STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 4, 2011

A NEW ARCHITECTURE SUPPORTING THE SIZING WINDOW EFFECT WITH STREAMINSIGHT

SABINA SURDU

Abstract. Data stream processing is a new paradigm that emerged in the last years in the field of data management. Dedicated systems are designed to deal with the challenges posed in the process of executing continuous queries over infinite data streams. One of the most sensitive issues in this context is resource usage. The current approaches to minimize system memory and CPU consumption don't take into account the size of a window which is input to a query. In a previous paper we proposed a novel technique, that tackles resource usage by assessing the Sizing window effect. In this paper we propose a new architecture for the Sizing window effect, built on top of a commercially available DSMS, namely the WindowSized architecture.

1. Introduction

For a long time, traditional information processing has been the answer to a large number of data management applications. Conventional database management systems were handling a significant amount of the data processing. These systems store information persistently as finite relations. Queries are issued against the database when needed and their results only reflect the current state of the data. Recently, this processing paradigm proved its shortcomings when it came to handling data that was not static, but continuous, i.e. *data streams*. New challenges are posed in the paradigm of data stream processing. One of the most important ones is resource usage.

The objective of this paper is twofold. First, we propose a new architecture for optimal query processing, using the Sizing window effect, on top of a commercially available Data Stream Management System (DSMS): StreamInsight. We highlight our main contributions, as well as the benefits and limitations of

Received by the editors: November 25, 2011.

2010 *Mathematics Subject Classification.* 68M20, 68N01, 68P01.

1998 *CR Categories and Descriptors.* C.4 [**Computer Systems Organization**]: Performance of Systems – *Measurement techniques*; D.0 [**Software**]: General; H.2.4 [**Information Systems**]: Database Management – *Systems - query processing* .

Key words and phrases. data stream processing, sizing window effect, continuous queries, data stream processing systems, windowsized architecture.

this approach. Second, we motivate the need for continuous query processing in a world where continuous data is present in an ever larger number of fields. Health care monitoring, network monitoring, telecommunications, astronomy or seismography are only some of the fields that are coping with data streams. In the end we conclude on the results of our work and provide future research directions.

2. Data stream processing

A data stream is a sequence of values produced over time by a data source. Most of the applications that cope with data streams are called monitoring applications, because they monitor an arbitrary number of data streams. This class of data-intensive applications scans data streams, performs some processing on read data values and computes desired output in real time. For a detailed description of this field please refer to our paper [13].

Data streams are processed by continuous queries, that perpetually run over time and provide updated results as their underlying input data changes. As the field matures, novel challenges are being posed in the process. One of the most important ones is resource usage.

2.1. **Resource usage in data stream processing.**

Taking into account the temporal dimension adds great complexity to the query processing paradigm. The number of data sources as well as the rate at which data arrives at a system can greatly increase. This can affect the system's ability to output desired results in a real time manner, when processing a great number of complex, continuous queries. Various query optimization techniques (*e.g.* operator scheduling or load shedding [15]) are proposed, in order to tackle resource usage when processing high throughput data streams with continuous queries. To the best of our knowledge, none of them takes into account the size of the input window. In this subsection we will briefly review related work on query optimization and resource usage in data stream processing.

STREAM is a DSMS developed at Stanford, which supports declarative continuous queries over data streams and stored relations [4]. The system tackles performance issues in three ways. First, it eliminates data redundancy, by sharing state within and across query plans. Second, it drops data that will not be used, by exploiting constraints on streams (referential integrity, ordered-arrival and clustered-arrival). Third, the system uses operator scheduling techniques, which aim at minimizing intermediate state [2].

[11] discusses approximation methods from STREAM, which are caterogized into static techniques and dynamic techniques. Load shedding is mentioned in the second category, as the process of dropping tuples from inter-operator queues in the query operator tree, when the queues become too large.

SoCQ, described in [8], is a Pervasive Environment Management System, which provides a declarative way of writing continuous queries against classical data, streams and services. The cited paper describes basic query optimization goals (*e.g.* reduce intermediary relations in queries) and rules (*e.g.* pushing selections down in the query operator tree).

NiagaraCQ introduces a novel approach to executing continuous queries. Apart from executing queries in a change-based manner, every time a tuple appears on a stream, NiagaraCQ introduces timer-based continuous queries, which execute at time instants specified by the user, as [6] shows. This manner of executing queries can greatly impact resource consumption.

Aurora and Medusa are two related DSMSs, described in [18]. The former is centralized, whereas the latter is distributed, using Aurora as a single-site processing engine.

Aurora is based on a combined train and superbox scheduling approach [5]. By train scheduling, Aurora batches multiple tuples as long input trains for operators (boxes in Aurora query diagram) to execute. This saves box call overhead (as the number of box calls decreases) and allows a box to better optimize its execution with an increased number of data elements. By superbox scheduling, Aurora pushes a tuple train through multiple boxes, which avoids the cost of going to disk. Among other optimization techniques, we can enumerate: inserting projections as soon as possible in the query diagram in order to reduce tuple sizes, combining boxes whenever possible to reduce box execution cost and reordering commutative boxes. Apart from these techniques, Medusa encompasses cross-site optimization methods.

Borealis is a second generation DSMS, based on Aurora (for stream processing features) and Medusa (for distributed stream processing capabilities) [12]. This system uses collaborative optimizers on three levels: local, neighbourhood and global. Their purpose is to tackle problems like locating throughput bottlenecks or latency [1].

2.2. **Linear Road.** In order to compare DSMSs, various benchmarking frameworks have been proposed. A key benchmark designed for this purpose is Linear Road [16]. We choose to discuss this benchmark, since our experiments in [14] were conducted on data from Linear Road and we plan to use it in the WindowSized architecture as well.

The Linear Road benchmark is designed to compare performances of Data Stream Management Systems relative to each other and to traditional relational Database Management Systems [3]. Linear Road simulates a variable tolling system in a fictional metropolitan area, Linear City. This urban setting contains 10 parallel expressways, that run horizontally from one another. Each expressway is 100 miles long and is divided into 100 one-mile long segments.

Each expressway has two directions of travelling, Eastbound and Westbound and 4 lanes in each direction (3 travelling lanes and one dedicated lane for entering into and exiting from the expressway). An MIT traffic simulator [17] generates input data for the benchmark, as a set of vehicles that take trips on the expressways. Each vehicle is considered to be equipped wich a device that emits a position report every 30 seconds (describing the time at which the report is generated, the vehicle's speed and location). Based on segment statistics (like average number of vehicles, average speed and proximity of accidents), a variable toll is issued for a vehicle, every time it enters a new segment. The purpose of variable tolling is to reduce expressways congestion at peak traffic periods.

Apart from the continuous queries that compute tolls and detect accidents, the system also supports historical queries issued each time a position report is emitted by a vehicle, with a given probability (like requests for vehicle account balance or travel time predictions). Each query response must fulfil response time and precision requirements specified by the benchmark. A DSMS that implements Linear Road is assigned an L-rating, which represents the number of expressways it can support while still meeting the benchmark requirements. Linear Road is supported by systems like STREAM and Aurora.

3. The Sizing window effect

In [14] we laid the bricks of our proposed novel approach to cope with resource usage in data stream processing, namely the Sizing window effect. Our technique attempts to compute an optimal window size for a given continuous query, thereby placing a minimal upper bound on the resource consumption for that query, as our previous paper shows. We are interested in aggregate queries [10] over windows that are not semantically significant.

3.1. **Formalization.** We will describe the formalization model that we developed in [14]. We consider a discrete, ordered time domain T, as the infinite countable set of time instants, starting from the past and going into the future. For simplicity, we will choose the domain of relative integers to represent T: $T = \mathbb{Z} = \{-\infty, \ldots, -2, -1, \ldots, 3, 4, \ldots, +\infty\}$, where $t_i = i$, adopting the approach described in [9].

Then we define a stream S as follows: $S = \{(s, t)|s \in D^{|schema(S)|}, t \in T\}$, where D is the countable infinite set of constants and $schema(S)$ is the list of attributes in the schema of S. In the rest of this paper we will consider that t_c is the current timestamp.

If we denote by Q a continuous query, then by writing $Q(S, t_c)$ we refer to the result of evaluating Q against stream S, at time t_c. We will denote the

result of executing a query against a stream, at current timestamp, with R_s:

$$R_s = Q(S, t_c) \tag{1}$$

A sliding window over stream S is defined as follows: $SW(S, [t_i, t_c]) = \{(s, t) \mid (s, t) \in S, t \in [t_i, t_c]\}$. The starting point of this window in time is t_i and the endpoint is t_c. For simplifying purposes, this alternative equivalent notation can be used: $SW_{ic}(S)$. Then the result of evaluating Q at time t_c against a sliding window over stream S that starts at t_i and ends at t_c is denoted by $Q(SW_{ic}(S), t_c)$. We will denote the result of executing a query against sliding window $SW_{ic}(S)$, at current timestamp, with $R_{w_{ic}}$:

$$R_{w_{ic}} = Q(SW_{ic}(S), t_c) \tag{2}$$

We will consider the size of a window to be the number of time instants contained by the window. The size of the window $SW_{ij}(S)$ is $t_j - t_i + 1$.

As we already mentioned, we focus on aggregate queries over data streams. This means that at any current timestamp t_c, both R_s and $R_{w_{ic}}$ contain aggregate results from the real numbers domain $\mathbb{R}$. We plan to extend our focus to queries that issue non-aggregate results, as explained in the last section.

We defined the following aggregate distance, for aggregate-queries results, which measures the precision of a windowed result, when compared with a result obtained from a query executed against the stream:

$$distance : \mathbb{R} \times \mathbb{R} \to \mathbb{R}, distance(R_s, R_{w_{ic}}) = |R_s - R_{w_{ic}}|, \tag{3}$$

where R_s and $R_{w_{ic}}$ are both obtained at current timestamp t_c, as explained above. The size of the window from which $R_{w_{ic}}$ is obtained can be deduced from the temporal coordinates t_i and t_c. R_s is considered to be the correct result, since it is obtained when executing the query against all the data from the stream.

3.2. **Experiments.** We performed the experiments in [14] on data from Linear Road, for three continuous queries. We will briefly present the strategy we used and the results we obtained for one of the queries.

In order to find an optimal window size, we started with a window that contained all the data that had arrived on the stream and evaluated R_s, the result of the query executed against this window (*i.e.* against the entire stream), which was the correct result. Subsequently, we constantly decreased the size of the window, up to the point when the result of the query did not fulfil specified precision requirements. At that point, we had just reached the minimal window size, which needed minimal resource usage, while still meeting precision exigencies.

Let us denote this window size by σ. This means the average of the *distance* function between the correct result and the results obtained from

windows of size σ is below a given precision threshold, whereas the average of the *distance* function between the correct result and the results obtained from windows of size $\sigma - 1$ or less is above the given precision threshold. At the same time, a query executed against any window of size greater than σ would use more system resources (CPU and memory to process and to store respectively more tuples).

For query: "Compute the average number of vehicles per time unit that have been travelling on a given segment", we reached an optimal window size of 1000 time instants, for a precision threshold of 1, as Figure 1 shows. This means choosing a window size of 10000 time instants would produce correct results, with considerably increased resource usage. On the other hand, a window size of 100 time instants would use less memory and CPU, but the precision of the result will be degraded.

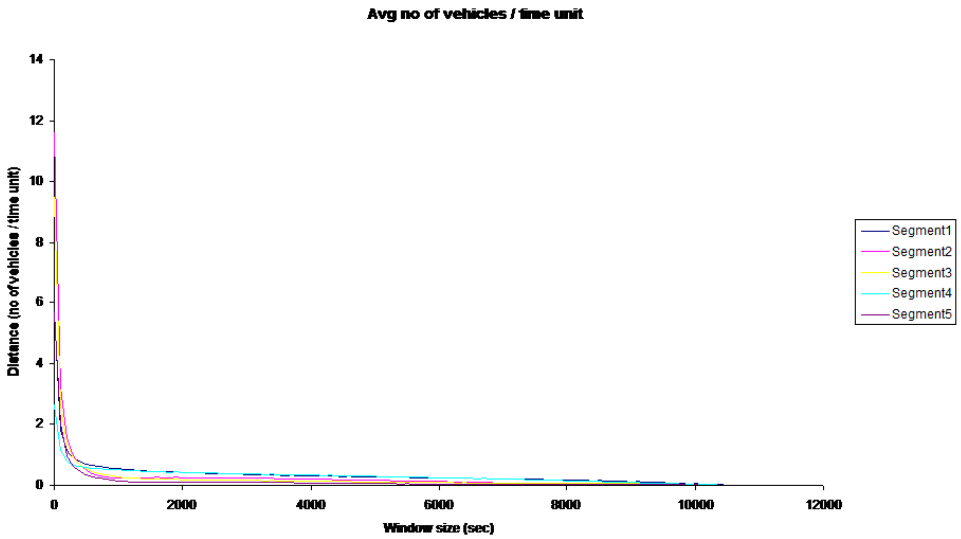

FIGURE 1. Average number of vehicles / time unit

4. WINDOWSIZED ARCHITECTURE

StreamInsight is a DSMS released by Microsoft, as part of SQL Server 2008 R2 [19]. It encompasses a temporal query engine, which executes *standing queries* (Microsoft's terminology for continuous queries) over input data streams, which flow through the system. Data streams are produced by data sources and are feeding input adapters, which translate them into a format

understood by the StreamInsight engine. The results of the standing queries are fed into output adapters. The latter translate these results into formats that the event targets understand.

In order to be able to write any kind of queries and register them on the server, a C# application must be developed. This encompasses continuous queries written in LINQ, which are registered into the StreamInsight server. The complete integration of LINQ queries in the C# application makes it easier to develop a monitoring application on this platform.

On top of this application we are planning the integration of a WindowSizing module, which takes into account the executing query and computes an optimal window size for the query. Figure 2 highlights the main components of such an architecture. This architecture is based on the StreamInsight application architecture approach, composed of event sources, input adapters, standing queries in the StreamInsight query engine, output adapters and event targets [7]. Our contribution is represented by the WindowSizing module, which communicates with the query engine to evaluate the nature of the queries, with input adapters - to change the window size and with output adapters - to obtain query results.

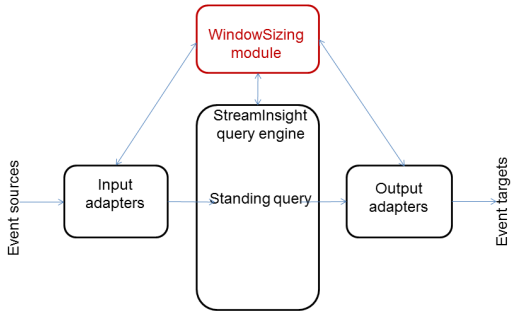

FIGURE 2. WindowSized architecture

The main difficulty is represented by the computation of the correct result. In the experiments we previously described, we bulk loaded the Linear Road data into a commercially available DBMS and computed the correct result on all the available data. But in a DSMS we cannot compute the correct result against an in-memory stored collection of 12 million records. Hence we

propose to choose a maximum window size, depending on available system resources. For this purpose we define the function:

$$maxWindowSize : CPU \times Memory \times TupleSize \times DataRate \to \mathbb{N} \tag{4}$$

This function takes as input the available CPU and memory in the system, the maximum size of the tuples from the input stream and the maximum data rate (number of tuples received every time instant). It outputs a natural number, representing the maximum number of time instants a window can contain, when being processed on the current machine. This is the maximum window size, expressed as number of time instants, accepted by our architecture.

Then we will consider the correct result to be the one obtained when executing the query against a window, whose size is computed by evaluating maxWindowSize. Even though this result is an approximation of the correct, ideal result, it is the best approximation we can obtain when executing continuous queries and will be used as a reference point for queries executed against considerably smaller windows.

The task of establishing the optimal window size is not left for the query developer any more. In cases where the result of a query is more and more accurate as the window size increases, resource consumption also increases. Hence a developer may not choose the best window size when writing continuous queries. By using the WindowSized architecture, the system can better reason about the available resources and can greatly minimize both memory and CPU usage. Another advantage is the use of the Visual Studio IDE and the complete .NET Framework integration, which can greatly reduce development time for monitoring applications, when the .NET platform is already used by an entity [7].

5. Conclusion and future directions

In this paper we proposed a new architecture for processing continuous queries over data streams, using our previously developed Sizing window effect: the WindowSized architecture. We designed a WindowSizing module that computes the optimal window size for a given continuous query, on top of a monitoring application developed with StreamInsight. We highlighted the benefits and limitations of our approach. Our implementation of a prototype based on the WindowSized architecture is currently in progress. We could have chosen any other DSMS. StreamInsight was chosen based on personal technical experience with .NET technologies and the availability of the platform on the MSDN Academic Alliance Software Center.

Future work will revolve around two research directions. First, we will rigorously formalize our WindowSized architecture. Subsequently, we will finalize the development of a prototype based on the WindowSized architecture. Eventually we will conduct a set of extensive experiments using the prototype. As a second objective, we plan to extend our focus of research on queries which issue non-aggregate results. We are working on a distance function for queries that issue collections of tuples as results. We will conduct experiments on these types of queries as well and integrate the newly obtained functionalities in the WindowSized architecture.

References

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

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

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

[4] S. Babu, J. Widom, *Continuous Queries over Data Streams*, SIGMOD Rec., vol. 30, n° 3, pp. 109-120, 2001.

[5] D. Carney, U. Cetintemel, A. Rasin, S. Zdonik, M. Cherniack, M. Stonebraker, *Reducing Execution Overhead in a Data Stream Manager*, ACM Workshop on Management and Processing of Data Streams, 2003.

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

[7] T. Grabs, R. Schindlauer, R. Krishnan, J. Goldstein, R. Fernández, *Introducing Microsoft StreamInsight*, Technical article, 2010.

[8] Y. Gripay, F. Laforest, J.-M. Petit, *SoCQ: A Framework for Pervasive Environments*, 10th International Symposium on Pervasive Systems, Algorithms and Networks, pp. 154-159, 2009.

[9] Y. Gripay, *A Declarative Approach for Pervasive Environments: Model and Implementation*, Ph.D. Thesis, Institut National des Sciences Appliquées de Lyon, 2009.

[10] J. Li, D. Maier, K. Tufte, V. Papadimos, P.A. Tucker, *Semantics and evaluation techniques for window aggregates in data streams*, In SIGMOD Conference, pp. 311-322, 2005.

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

[12] E. Ryvkina, A. S. Maskey, M. Cherniack, S. Zdonik, *Revision Processing in a Stream Processing Engine: A High-Level Design*, Proceedings of the 22nd International Conference on Data Engineering (ICDE 2006), 2006.
[13] S. Surdu, *Data stream management systems: a response to large scale scientific data requirements*, Annals of the University of Craiova, Mathematics and Computer Science Series, vol. 38, n° 3, pp. 66-75, 2011.
[14] S. Surdu, V. M. Scuturici, *Addressing resource usage in stream processing systems: sizing window effect*, The International Database Engineering and Applications Symposium ACM International Conference Proceeding Series, pp. 247-248, 2011.
[15] N. Tatbul, *QoS-Driven Load Shedding on Data Streams*, EDBT '02 Proceedings of the Workshops XMLDM, MDDE, and YRWS on XML-Based Data Management and Multimedia Engineering-Revised Papers, pp. 566-576, 2002.
[16] R. S. Tibbetts, III, *Linear Road: Benchmarking Stream-Based Data Management Systems*, M.Sc. Thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 2003.
[17] Q. Yang, H. N. Koutsopoulos, *A Microscopic Traffic Simulator for Evaluation of Dynamic Traffic Management Systems*, Transportation Research Part C, vol. 4, n° 3, pp. 113-129, 1996.
[18] S. B. Zdonik, M. Stonebraker, M. Cherniack, U. Cetintemel, M. Balazinska, H. Balakrishnan, *The Aurora and Medusa Projects*, IEEE Data Engineering Bulletin, vol. 26, n° 1, pp. 3-10, 2003.
[19] ***, *Microsoft StreamInsight. Product documentation*, http://msdn.microsoft.com/en-us/library/ee362541.aspx, accessed: 03.11.2011.

Babeş-Bolyai University, Department of Computer Science, 1 M. Kogălniceanu St., 400084, Cluj-Napoca, Romania
E-mail address: surdusabina@yahoo.com