is completed. Since the data items to be removed are determined after processing, incremental computation in such cases may not be possible unless supported by additional resources.

2) Data-binding Layer: This layer takes data elements from *n* different windows and forms a single window of n-tuples. Then the processing layer functions are applied on this window.

3) Functional layer: This layer is responsible for computation of query results as well as the value of the key-qualifier that determines the validity of the current data items for the future computations. The components of this layer should be able to do the processing at the data element-level as well as at the set-level.

Any system that processes continuous queries on dynamic data sets, must know when to change the contents of the old data set and when to process its contents. In the next section, we discuss this 'when' parameter of query processing.

B. Instance of processing

Any query posed at time-instance t can process the data items from the stream instance S_t, but not the data items observed at the next successive time-instances. Thus a query on a data stream must be repeated periodically in order to process all *new* data items. The window must be formed by applying the window-formation predicate on the new stream-instance.

In the next section we present the formal model.

5. FORMAL MODEL

A. Terms and Definitions

Let Ω denote the domain of the data items. A data item is denoted by x. x can be structured or atomic. When data items enter into the stream, time-stamp is recorded along with its value.

- Data Stream at instance t:
  At any point of observation, data stream instance S_t is defined as $S_t = \{(x,t)|x$ is observed at time t and t $\in [0, now())\}$ If $S_t$is a stream of values observed at t, then $S_t \supseteq S_{t-1} \supseteq S_{t-2}..... \supseteq S_1 \supseteq S_0$ If $(x_1,t_1),(x_2,t_2) \in S_t$ then $t_1, t_2 \leq$t and $(x_1,t_1) \leq (x_2,t_2)$ only if $t_1 \leq t_2$. The time-stamp attribute provides a natural index and an order relation $\leq$ for the data set. We are modeling the stream as 'append' only databases.
- Data Window at instance t:
  $W_t = \{(x,t)|x \in S_t$ and P(x) is true at t$\}$ where P(x) is the window-formation predicate. The window formation predicate is of the type $P(x,t)$ or $P(x,t,\theta(S_t))$ where $\theta(S_t)$ is a function of the stream instance S_t.
- Inductive definition of window:
  W_0 is the initial window. We define the window at instance $t > 0$ inductively as $W_t = \{(x,t_k)|(x,t_k) \in W_{t-1}, P(x,t_k,\theta(W_{t-1}))$ is true at instance $t_k\}$
- Formal Model :
  Our model is given by (W_t, $\aleph$,$\Im$,ψ)
  $\aleph$: is a set of output transformations
  $\Im$: is a set of transition transformations together with a rule of removal defined by the window formation predicate. Transition transformations are defined through set-operations union and set-difference. Output transformations are also defined by set operations. Definitions of output transformations are domain specific.
  ψ : is a binding transformation defined by the cartesian product.

B. Formal definition - output transformations

Let Λ be a collection of windows over a data stream S. Let F be the set of all functions defined on the domain Ω of data items of S.

Set of output transformations is given by $\aleph$=$\{Transform, Aggregate\}$

- Operator transform (T): Definition of Operator $Transform$ is time-dependent and its contents may change as the underlying sliding window changes.

T_t is defined as a set of functions $\{f_1, f_2, \ldots f_n\}$ where each f_i operates on (x_i, t_i) in the window.
$T_t(W_t) = \{(f_i(x_i), t) | (x_i, t_i) \in W_t$ and $f_i : \Omega \mapsto \Omega'\}$
The size of the set T_t is equal to the size of the sliding window W_t. This facilitates processing at the data-element level.

- Operator aggregate (A): Let W_t=$\{(x_1, t_1), (x_2, t_2) \ldots (x_n, t_n)\}$
We define transformation A as a function defined on Ω^n.
$A(W_t) = \Omega^n :\mapsto \Omega^k$
$A(W_t)$ is the set containing the output of the query computed at instance t.
$A(W_t) = V_t$ where V_t is a set of values obtained iteratively or recursively as follows:
$A(\{x_1\}) = V_1$ defined as the initial set.
$A(\{x_1, x_2\}) = V_1 \oplus \{x_2\}$ where $\oplus$ indicates adding, removing or replacing elements from V.
...
A$(\{x_1, x_2, \ldots x_n\}) = V_{n-1} \oplus x_n$

We demonstrate the application of output transformation by considering the following query:

Let SalesStream S be the stream of sales figures as described in [17]. The objective query is: *For SalesStream S continuously report the total sales of a set of items with prices greater than 4 in the last hour.*

Processing function is θ=$\sum\{x|(x,t) \in W_t\}$, which is modeled by the transformation operator Aggregate. Evaluation of the function $\sum$ is defined inductively as follows: V_1=$\sum\{x_1\} = x_1$; $V_2 = \sum(V_1, x_2) = \{x_1 + x_2\}$ ... $V_n = \sum(V_{n-1}, x_n)$ for $n > 1$

C. Formal definition - transition transformation

Low-level operations required for transition are expressed set-theoretically. Our model proposes an operation *Filter* for removal of data item and another operation *Merge* for addition of data elements.

$\Im$= Set of transition operations = $\{Filter, Merge\}$

- Transition operation Filter is defined by the operation of set-difference. This operation removes those elements from the current window which make the predicate P false.
Data items from the window which make the predicate P false are precisely, those data items, which expire from the current window. Some of the data items may expire temporarily where as some of them expire permanently. For example, in append-only data streams, where the validity of the data item is time-stamp based, the expired data items do not re-enter the window. In such cases, those data items can be removed from the window.
- Transition operation Merge is a binary operation defined by the operation set union. It forms the set union of the current window with the set of elements arrived after computation.

D. formal definition - binding transformation

We define an operation Zip for processing data from multiple sliding windows. Two distinct streams may have different values for key-predicate and instance of query evaluation. Thus two windows may slide at different instances or one of them may be a static window. This may result into asynchronous movement of two windows and may require multiple scanning over the data window in some situations.

$Z(W_t, W'_{t'}) = \{(x, y, t) | (x, t_{t_x}) \in W_t, (y, t_y) \in W'_{t'}\}$.

Operation Zip is defined as the Cartesian product of the two windows subject to certain condition. Operation Zip transforms operations on higher dimensions to operations on one stream of higher order. This simplifies the problem of asynchronous movement of two windows. This operation is applied in processing data items from two windows simultaneously.

We demonstrate application of zip transformation for expressing the processing function expressed in the following query.

Which pairs of stocks are correlated with a value of 0.9 over the last three hours? The user might want the answer continuously, say every second[18].

The query involves processing of values from two streams. Processing function is the correlation coefficient given by the following formula: $corr(s,r)$=$\frac{\frac{1}{w}(\sum s_i.r_i - \overline{s}.\overline{r})}{\sqrt{\sum (s_i - \overline{s})^2 . \sum (r_i - \overline{r})^2}}$

$\overline{s}$ is modeled by A(s_t) where s_t is the window on stream s at instance t. Similarly, $\overline{r}$ is modeled by A(r_t).

Operator Aggregate in both cases is evaluated as average, which is modeled as $T_t(A(s_t))$. Computation of average requires dividing the sum by the count. The division is modeled by Transform operator where $T_t = \{f | f(x) = \frac{x}{n}\}$. Computation of the expression in the denominator requires application transform operator before summation. Transform operator T on s_t is given by $\{f_i | f_i(s_i) = (s_i - \overline{s})^2, i = 1, 2 \ldots n\}$. Similar transformation will be applied on the window r_t.

Zip transformation will be applied on the two windows r_t and s_t for computing $\sum s_i.r_i$.

Z$(s_t, r_t) = \{(s_i, r_i) | s_i$ and r_i have same time-stamp value$\}$. [18]. A(T(Z(s_t, r_t))) will compute $\sum s_i.r_i$; where A is modeled by $\sum$ and T is given by $\{f_i(s_i, r_i) = s_i.r_i | i = 1, 2 \ldots n\}$. The result of this A(T(Z(s_t, r_t))) is a set of single value, on which the Transform operator is applied to obtain the result of the expression in the numerator. Transform operator in this case is T=$\{f | f(x) = \frac{x}{w} - \overline{s}.\overline{r}\}$

6. DISCUSSION

Even though we do not discuss the implementation issues at this point, we note here that each window can be mapped into a relation. The operators included in the proposed model can express all relational algebra operators as shown in Table 1.

A. Important points about the model

Operation of Filter and Merge are applicable to windows formed from instances of the same stream.

Operations of Filter and Merge are applicable only at the data formation layer. The output of relational operators Union

TABLE 1: EQUIVALENCE WITH RELATIONAL ALGEBRA OPERATORS

Relational Algebra operator	Operator(s) in the new system	Type of Operator
Selection	Aggregate	Output transformation
Projection	Transform	Output transformation
Join	Zip	Data Binding
Min, Max, any aggregate function	Aggregate	Output transformation
Union, Union All, Minus	Zip followed by Aggregate	Data binding followed by output transformation

and Minus are obtained by applying the Zip operator with appropriate condition, followed by the Aggregate operator.

The condition applied for the Zip operation will be as specified. If it is not specified then the following default definitions apply:

When 'Zip' is used for evaluating Minus operator, then the condition for zipping is equivalent to the Outer Join as defined in the extended relational algebra. All data items from first window are included allowing 'null' values for the second set. Then the operator aggregate is applied for selecting ordered pairs where the second element is null. These are the elements present in the first window but not in the second window.

When 'Zip' is used for evaluating Union or Union All, then the condition for zipping is equivalent to the Full Outer Join as defined in the extended relational algebra. Application of aggregate operator will select elements which are present in any one of the windows.

B. Application of the model

We consider here an example of searching a pre-defined pattern in a long sequence of characters. Let S be a stream of characters from domain Ω and P be a finite sequence of characters from the same domain.

Let P={y[1],y[2],. . .,y[n]}

S=$\{(x,t)|x \in \Omega$ t∈[0,now())$\}$ represents the stream of characters observed till the current instance.

$W_t = \{(x_i, t_i)|x_i$ is observed at $t_i\}$ is the window observed at t. Initial window W_0 contains exactly *n* elements. In other words, the first instance of query evaluation is the instance when *n* data elements enter into the stream. After that, whenever a new data item enters into the stream, a new window is formed and the data items are processed. Thus the instance of query evaluation is the instance of arrival of a new data item and hence is 'event' based. The element entered currently into the stream is added to the old window. At this instant the new window is formed by removing the element with the least time-stamp from the old window. Thus the rule of removal of data item is based on the time-stamp of the data item, hence it is extrinsic. Comparison of two strings can be done using some distance metric, such as Hamming Distance.

$D(W_t, P)$=$\sum$(d(x_i, y_i)) for i=1,2,. . . n, where
d(x,y)=1 if x=y

$= 0$ otherwise.

Computation of D involves data elements from two windows. Operation Zip is applied first to bind these two windows together. The result of the Zip operation is a window of *n* ordered pairs. Data elements from two windows are paired if their positions within the window are same.

Zip(W_t,P)={(x,y)| $x \in W_t$, y∈P, position of x in W_t =position of y in P }

Initial window elements are processed for checking the equality of each x[i] with y[i].

Initial instance: Elements of W_0 can be enumerated as $\{x[1], x[2], x[n]\}$ where x[i] is observed before x[i+1] for each i=1,2,..n-1.

Functional layer involves computation given by A(T(Z(W_0,P))). The Transform operator T is given by the set $\{f_i|f_i(x,y) = d(x,y)$ for each i=1,2,. . . $n\}$

Since range of the function d(x,y) is $\{0,1\}$, result of operator T on the zipped window Z(W_0,P) will be a set of n bits. Operator Aggregate in this case is given by Σ. The output V_0 is a singleton set containing the sum of n bits in the window Z(W_0,P).

At every successive instant the stream receives $n + 1^{th}$ element and the window slides by 1 element.

W_t= Merge ((W_{t-1} Filter $\{x[1]\}$), $\{x[n+1]\}$).

Keeping with the same rule of enumeration, the resulting window can be mapped to positions within the window as $\{x[1], x[2], ..x[n]\}$ where time-stamp of x[i] is less than the time-stamp of x[i+1] for each i=1,2... . .n-1.

The output of processing is a stream of output values, which are then used for determining occurrences of the pattern in the stream of observed characters.

Generic definition for the data model and for the rule of forming data set makes the model generic. Set theoretic definition of the model makes it implementable.

7. SUMMARY CONCLUSION AND FUTURE DIRECTION

We have presented here a generic system of processing a continuous query over sliding windows. The system is minimal in a sense that none of its operations can be expressed as a combination of the other operations. The operations are orthogonal i.e. the semantics of any two operations do not overlap. The system is computationally complete such that it is possible to express any computable function as a combination of its operations. The basic operations satisfy the closure property so that output of one operation can be passed as an input to another operation.

We conclude here that the proposed model can express queries over sliding windows from different domains. The main contribution of this paper is a formal model that can express a continuous query on sliding window.

We did not address the issue of time and space complexity of computations over sliding windows. Characterization of window-formation predicates and their effects on optimization will be studied in future. The concurrency issues that arise as a result of execution of more than one query with different sliding instances will also be studied in future.

REFERENCES

[1] M. Stonebraker,U.Cetinemel,S.Zdonik, "The 8 requirements of Real-Time Stream Processing", *SIGMOD Record*,vol 34,No 4, 2005, pp 42-47.

[2] A. Dani,J.R. Getta, "Incremental Computation Of Aggregate Operators Over Sliding Windows", In: *Proc. of 3rd International Conference: Sciences of Electronic,Technologies of Information and Telecommunications*, 2005,Tunisia, pp 42-47.

[3] B. Babcock,S. Babu,M. Datar,R. Motwani,J. Widom, "Models and issues in data stream", In: *Proc. of the Twenty-first ACM SIGACT-SIGMODSIGART Symposium on Principles of Database Systems*, 2002, Madison, Wisconsin, pp 1-16.

[4] A. Deshpande, S. Nath, P.B. Gibbons, S. Sesh, "Cache-and-Query for Wide Area Sensor Database Systems", In: *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, 2003, San Diego, California.

[5] A. Das, J. Gehrke,M. Riedewal, "Approximate Join Processing Over Data Streams", In: *Proc.of ACM SIGMOD Intl. Conf. on Management of Data*, 2003,San Diego, California.

[6] E. Vossough, "A System for Processing Continuous Queries over Infinite Data Streams", In: *Intl. Conference on Database and Expert Systems Applications*, 2004,Zaragoza, Spain.

[7] A. Arasu,B. Babcock,S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosensstein, J. Widom, "Stream: The Stanford Stream Data Manager (demonstration description)", In: *ACM SIGMOD Intl. Conf. on Management of Data* , 2003,San Diego, California.

[8] D. Abadi,D. Carney, U. Cetintemel, M. Cherniack, C. Convey, C. Erwin, E. Galvez, M. Hatoun, A. Maskey, A. Rasin, A. Singer, M. Stonebraker, N. Tatbul, Y. Xing, R. Yan, S.Zdonik, "Aurora: A Data Stream Management System", In: *ACM SIGMOD Intl. Conf. on Management of Data* , 2003,San Diego, California.

[9] Z. Bar-Yossef, M. Fontoura, V. Josifovski, "On the Memory Requirements of XPath Evaluation over XML Streams", In: *ACM SIGPODS Intl. Symposium on Principles of Database*, 2004, Paris, France.

[10] C. Luo, H. Thakkar, H. Wang, C. Zaniolo, "A Native Extension of SQL for Mining Data Streams", In: *ACM SIGMOD Intl. Conf. on Management of Data*, 2005,Baltimore, Maryland, pp 662-662.

[11] A. Arasu, J. Widom, "A Denotational Semantics for Continuous Queries over Streams and Relations", *SIGMOD Record*, Vol 33, No 3, 2004,pp 6-11.

[12] A. Arasu, S. Babu, J. Widom, "The CQL Continuous Query Language: Semantic Foundations and Query Execution", Stanford University,2003,October,*http://dbpubs.stanford.edu/pub/2003-67¿* .

[13] J.R. Getta,E. Vossough, "Optimization of Data Stream Processing", *SIGMOD Record*,2004,Vol 3, No 3.

[14] A. Dani, J.R. Getta, "Computation Of Mathematical Operators On Sliding Windows", In: *Proceedings of the 5th International Conference on Recent Advances in Soft Computing*, 2004,Nottingham, United Kingdom.

[15] K. Patraoumpas,T. Sellis, "Window Specification over Data Streams", *http://sipl72.si.ehu.es/ICSNW/CP/CR-109.pdf*,2006.

[16] T.M.Ghanem, W.G.Aref,A.K.Elmagarmid, "Exploiting Predicate-Window Queries in Data Stream Management Systems", *SIGMOD Record*,2006,Vol 35 No 3.

[17] M.A.Hammad, W.G.Aref, M.J.franklin, M.F.Mokbel, A.K.Elmagarmid, "Efficient Execution of Sliding Window-Queries Over Data Stream", *http://www.cs.purdue.edu/homes/aref/papers/StreamQueryProcessing-TechReport2003.pdf*, 2003.

[18] Y.Zhu, S.Sasha, "Fast Approaches to Simple Problems in Financial Time Series Streams", In: *Proc. Workshop on Management and Processing of Data Streams*, 2003, San Diego, CA.

[19] A. Dani, J.Getta, "Modeling Evaluation of Continuous Queries on Sliding Windows", To appear in: *Proc. IEEE International Workshop on Mining Evolving and Streaming Data in conjunction with ICDM*, 2006, Hongkong.

[20] Y.N.Law, H.Wang, C.Zaniolo, "Query Language and Data Models for Database Sequences and Data Streams", In: 30^{th} *VLDB Conference*, Toronto, Canada, 2004.