2005

Incremental computation of aggregate operators over sliding windows

Anita Dani
University of Wollongong in Dubai

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

Publication Details

Dani, A. & Getta, J. R. (2005). Incremental computation of aggregate operators over sliding windows. In M. Bouhlel & B. Soliman (Eds.), International Conference: Sciences of Electronic, Technologies of Information and Telecommunications (pp. 1-8). Susa, Tunisia: SETIT.

Incremental computation of aggregate operators over sliding windows

Abstract

Sliding Window is the most popular data model in processing data streams as it captures finite and relevant subset of an infinite stream. This paper studies different Mathematical operators used for querying and mining of datastreams. The focus of our study is on operators, operating on the whole data set. These are termed as blocking operators.We have classified these operators according to their method of evaluation. An operator is termed as aggregate operator if it produces the sub-set of values satisfying some given condition. The evaluation of these operators is more complex when the query is continuous and data are changing. We present here a formal definition of incremental computation onsliding windows. We have proposed a vector model and graph abstraction to represent the sliding window and algorithms to evaluate Mathematical operators on the sliding window. The model is robust and can be applied to visualize different mathematical operators on the sliding window. We have introduced a novel concept of checkpoints in order to make the computation incremental. We present an efficient algorithm to find maximum (or minimum) on the sliding windows using the checkpoint concept.

Disciplines

Business | Social and Behavioral Sciences

Publication Details

Dani, A. & Getta, J. R. (2005). Incremental computation of aggregate operators over sliding windows. In M. Bouhlel & B. Soliman (Eds.), International Conference: Sciences of Electronic, Technologies of Information and Telecommunications (pp. 1-8). Susa, Tunisia: SETIT.

Incremental Computation Of Aggregate Operators Over Sliding Windows

Anita Dani and Janusz Getta

University Of Wollongong In Dubai,

U.A.E.

AnitaDani@uowdubai.ac.ae

University Of Wollongong,

Australia

jrg@uow.edu.au

Abstract: Sliding Window is the most popular data model in processing data streams as it captures finite and relevant subset of an infinite stream. This paper studies different Mathematical operators used for querying and mining of data streams. The focus of our study is on operators, operating on the whole data set. These are termed as blocking operators. We have classified these operators according to their method of evaluation. An operator is termed as aggregate operator if it produces the sub-set of values satisfying some given condition. The evaluation of these operators is more complex when the query is continuous and data are changing. We present here a formal definition of incremental computation on sliding windows. We have proposed a vector model and graph abstraction to represent the sliding window and algorithms to evaluate Mathematical operators on the sliding window. The model is robust and can be applied to visualize different mathematical operators on the sliding window. We have introduced a novel concept of checkpoints in order to make the computation incremental. We present an efficient algorithm to find maximum (or minimum) on the sliding windows using the checkpoint concept.

Key words: Algorithm, incremental, sliding windows

1 Introduction

A data stream is a continuous flow of data from the source to the destination. Digital Signal processing, Time series analysis in Economics and Finance, processing data obtained from sensor networks are some examples of data streams. In a typical data stream, the volume of data is potentially unbounded. 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 applications where real time responses are of prime importance, e.g. in stock market applications, speed of processing is a vital issue. In applications where interpretation of data is

Even though 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 a window is the most popular data structure in data streams processing. It captures a finite subset of an infinite stream that is relevant to the present time slot. A timestamp associated with each tuple in a stream allows us to broadly categorize windows as time windows and data windows.

Finite subsets of the data stream are processed at certain time interval. In order to achieve real-time responses to the queries on changing data, approximate computations are performed. A typical approach is to store synopses (Gibbons, Matias 1998) such as histograms, or summary results (or sketches), such as average of values, number of distinct values or values that satisfy certain condition. Using these synopses, queries are evaluated with approximate answers. Determining nature of ideal synopsis has always been a research issue. We propose that mining mostly qualitative, accuracy of answers is less important than the processing speed.

queries on data stream applications can be best evaluated on the basis of sliding window model.

Windows is the most popular data model in data streams as it captures finite and relevant subset of infinite stream. These windows can be large and can change rapidly and unpredictably. It is important to develop algorithms that can trade off between the memory requirements and the running time at the same time produce exact answers. Incremental computation of aggregate operators on sliding windows is not as simple as it is on append-only data sets since it is required to synchronize addition and deletion of set of tuples at the same time. We present here a formal definition of incremental computations on sliding windows. We can apply this definition to any aggregate operator on sliding windows.

We have analyzed algorithms on sliding windows for evaluation of aggregate operators. Aggregate operators are those operators, which take all elements from the underlying data set and produce a single value or a list of values satisfying some given condition. We call these algorithms as searching algorithms. For evaluation of aggregate operators on sliding window, not just the value, but also the position of the result within the data set is important. If we model the underlying data set carefully, we can always relate the position to the time-stamp attribute. We propose here a vector model to represent the sliding window. The vector model has facilitated some linear algebraic representation for mathematical operator. We also provide some guidelines to save intermediate results in order to make the computation incremental. In particular, we propose an efficient algorithm to find maximum (or minimum) on the sliding windows.

An algorithm can be considered as *incremental* or windowed if it can avoid re-processing of the elements within the window, which have not expired from the window. Reusability is the important criterion of incremental algorithms. An algorithm can be *partially incremental* if certain part of the computation can be done incrementally. These algorithms can decompose the required computation into smaller sub-tasks and achieve optimization. An algorithm can be considered as *adaptive* algorithm if it can utilize the idle time to pre-process and store certain results for future computation. Awareness of future computations is an important criterion of adaptive algorithms. An *approximate* algorithm can perform certain approximation at the operator level in order to achieve incremental computations. It is important to know

representation. It also includes the concept of checkpoints. Section 8 illustrates how to model the operator Max using the given algorithm. Conclusion and future work is given in Section 9. Illustration of low-level operators is given in the Appendix.

2. Related work

Comparative study of issues in databases and data streams is presented by (Babcock et al 2003). Data stream queries can be reasonably complex and persistent as well transient. Resources such as memory are limited. So the focus of research is on efficient query processing. Researchers (Guha, Kudas 2001) (Qiao et al, 2003) have worked on developing single pass algorithms or incremental algorithms to evaluate aggregate queries. A study of space requirements for single-pass algorithms and the algorithms, which make a few passes over the input data, is presented by (Rauch et al 1998). Some applications require joining of two streams or joining a stream with a table. Since all of the stream data need not be archived, it is necessary to develop an efficient join strategy. Das at al (Das et al 2002)and L. Golab and M. Tamer (Golab, Tamer 2000) have developed a new strategy to join two data streams. Datar et al (Datar et al 2003) have presented efficient algorithms to compute stream statistics over sliding windows of bits. The concept of basic windows has been presented by (Zhu, Shasha 2002). The evaluation of an aggregate using basic window is partially incremental as the results are refreshed only after the stream fills the basic window. An algorithm to find MAX/MIN based on sorting of the window elements presented by (Qiao et al 2003). (Arasu et al 2001) have presented a class of queries over multiple data streams, which can be computed using bounded memory. In presence of efficient algorithms to evaluate queries, next challenge is to build efficient query plans. Niagara CQ (Chen et al 2002) group and (Avnur, Hellerstein 2000) have worked on optimization of query processing. Research groups such as STREAM (Stream 2003), Aurora (Zdonik et al 2003), Telegraph CQ (Krishnamurhy et al 2003) have worked on developing DSMS in order to process streams. Comparative study of these systems is presented by (Koudas, Srivastava 2003). A survey of recent work on data stream management systems is presented by (Golab et al 2003). Golab, Tamer and Ozsu (Golab et al 2003)

incremental computations. It is important to know before hand, if an algorithm evaluating the given operator is incremental or not. If it is not incremental then it can be made to work faster by storing certain values.
The rest of the paper is organized as follows: Summary of related work is given in section 2. We have highlighted our contributions in section 3. Section 4 contains definitions and new notation. Low-level operators and their symbolic representation are included in Section 5. Two classes of algorithms are introduced in Section 6. Section 7 includes formal definition of the aggregate operator and its matrix

2003). Golab, Tamer and Ozsu (Golab et al 2003) have presented a classification of different streaming algorithms according to the method of generating synopses and according to the function. B.Moon et al (Moon et al 2003) have presented a variety of temporal aggregation algorithms. The algorithm presented in this paper for small-scale aggregation is based on data partitioning and parallel processing technique. The algorithm to find min/max is similar to merge-sort algorithm. The underlying assumption is that the available memory is large enough to store entire data structure, such as tree, required by each aggregation algorithm.

3. Our Contributions

We present a detail study of the algorithms on a generic level. Our research is directed towards exact computation for any data. We have explored different characteristics of an algorithm, which are useful in the context of sliding window. It is not practical to store all data, and the data once rejected cannot be recovered. Theoretical derivations given here can detect the data elements, which are required for future computation. The algorithm presented here, can decide usefulness of the data element for future computations, thus making the algorithm incremental. Our optimization technique is data independent. The optimization is not affected by the arrival order or distribution of values. It does not require a priori knowledge of the size of the data set.(Gibbons and Matias 1998)

4. Definitions and Symbols

4.1 Basic terms

A data *stream* is an infinite sequence of tuples where each tuple has timestamp as an additional attribute. The tuples arrive into the stream in sequential manner and the default access mode is sequential only. *Window* is a finite sub-sequence of data stream, which has well defined scope. *Sliding window* is the mechanism of forming overlapping sub-sequences of tuples, which are relevant to the application, at pre-determined instances. *Time Window* is the window formed on the basis of time-stamp, where the time-stamp of each tuple lies within the 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 is not fixed. *Data window* is the window formed on the basis of number of tuples. If the rate of

5. Symbolic representation

5.1 Low-level operators

These are the operators that operate on vectors and output another vector. Each of these operators will modify one of above mentioned attributes of the input vector. We can represent the stream operators as a sequence of low-level operators. In this section we introduce graph abstraction of different types of operators. The nodes of the graph will represent operators and arrows will indicate the flow of data and the sequence of operator. This is based on the abstraction presented by (Chen, Kotz 2002).

The operator *Filter* splits the given vector into sub-vectors according to some given condition. The operator *Merge* appends elements of the second vector to the first vector. The operator *Transform* is a vector $<f_1, f_2,..f_n>$ of functions, such that each fi operates on xi The output of this operator is another vector of same dimension, but having different values. The operator *Accumulator* operates on a single vector of dimension n and the outputs a vector of dimension 1. The output vector is a result of some mathematical processing, such as addition. The operator *Aggregator* operates on a vector of dimension n and outputs a sub-vector of dimension k. The output sub-vector is the vector of elements satisfying certain condition. The evaluation of this *Aggregator* operator involves evaluation of certain predicate for each element. The operator *Permutation* operates on a vector of dimension n and outputs another vector of same dimension by changing the positions of the elements within the window. *Source* and *Write* operators are streaming operators, which are used for reading and writing. They do not produce any output but transfer data from the source to the destination. The operator *Write* writes the input vector to working area, intermediate storage area or permanent storage. We formed on the basis of number of tuples. If the rate of arrival is constant, then the window will slide at regular interval.

4.2 Modelling sliding window as a vector

Let the stream consist of a single attribute x. Each tuple will have additional attribute timestamp t. Let R_1, R_2,..R_n,.. . be the tuples from the original stream where Ri = (xi, ti) and ti ≤ tj if i<j. We define the sliding window at any instance as a vector X= $<x_{t+1}, x_{t+2}, x_{t+3}, \ldots x_{t+n}>$ where t=0, 1, 2,..

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

intermediate storage area or permanent storage. We make this distinction in the view of the architecture proposed by (Babu, Widom 2001).

Operator	Symbol	Domain	Co-domain	Changed attribute
Filter	F	$\Re n$	$\Re_{k1}$ X $\Re_{k2}$.. X $\Re_{km}$	Dimension
Merge	**M**	$\Re_{k1}$ X $\Re_{k2}$.. X $\Re_{km}$	$\Re n$	Dimension
Aggregator	**A**	$\Re n$	$\Re 1$	Dimension
Accumulator	**Ac**	$\Re n$	$\Re 1$	Dimension
Permutation	**P**	$\Re n$	$\Re n$	Position
Transform	**T**	$\Re n$	$\Re n$	Value
Source	**S**			
Write	**W**			

Table 1. *Lower-level operators on a Window Vector*

Any operator on a sliding window can be represented as a sequence of low-level operators.

Example: Operator Avg can be represented as **T**(**Ac**(**T**($<x_{t+1}, x_{t+2}, x_{t+3}, \ldots x_{t+n}>$)))

Operator Max can be represented as **A**($<x_{t+1}, x_{t+2}, x_{t+3}, \ldots x_{t+n}>$).

5.2 Graph abstraction

Any operator, which performs some mathematical operation on the elements before accumulating the values, can be modelled by Figure (1). Any aggregate operator, which searches for a value or a set of values satisfying the given condition, can be modelled by Figure (2).

Elements of the window vector are written to the temporary storage area. Whenever new elements arrive, the window vector is refreshed. The operator is evaluated repeatedly over the elements of the window vector. Since the window is sliding, any two successive window vectors have some elements common. According to the flow of evaluation indicated above, these common elements are re-processed every time. At some instances only one element may arrive and one element may leave the window. Even though n-1 elements have not changed, they are re-processed.

It is important to know, if the operator can get the

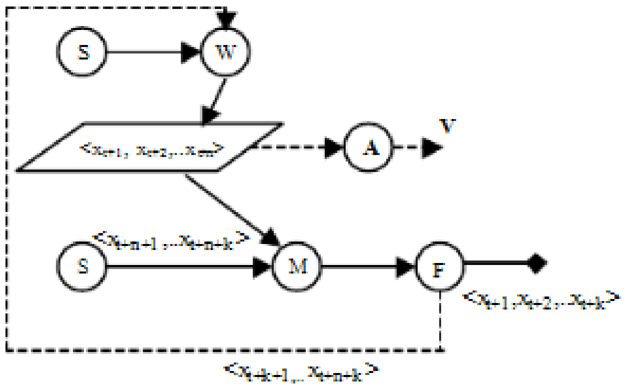


Figure 2. *Searching on a Sliding Window* [1]

6. Categories of algorithms

We introduce two categories of the algorithm as follows:

- *Searching algorithm* – the algorithm, which finds or searches for a value satisfying the given condition. Example algorithm to find max or algorithm to find first even number within the window.
- *Processing algorithm* – the algorithm, which processes all elements within the window and produces a result through some mathematical operations. Example Finding Lp norm or finding

It is important to know, if the operator can get the next result from the elements, which are moving. For the incremental evaluation of the operator the following steps are required:

- Undo the effect of elements, which are moving out of the window.
- Evaluate the operator on the elements, which are entering the window.
- Combine the results of the above two steps with the result of the previous evaluation.

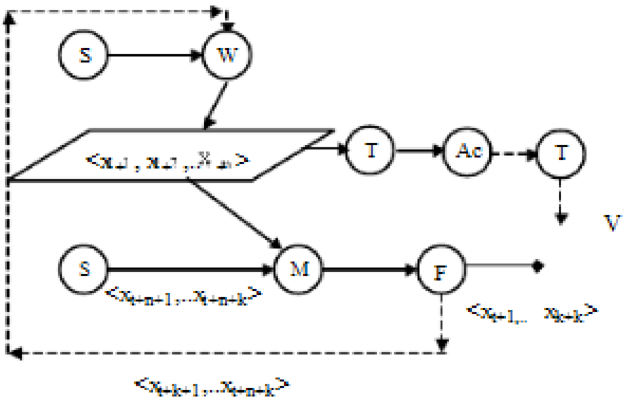


Figure 1. *Processing on a Sliding Window*

operations. Example Finding Lp norm or finding weighted average of all elements within the window.

The searching algorithms may become incremental in some cases but in general may not be incremental. The processing algorithm can be made incremental with proper choice of processing function. We will analyze searching algorithms in the following section.

7. Aggregate operator

7.1 Definition

Simple searching operation can be implemented via a select query with appropriate where condition. If the predicate included in the where clause contains constants, then the evaluation of searching algorithm is simple and non-blocking. Our aim is to study those aggregate operators, which can be evaluated only after processing all the elements of the window. Let $W=\langle x_1, \ldots x_n \rangle$.

We define an aggregate operator $\mathbf{A}: \Re^n \rightarrow \Re^m$ as a sequence of iterations:

$V_1 = A'(x_1, V_0)$
$V_2 = A'(x_2, V_1)$
…
$V_n = A'(x_n, V_{n-1})$

V_n is the required output which is the value of $\mathbf{A}(W)$. V_0 is the given vector of initial values and each of the V_i is a vector that stores the intermediate results computed at the i-th step. An operation A' takes on input an element from the window and the vector of values obtained from previous operation. For example,

[1] ——◆: Discarded ---: Repeated operations

if vector V_i contains the largest k elements in a sub-window $\langle x_1, x_2, \ldots x_i \rangle$ then operation $A'(x_{i+1}, V_i)$ finds the largest k elements in a sub-window $\langle x_1, x_2, \ldots, x_i, x_{i+1} \rangle$.

7.2 Matrix representation

Let $W_t = \langle x_{t+1}, x_{t+2}, \ldots x_{t+n} \rangle$ be the window vector observed at instance t. Let $(x_{t+n+1}), \ldots, (x_{t+n+k})$ be the tuples that enter the stream at the next instance. We can denote this as the vector Wnew= $\langle x_{t+n+1}, \ldots x_{t+n+k} \rangle$. The sub-vector Wold= $\langle x_{t+1}, \ldots x_{t+k} \rangle$ will indicate the tuples removed from the window. New window Wt+1 is represented as W_t **M** W_{old} **F** W_{new}

the previous result. Intuitively, this is equivalent to computing $\mathbf{A}^{-1}$(Wold). In many cases, from the very nature of the operator **A**, it is not possible to define $\mathbf{A}^{-1}$. $\mathbf{A}(W_{new})$ can be evaluated before merging W_{new} with W_t **F** W_{old}.

We can formulate this evaluation as follows:

Let $\mathbf{A}(W_t) = V_k$.
If $\mathbf{A}(\mathbf{A}(W_{new}), \mathbf{A}(W_t)) = \mathbf{A}(W_{new})$ then $\mathbf{A}(W_{t+1}) = \mathbf{A}(W_{new})$
else
If $V_k \subseteq W_{old}$ then
$\mathbf{A}^{-1}(W_{old}) = \mathbf{A}(W_t$ **F** $W_{old})$
$\mathbf{A}(W_{t+1}) = \mathbf{A}(\mathbf{A}^{-1}(W_{old})$ **M** $\mathbf{A}(W_{new}))$
else
$\mathbf{A}(W_{t+1}) = \mathbf{A}(W_t)$
Replace W_t by W_{t+1}.

It follows from this formulation that the represented as W_t **M** W_{old} **F** W_{new}

Observations:

- dim (W_{old})= dim (W_{new}) in case of data window only.
- W_t **F** W_{old} M W_{new}= W_t **M** W_{new} **F** W_{old}
- W_t **F** W_{old} contains elements, which are not expired from the window.

 We construct a triangular matrix ?W to represent the contents of the window W_t while it moves by one element at each instance. The aggregate function **A** operates on the diagonals of ?W. Let D_t be the diagonal that contains the elements x_{t+i} for i=1,2..n. D_{t+1} includes elements x_{t+i+1} for i=1,2..n.

 In general, D_{t+k} = W_t **F** <x_{t+1},...x_{t+k}>.

 Similar representation is given by (Dwilde, Alle-Jan van der Veen 1998).

When new element arrives, it is appended as a new column on the right and the left-most column is removed from the matrix. Hence the matrix can be considered as a right-open matrix.

$$\Delta W = \begin{bmatrix} x_{t+1} & x_{t+2} & \cdots\cdot & \cdots x_{t+n} \\ & x_{t+2} & \cdot\cdot & \cdots x_{t+n} \\ & & x_{t+3} & \cdots \ \cdots\cdot \\ & & & \cdots\cdot x_{t+n} \end{bmatrix} \ldots\ldots(I)$$

Each diagonal represents W_t **F** W_{old}

7.3 Incremental evaluation

We redefine the concept of incremental evaluation to suit the sliding window applications as follows:

Evaluation of an algorithm is incremental over the sliding window, if it is not re-processing the elements from the window, which have not expired from the window. We present here a formal model of evaluation of any aggregate operator over sliding window.

Let **A**(W_t) be the result of evaluation of aggregate operation over the window at instance t and **A**(W_{t+1})= **A**(W_t **F** W_{old} **M** W_{new}). The evaluation of **A** is incremental if we can undo the effect of **A**(W_{old}) from

It follows from this formulation that the incremental evaluation is not uniform but depends on the values of the elements, which are moving.

Lemma: The evaluation of the operator **A** is incremental from instance t to t+λ if A(D_t)=$V_{t+\lambda}$ where $\lambda \in$ {1,2,..n)

7.4 Introducing checkpoints

The algorithm can be made incremental by supporting this evaluation with some other results, such as, finding **A**(Wt **F** Wk) in advance and using it as $\mathbf{A}^{-1}$. Here W_k is the sub-vector <x_{t+1},..x_{t+k}> where **A**(W_t)=V_k.

We call **A**(W_t **F** W_k) as the checkpoint to which the effect of **A**(Wold) can be rolled back. It is possible to repeat this process by filtering out sub-vector containing the previous result.

We introduce checkpoints for any aggregate operator in order to achieve incremental computation. These checkpoints will provide the successive results when the window moves beyond the incremental scope. The checkpoints are computed and stored at the initial stage and updated later on as the window moves.

For any application, where data are random, the number of checkpoints is much smaller than the window size.

Algorithm to find the list of checkpoints

Consider the elements $V_{\lambda 1}$, $V_{\lambda 2}$, …$V_{\lambda p}$.

Observe that $\lambda_1 > \lambda_2 ... > \lambda_p$ and

$V_{\lambda 2}$ = A(W **F** <x_1,..$x\lambda_1$>) = A^{-1}(x_1, x_2,..$x_{\lambda 1}$)

$V_{\lambda 3}$ = A^{-1}($x_{\lambda 1}$,...$x_{\lambda 2}$)

..

$V_{\lambda p}$ = A^{-1}($x_{\lambda(p-2)}$,...$x_{\lambda(p-1)}$)

When the new elements arrive, aggregate is evaluated over these elements in order to update the list of checkpoints.

$x_{\lambda(p+1)}$ = A($x_{\lambda 2}$, x_{n+1}) , $x\lambda_{(p+2)}$ = A($x_{\lambda 2}$, x_{n+1}), ... Depending on the definition of the operator A and the outcome of the evaluation, some of the checkpoints will be cleared or the new element will be added to the checkpoint list. The algorithm is given below:

Algorithm to evaluate aggregate operator over sliding window, using checkpoints

Let $W = \langle x_1, x_2, .. x_n \rangle$

i=1; μ=0

algorithm is sorting only the key elements and this list, in general is much smaller than the window size. The size of this list, in the worst case is same as the window size, which means the entire window is sorted. This becomes a special case and can be treated differently. By maintaining a separate buffer, we can

Repeat

$V_k = A(W)$

$\lambda_i = \mu + k$

$W \to WF\langle x_{\mu+1}, x_{\mu+2}, ..x_k \rangle$

$i \to i+1, \mu \to k$

Until k ≤ n

$Output: \lambda_1, \lambda_2, ..\lambda_p$

Repeat for every j → 1, 2,…

$V_{n+j} = A(x_{\lambda_k}, x_{\lambda_{p-1}}, ...x_{\lambda_p}, x_{n=j})$

$\lambda_{p+1} = \mu + k$

if k≤p remove $\lambda_1, ..\lambda_{k-1}$ else add x_{n+j} to the checkpoint list.

Remove λ_1 if x_λ expires from the window.

8. Some aggregate operators

8.1 Aggregate Max (or Min)

Let us assume that aggregate operator indicates finding maximum of the window elements at every sliding movement. We can express this as A=Max(W_t). This operator will return the maximum value and its highest position within the window. Let A(W_t)= xk then

$V_1 = Max(x_1) = x1$.

$V_2 = Max(x_2, V_1)$

$V_3 = Max(x_3, V_2)$

..

$V_n = Max(x_n, V_{n-1})$

Note that each Vk consists of a single value and $Vk = V_{K+1} = .. = Vn$

From the formal algorithm given in section 7.4, we get A(Wt **F** Wold)= Max(Wt **F** Wold).

Let us assume that the window slides after reading a single element every time. That is dim (W_{old})= dim (W_{new}) =1. The matrix representation in (I) can reveal the status of the window for next n instances. We use this information to compute the results for next n instances and make the computation incremental. The best case is when k=n and the worst case is when k=1. We propose here an algorithm to find maximum of the window elements incrementally. The evaluation of this operator is data dependent. We use this characteristic of the algorithm to make it efficient by probing on the positions of the key values. This avoids unnecessary sorting and minimizes number of comparisons.

Max can be evaluated incrementally by keeping a sorted list in the memory (Qiao et al 2003). Our efficiently. By maintaining a separate buffer, we can process the new elements separately. Moreover, once the checkpoints are established, the old elements are not pre-processed even when the max element expires from the window. This is consistent with the new definition of incremental computation. Hence we claim that this is computation of max over sliding window is incremental.

Algorithm to find Max over a sliding window using checkpoints:

Initialization: $W_0 = \langle x_1, x_2, ..x_n \rangle$

Find checkpoints $\lambda_1, \lambda_2, ..\lambda_k$ such that

$x_{\lambda_1} > x_{\lambda_2} ..... > x_{\lambda_k}$

and all elements between x_{λ_i} and $x_{\lambda_{i-1}}$ are smaller than $x_{\lambda_{i-1}}$.

Return ($x_{\lambda 1}$) as the answer.

Repeat for each i

Let $W_{new} = \langle x_{n+i} \rangle$

Compare x_{n+i} with $x\lambda$ for each $\lambda = \lambda_1, ..\lambda_k$

If ($x_{n+i} \geq x_{\lambda i}$) then reset $x_{\lambda i} = x_{new}$ and clear the smaller checkpoints.

Else

Add this element to a temporary buffer

Until i<λ1

If (i==λ1) then find maximum of the buffer elements x_{n+p}

Return $x_{\lambda 2}$ as the answer and add the new check point x_{n+p}

Illustration: Let W_0 = <3,8,12,6,5,10,4,2>

Checkpoints= <12,10> max = 12

x_9= 1 ...add it to the buffer

x_{10}=11 reset checkpoints, checkpoint=<12,11>

8.2 Finding the longest increasing sequence

The algorithm to find and maintain the longest increasing (or decreasing) sequence when the window is sliding can be modeled in a similar manner. Each Vk will be a vector of increasing values. First checkpoint is set at the λ where xλ is the last element of the first longest increasing sequence. The procedure is repeated for window formed by removing first λ elements. The last checkpoint is xn. If xn+1 < xn, then a new sequence is formed in the buffer else, xn+1 will be added to the last sub-sequence. As the elements from the first longest sequence are being removed, its length is compared with second-longest subsequence.

9. Conclusion and future Direction

Aggregate operator and its incremental computation over sliding window are defined formally. Using this definition, it is possible to compute exact answers to an aggregate operator with optimal memory utilization.

Main characteristic of an aggregate operator is to search for an element or set of elements satisfying the given condition. We have developed an efficient incremental algorithm that computes some useful results in advance. The algorithm checks for usefulness of data for future computation and optimizes the memory utilization by storing the positions of the key elements along with their values.

The vector model for the window is robust and can be extended to structures instead of single values. The checkpoint list technique can be used for algorithms, which require multiple passes over the data set. Since the algorithm implements the checklist as a simple list and the list is changed by insert and delete operations only, there are no overhead expenses to update this list. The new concept of incremental computation can be generalized for complex operators formed by finite sequence of low-level operators **T**, **Ac** and **A**. We will present modeling and optimization of such operators in future.

References

(Arasu 2001) Arasu, Babcock B.,Babu S., McAlister J., and Widom J. Characterizing Memory Requirements for Queries over Continuous Data Streams, *Symposium on Principles of Database Systems, Madison, Wisconsin, USA, 2002.*

(Avnur2000) Avnur R. and Hellerstein J.M., Continuous Query Optimization, *Proc. ACM SIGMOD, 2000.*

(Babu 2001) Babu S. and Widom J., Continuous Queries Over Data Streams, *ACM SIGMOD Record , 2001.*

(Babcock 2003) Babcock B.,Babu S., Datar M.,Motwani R. and Widom J., Models and Issues in Data Stream System, *Proc.21st ACM-SIGMOD-SIGACT-SIGART, 2002.*

(Chen 2001) Chen G. and Kotz D., Context Aggregation and Dissemination in Ubiquitous Computing Systems, *4th IEEE Workshop on Mobile Computing Systems and Applications, Callicoon, New York, 2002.*

(Chen 2002) Chen J., DeWitt D., Tian F. and Wang Y., Niagara CQ:A Scalable Continuous Query System For Internet Databases, *Proc. ACM SIGMOD, Conference, Dallas, Texas, USA, 2000.*

Spatscheck O., and Shkapenyuk V., The Gigascope Stream Database , *Bulletin of the Technical Committee on Data Engineering, IEEE Computer Society, March 2003.*

(Das 2002) Das A., Gehrke J., and Riedewald M., Approximate Join Processing Over Data Streams, *Proc. ACM SIGMOD Conference on Management of Data, San Diago, California, USA, 2003.*

(Datar 2003) Datar M., Gionis A., Indyk P. and Motwani R., Maintaining Stream Statistics Over Sliding Windows, *Proc. of 13th ACM-SIAM symposium on Discrete Algorithms, 2002.*

(Dewilde 1998) Dewilde P. and Alle-Jan van der Veen, *Time-Varying Systems and Computations*, Kluwar Publications 1998.

(Dobra 2003) Dobra A., Garofalakis M., Gehrke J.,Rastogi R., Processing Complex Aggregate Queries Over Data Stream, *ACM SIGMOD Madison, USA, 2002.*

(Gibbons 1998) Gibbons P. and Matias Y. Synopsis Data Structures For Massive Data Sets, *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science: Special Issue on External Memory Algorithms and Visualization,1998.*

(Golab 2000) Golab L. and Tamer M., Sliding Window Multi-way Joins Over Data Streams, *Proc. Of 29th VLDB Conference, Berlin, Germany 2003.*

(Golab 2003) Golab L., Tamer M. and Ozsu, Data Stream Management Issues - A Survey Technical Report CS-2003-08 Available at http://db.uwaterloo.ca/~ddbms/publications/stream/streamsurvey.pdf

(Greenwald 2001) Greenwald M and Khanna S., Computation of Quantile Summaries, *ACM SIGMOD , CA, USA, 2001.*

(Guha 2001) Guha S. and Koudas N., Approximating a Data Stream for querying and Estimation: Algorithms and Performance Evaluation, *Annual ACM Symposium on Theory of Computing, Hersonissos, Greece, 2001.*

(Koudas 2001) Koudas N. and Srivastava D., Data Stream Query Processing, *Proc. Of 29th VLDB Conference, Berlin, Germany 2003.*

(Krishnmurthy 2003) Krishnmurhty S., Chandrasekaran S., Cooper O., Deshpande A., Franklin M., Hellerstein J.M., Hong W., Madden S., Reiss F. and Shah M., Telegraph CQ : An Architectural Status Report, *IEEE Data Engineering Bulletin 26,1 March 2003.*

(Moon 2003) Moon B., Lopex I.F.V. and Immanuel V., Efficient Algorithms for Large Scale Temporal Aggregation, *IEEE Transactions On Knowledge and Data Engineering VOL .15 NO. 3 MAY/JUNE 2003.*

(Motwani 2003) Motwani R., Widom J., Arasu A.,Babcock B.,Babu S., Datar M., Manku G., Olston C.,Rosenstein J. and Varma R., Query Processing Resource Management and Approximation in a Data Stream, *Proc. Conference on Innovative Data Systems Research, Asilmor, CA, USA, 2003.*

(Qiao 2003) Qiao L. Chen S., Li H.G., Agarwal D. and Abbadi E.E., Efficient Computation for Max/Min queries in Sliding Windows, *Unpublished Manuscript.*

(Rauch 1998) Rauch M., Henzinger, Raghavan P. and Rajgopalan S., Computing On Data Streams available at http://hpl.hp.com/rechreports/Compaq-DEC/SRC-TN-1998-011.html

(Stream 2003) Stream Group, STREAM The Stanford Stream Data Manager, *IEEE Data Engineering Bulletin 26, March 2003.*

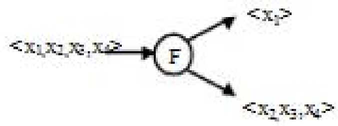
(Zdonik 2003) Zdonik S., Stonebreaker M., Cherniack M., Cetintemel U., Balazinska M. and Balakrishan H., The Aurora And Medusa Projects, *Bulletin of the Technical Committee on Data Engineering, IEEE Computer Society, March 2003.*

(Zhu, 2002) Zhu Y. and Shasha D., Stat Stream : Statistical Monitoring of Thousands of Data Streams in Real Time, *Proceedings of 28th VLDB 2002.*

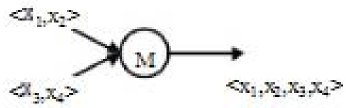
Appendix:

Graphical representation of low-level operators on data stream

A-(i)



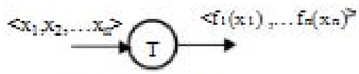
A-(ii)



A-(iii)

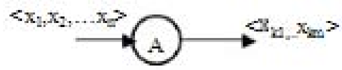
$V = \oplus x_i$ where $\oplus$ is some mathematical operator

A-(iv)



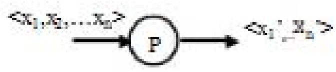
$f_i : \Re \rightarrow \Re$ and $f_i \neq f_j$

A-(v)



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

A-(vi)



$x_i' = x_j$ for some i and j