18-12-2006

Modeling evaluation of continuous queries on sliding windows

Anita Dani
University of Wollongong in Dubai

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

Follow this and additional works at: <https://ro.uow.edu.au/infopapers>

Part of the [Physical Sciences and Mathematics Commons](#)

Recommended Citation

Dani, Anita and Getta, Janusz: Modeling evaluation of continuous queries on sliding windows 2006.
<https://ro.uow.edu.au/infopapers/541>

Modeling evaluation of continuous queries on sliding windows

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. Due to potentially infinite size of the stream a query is evaluated after forming finite subsets of the data stream. The sliding window model is the most suitable model for processing finite subsets of a data stream where only recent data items qualify. In this paper, we propose a mathematical model to express a continuous query on sliding windows. This model expresses a window as an ordered set. Its sliding mechanism is expressed through a set of transition operations and query evaluation is expressed through a set of output operations.

Disciplines

Physical Sciences and Mathematics

Publication Details

This paper was originally published as: Dani, A & Getta, J, Modeling evaluation of continuous queries on sliding windows, Sixth IEEE International Conference on Data Mining Workshops 2006 (ICDM Workshops 2006), Hong Kong, China, 18-22 December 2006, 632-637. Copyright IEEE 2006.

Modeling Evaluation Of Continuous Queries on Sliding Windows

Anita Dani

University Of Wollongong in Dubai
United Arab Emirates
AnitaDani@uowdubai.ac.ae

Janusz Getta

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. Due to potentially infinite size of the stream a query is evaluated after forming finite subsets of the data stream. The sliding window model is the most suitable model for processing finite subsets of a data stream where only recent data items qualify.

In this paper, we propose a mathematical model to express a continuous query on sliding windows. This model expresses a window as an ordered set. Its sliding mechanism is expressed through a set of transition operations and query evaluation is expressed through a set of output operations.

1. Introduction

A data stream is a continuous flow of data items from its source to the processing application. Data items may arrive at high speeds and at irregular intervals. Data streams can be observed in various fields such as network monitoring and sensor-based monitoring [12].

An interesting research problem is the *construction of a universal system of operations on sliding windows over data streams* similar to the system of relational algebra operations of relational tables. 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.

A unified system of operations on sliding windows may have the same positive effect on the software architectures and implementations of future data stream processing systems as in the past a system of relational algebra operations had on the implementation of relational database management systems.

In this paper we present an abstraction for the data model and basic operations. Our model can express queries involv-

ing single or multiple streams in a consistent manner. Based on the abstraction provided, it is possible to form windows on different domains in different ways and at different moments.

In the section 2 we present a summary of related work. Section 3 describes computations on data streams. Section 4 presents the proposed model formally. Section 5 shows an application of various constructs of the model on a simple pattern matching example. Section 6 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 [7, 14] or as a sequence of numeric values [2, 1] or even as a sequence of XML documents [5]. The type of data items in a stream determines the operations that should be implemented to process 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 [9].

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 [8]. 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 [4]. This approach defines semantics for a generic continuous query language [3]. This 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 [6]. 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. 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 [13], called as predicate-windows. In this paper, authors have defined predicate-windows using a predicate on any attribute 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 [11]. The framework presented in this paper does not model processing of multiple streams.

In the next section we present the basic operations required to perform computations in order to evaluate a long running query on transient data.

3 Computations on data streams

Data items arrive into a stream continuously and hence the size of the stream increases with 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 a *window*. In order to process a continuous query, a data window must be formed before processing. Thus processing of a continuous 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*, which we call as a window.

3.1 Example queries

In this section we present some example queries and use them to capture entities and parameters of our model. Each of these queries specify the objective function and a condition that specify which data items qualify for processing. The condition is expressed as predicate which is applied repeatedly on the stream instance. The truth set of this predicate changes at every application. We identify the parameter of the predicate involved in forming the window in each case.

1. Let SalesStream S be the stream of sales figures as described in [10]. 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*."
2. The objective is to find the 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.[15]

3. S is the stream of auction-bids. The objective is to find the number of bids for each auction site over the *past 1000 tuples*. The result must be updated after every new tuple is added.
4. S is a stream of temperature values recorded by different sensors. The objective is to report the sensor identifiers for sensors that have *temperature greater than 90*. Report the modification in the answer query every two minutes.[13]

In the first and the second query the window formation predicate is based on the time-stamp of individual tuple. In the third query along with the time-stamp of the individual tuples, the count of the tuples in the window is compared with a constant. The predicate involves a function of the previous state of the window. The fourth query is a general predicate window query based on a single attribute temperature.

At any given time-instance a sliding window is obtained from the sliding window processed at the previous instance. Thus the definition of the sliding window is inductive and time-dependent. The predicate for forming a new window from the previous one is the parameter of window formation.

A general purpose system must be able to model movement of data items in and out of the window and computation of result on the window. In the next section we present different low-level operators to model basic requirements of query processing on sliding window.

3.2 Two layers of computations

Any application where the query is continuous and data are dynamic, the following two 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.

Given below are the two layers of processing a continuous query on sliding windows:

- *Functional* layer: This layer of computation is responsible for processing the current data set statically.
- *Data-formation* layer: This layer of computation is responsible for reshuffling the contents based on a certain rule and binding data items from multiple streams into one stream

Figure 1. Extrinsic rule of data removal.

Each layer is represented by a set of certain basic transformations. In addition to the basic transformations data-formation layer is controlled by rules of adding or removing data items at predefined instances of query evaluation. In case of applications involving multiple streams, there is an additional layer required. This optional layer of data-binding binds multiple streams into a single stream. This layer can be embedded in the layer of data-formation or data-processing layer. We model the *functional* layer as a set of output transformations and the *data-formation* layer as a set of transition transformations.

3.2.1 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.

3.2.2 Data-formation layer

Forming a new sliding window from the existing data set requires adding new data items and removing some data items. The rules of removing data items can be classified into two distinct categories as *intrinsic* or *extrinsic*.

An *extrinsic rule* of data removal is 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 rule for data formation is that the two layers can take place simultaneously facilitating optimized computation. While the processing layer transformations are being applied on the current window, the data formation layer functions can be applied to form the window to be processed at the next instance.

An *intrinsic rule* of data removal from the current data window is 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 dur-

Figure 2. Intrinsic rule of data removal.

ing processing layer of computation. The separation of two layers is not possible in this case, since the data items to be removed are determined after processing. Optimization in such cases may not be possible unless supported by additional resources.

3.3 Instance of processing

A query must be evaluated periodically to process an infinite stream. This periodicity of evaluation of results can be *procedure* based or *event* based. In the former case the instances are predictable where as in the later case they are in general unpredictable. The instance of query evaluation in example 1,2 and 4 from section 3.1 are procedure based and that in example 3 is event based. The procedure based periodicity in general is clock triggered [10]. The truth set of the window-formation predicate at each instance of query processing determines the window.

In the next section we present the formal model.

4 Formal Model

4.1 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.

- At any point of observation, data stream S is defined as $S = \{(x, t) | x \text{ is observed at time } t \text{ and } t \in [0, \text{now}())\}$. If S_t is a stream of values observed at t , then $S_t \supseteq S_{t-1} \supseteq S_{t-2} \dots \supseteq S_1 \supseteq S_0$. 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.
- *key-predicate* $\wp(x, t, \theta(W_t))$ is a predicate over data item and a function θ of the current window that determines the validity of the data item for the computations to be performed at the next instance.
- W_0 is the initial window. We define the window at instance $t > 0$ inductively as

$W_t = \{(x, t_k) | \wp((x, t_k, \theta(W_{t-1}))) \text{ is true at instance } t_k\}$

- **Formal Model :** Our model is given by $(W_t, \aleph, (\Im, \wp(x, t, \theta(W_{t-1}))))$

$\aleph$: is a set of output transformations

$\Im$: is a set of transition transformations together with a rule of removal defined by the *key - qualifier*. Transition transformations are defined through set-operations union, difference and cartesian product.

4.2 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) 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 $\{f_1, f_2, \dots, f_n\}$ where each f_i operates on (x_i, t_i) in the window.

$T_t(W_t) = \{(f_i(x_i), t) | (x_i, t_i) \in W_t \text{ and } f_i : \Omega \mapsto \Omega'\}$

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.

- (Aggregate A) Let $W_t = \{(x_1, t_1), (x_2, t_2) \dots (x_n, t_n)\}$

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 .

...

$A(\{x_1, x_2, \dots, x_n\}) = V_{n-1} \oplus x_n$

We demonstrate output transformations for example 1 given in section 3.1 as shown below.

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

4.3 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. We introduce third operation *Zip* that is required to form a window from two different streams.

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

- Transition operation Filter is defined by the operation of set-difference. This operation removes those elements from the current window which make the predicate $\wp$ false.

The data items from the window which make the predicate $\wp$ false are precisely, those data items, which expire from the current window. Some of the data items may expire temporarily while 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, these 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.
- 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 each data window in some situations.

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.

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

We demonstrate application of zip transformation for expressing the processing function in example 2.

The query involves processing of values from two streams. Processing function is the correlation coefficient given by the following formula:

$$corr(s, r) = \frac{\frac{1}{w} \sum (s_i \cdot r_i - \bar{s} \cdot \bar{r})}{\sqrt{\sum (s_i - \bar{s})^2 \cdot \sum (r_i - \bar{r})^2}}$$

$\bar{s}$ is modeled by $A(s_t)$ where s_t is the window on stream s at instance t . Similarly, $\bar{r}$ 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 of transform operator before summation. Transform operator T on s_t is given by $\{f_i | f_i(s_i) = (s_i - \bar{s})^2, i = 1, 2 \dots n\}$. 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 $\sum s_i \cdot r_i$.

$Z(s_t, r_t) = \{(s_i, r_i) | s_i \text{ and } r_i \text{ have same time-stamp value}\}$. [15]. $A(T(Z(s_t, r_t)))$ will compute $\sum s_i \cdot r_i$; where A is modeled by $\sum$ and T is given by $\{f_i(s_i, r_i) = s_i \cdot r_i | i = 1, 2 \dots n\}$. The result of this $A(T(Z(s_t, r_t)))$ is a singleton set, on which the Transform operator is applied to obtain the result of the expression in the numerator. Transform operator in this case is $T = \{f | f(x) = \frac{x}{w} - \bar{s} \cdot \bar{r}\}$

5 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. Stream S is scanned for checking occurrence of the pattern P .

Let $P = \{y[1], y[2], \dots, y[n]\}$; $S = \{(x, t) | x \in \Omega, t \in [0, \text{now}())\}$ represents the stream of characters observed till the current instance.

$W_t = \{(x_i, t_i) | x_i \text{ is observed at } t_i\}$ 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 $i=1, 2, \dots, n$, where $d(x, y) = 1$ if $x \neq y$
 $= 0$ 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 in respective windows are same.

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

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 $\{x[1], x[2], \dots, x[n]\}$ where $x[i]$ is observed before $x[i+1]$ for each $i=1, 2, \dots, n-1$.

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

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 $\sum$. 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.

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

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

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

6 Summary conclusion and future direction

We have presented here a generic model of forming 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. Effects of rules of formation of sliding windows on efficiency of the computation model will be studied in future.

References

- [1] 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, and S.Zdonik. Aurora: A data stream management system. In *ACM SIGMOD Intl. Conf. on Management of Data*, San Diego, California, June 9-12 2003.
- [2] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosensstein, and J. Widom. Stream: The stanford stream data manager (demonstration description). In *ACM SIGMOD Intl. Conf. on Management of Data*, San Diego, California, June 9-12 2003.
- [3] A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: Semantic foundations and query execution. Technical report, Stanford University, October 2003.
- [4] A. Arasu and J. Widom. A denotational semantics for continuous queries over streams and relations. In *SIGMOD Record*, volume 33 no 3, pages 6–11, 2004.
- [5] Z. Bar-Yossef, M. Fontoura, and V. Josifovski. On the memory requirements of XPath evaluation over XML streams. In *ACM SIGPODS Intl. Symposium on Principles of Database*, Paris, France, June 13-18 2004.
- [6] A. Dani and J. Getta. Computation of mathematical operators on sliding windows. In *Proceedings of the 5th International Conference on Recent Advances in Soft Computing*, Nottingham, United Kingdom, Dec 2004.
- [7] A. Das, J. Gehrke, and M. Riedewal. Approximate join processing over data streams. In *ACM SIGMOD Intl. Conf. on Management of Data*, San Diego, California, June 9-12 2003.
- [8] J. Getta and E. Vossough. Optimization of data stream processing. In *SIGMOD Record*, volume 3 no 3, 2004.
- [9] C. Luo, H. Thakkar, H. Wang, and C. Zaniolo. A native extension of SQL for mining data streams. In *ACM SIGMOD Intl. Conf. on Management of Data*, pages 662–662, Baltimore, Maryland, June 14-16 2005.
- [10] M.A.Hammad, W.G.Aref, M.J.franklin, M.F.Mokbel, and A.K.Elmagarmid. Efficient execution of sliding window-queries over data stream, 2003.
- [11] K. Patraoumpas and T. Sellis. Window specification over data streams, 2006.
- [12] M. Stonebraker, U. Cetinmel, and S. Zdonik. The 8 requirements of real-time stream processing. In *SIGMOD Record*, volume 34(4), pages 42–47, 2005.
- [13] W. T.M.Ghanem and A.K.Elmagarmid. Exploiting predicate-window queries in data stream management systems. In *SIGMOD Record*, volume 35 no 3, 2006.
- [14] E. Vossough. A system for processing continuous queries over infinite data streams. In *Intl. Conference on Database and Expert Systems Applications*, Zaragoza, Spain, Sept 2004.
- [15] Y.Zhu and S.Sasha. Fast approaches to simple problems in financial time series streams. In *Proc. Workshop on Management and Processing of Data Streams*, San Diego, CA, June 2003.