Towards a Semiring for Continuous Query Provenance

Sebastian Labbe, Samuele Langhi, Angela Bonifati, Riccardo Tommasini

Towards a Semiring for Continuous Query Provenance

Sebastian Labbe[1], Samuele Langhi[2], Angela Bonifati[2] and Riccardo Tommasini[3]

[1]ENS Ulm PSL, Paris; [2] Lyon 1 University ; [3] INSA Lyon, France

Abstract

Data provenance encompasses tracking data's origins, transformations, and derivations within a system. Provenance semirings, a mathematical framework used mainly in databases, play a key role in representing and managing provenance information and offering a structured approach to handle complex provenance queries, such as why-provenance and how-provenance, by encapsulating the essential properties of these models within their algebraic structure. In streaming scenarios, however, such models are not enough to describe the full spectrum of data provenance. Indeed, we need to account for the time dimension in streaming since data is continuously generated and processed in real time. For this reason, we study data provenance under a novel perspective, introducing When-provenance, which aims to describe the origin of the data over the time dimension and provide insights into the temporal dynamics of data transformations. This includes identifying timestamps of data generation, processing intervals, and the timing of query execution stages within defined time windows while minimizing the related performance overhead. Our contribution aims to define a when-provenance formalism to allow modification of a stream query window operator in sublinear time complexity in the input size to allow real-time, enhanced debugging, performance optimization, and audibility of streaming systems. It can also ensure compliance with temporal constraints and policies. By integrating When-provenance mechanisms, streaming query systems can achieve robust and transparent temporal tracking, leading to greater reliability and deeper insights into the temporal behaviour of data streams.

Keywords

provenance semirings, stream provenance, stream processing

1. Introduction

Data provenance is essential in several areas of data management, such as scientific data processing or consistent query answering. Given the growing need for processing data in real-time, recent studies have investigated data provenance models for streaming data and continuous queries [1]. Such works on streaming data study the problems of efficient annotation propagation [2], explaining missing answers [3] of continuous queries, or maintaining the query results upon rapid changes [4]. However, the characterization of the temporal aspect of continuous queries was neglected. Notably, window operators are critical in defining continuous query semantics. Window operators address the data unboundedness problems by dividing the infinite input streams into finite portions based on temporal conditions.

Nevertheless, the stream processing community has long struggled to determine the appropriate window for continuous queries. This task requires prior knowledge about the data to reduce the risk of unwanted approximation [5]. Ideally, a given continuous query over the whole stream to obtain the most complete results. Naturally, this is not possible, as the query is limited by the working memory size. Moreover, windowing introduces non-monotonicity in the query results [6] and non-determinism [7]. Given such challenges, it is critical to empower users of stream processing systems with a theoretical framework that can help explain the evaluation of a window-based continuous query and potentially help correct it.

This paper presents a provenance framework for tracking the results of continuous queries within a given window for providing a quality assessment over the adopted query. We called this framework

AMW 2024: 16th Alberto Mendelzon International Workshop on Foundations of Data Management, September 30th–October 4th, 2024, Mexico City, Mexico

sebastien.labbe@ens.psl.eu (S. Labbe); samuele.langhi@univ-lyon1.fr (S. Langhi); angela.bonifati@univ-lyon1.fr (A. Bonifati); riccardo.tommasini@insa-lyon.fr (R. Tommasini)

0000-0002-3674-0292 (S. Langhi)

When provenance, and it provides an alternative, interval-based interpretation of why and how provenance through provenance semirings. To efficiently track When provenance, we designed an alternative data structure that, given a window, allows for an efficient re-computation of the result over the window sub-portions. Then, we approach the quality assessment of the window as a *counting problem* by calculating the cardinality of the recomputed results and assess whether a window sub-interval is capable of returning (almost) the same number of results of the original window, but with fewer resources. Finally, we show that such recomputation and the related quality assessment are sublinear in the number of results.

2. Problem Formulation

This section lays the groundwork for this work and formally defines the targeted problem. More specifically, it presents a definition of continuous queries with a focus on window operators. Additionally, it provides an overview of provenance semirings and the various types of provenance, i.e., *Why*, *How*, lineage (*Lin*), and *Bool*.

2.1. Preliminaries

Continuous Queries (CQ) are a kind of query evaluated over an infinite stream. The result of a CQ evaluation is also a stream, as described in the Continuous Semantics [6, 5], which allows extending query execution to unbounded data. In this paper, we consider the data model of [5], i.e., a **stream** is a totally ordered, infinite sequence of records $(\omega, \tau) \in \Omega \times \mathrm{T}$, where ω is a tuple in Ω, and τ is a timestamp in the time domain T, that for simplicity we consider as the set of natural numbers. We also consider time-based window operators since they are popular in existing systems, e.g., Flink [8] or Spark [9][1]. Operationally, time-based sliding windows are defined as a tuple (ι, β) where ι represents the size and β the slide [5]. Window operators are defined as a function $W : \mathrm{T} \to I$ from the time domain T to the set of intervals $I = \mathrm{T} \times \mathrm{T}$. Hence, applying a window operator W to a stream S assigns each record of S to an interval $w = (\tau_s, \tau_e) \in I$. We write Q for a continuous query and S for a stream. We write $Q[S]$ as the result of the query Q on the stream S and $Q[S]_w$ for the same result restricted to the interval w, i.e.,

$$Q[S]_w = Q[S'] \text{ where } S' = \{s \mid s \in S \wedge W(s) = w\}$$

Provenance semirings [11] represent various provenance models by annotating database tuples with elements from a set K, e.g., provenance in event tables [11] assumes $K = \mathscr{P}(X)$, with $\mathscr{P}(X)$ being the powerset of all tuple IDs X, assigned through function $ID : \Omega \to X$. A *semiring* based on K is an algebraic structure of the form $(K, +, \times, 0, 1)$, where $+$ and $\times$ are addition and multiplication over K, e.g., $+ = \cup, \times = \cap$ when $K = \mathscr{P}(X)$; Element 0 and 1 are respectively the neutral element of $+$ and $\times$, e.g., $0 = \emptyset, 1 = X$ when $K = \mathscr{P}(X)$. Notably, $(K, +, 0)$ and $(K, \times, 1)$ are commutative monoids and $+$ is distributive over $\times$, i.e., , 0 is the annihilating element for $\times$, i.e., $\forall a \in K, a + 0 = a$, $a \times 1 = a$, and $a \times 0 = 0$. On top of this, a K-relation is a function $\mathscr{A}$ that associates elements of the database to elements of the semiring, e.g., $\mathscr{A} : \Omega \to \mathscr{P}(X)$. K-relations propagate provenance annotations according to relational operators in relational algebra [11]. The requirement for finite support of K-relations makes them inapplicable on infinite streams. Some notable provenance semirings in our scope are: How provenance, based on polynomials over $\mathbb{N}$, defined as $How(X) = (\mathbb{N}[X], +, \cdot, 0, 1)$, captures the most informative form of provenance; Boolean polynomial provenance, based on polynomials over $\mathbb{B}$, defined as $Bool(X) = (\mathbb{B}[X], +_{idem}, \cdot, 0, 1)$, with idempotent addition; Lineage, $Lin(X) = (\mathscr{P}(X), \bigcup, \bigcup, \bot, \emptyset)$, represents data lineage in data warehousing; Why provenance, which extends data lineage with subsets, defined as $Why(X) = (\mathscr{P}(\mathscr{P}(X)), \bigcup, \bigcup^{\cup}, \emptyset, \{\emptyset\})$, or as $Why(X) = (\mathbb{B}[X]/\{x^2 - x, \forall x \in X\}, +_{idem}, \cdot_{idem}, 0, 1)$.

[1] For a comprehensive survey on window operators, we suggest [10]

2.2. Problem Statement

Our goal in this paper is to assess the quality of a given window operator W adopted by a continuous query $\mathcal{Q}$. Indeed, defining the correct window for a given continuous query is crucial as it provides the basis for complex stateful computations: when the window is too small, the CQ misses some relevant results; when it is too big, the CQ requires too much memory, potentially hindering the performance of the query. While the first case has been studied by [3] in the context of missing answers identification, less attention has been paid to assessing if a window requires too many resources wrt query results.

This paper approaches the window quality assessment as a result counting problem. The intuition is that if query $\mathcal{Q}$ produces almost or all the results over a portion of a given interval w produced by W, then w is overestimated. For this reason, we focus on an efficient method for the problem of window-based result counting.

Problem 1. *Let S be a stream, W a window function, $\mathcal{Q}$ a continuous query. The result counting problem over a given interval w returned by W is determining the cardinality of $\mathcal{Q}[S]_{w'}$, for each $w' \subseteq w$.*

3. The When Provenance Framework

To study When provenance, we build on the hierarchy defined by Green et al.'s semiring framework. In particular, we study Why provenance ($Why(X)$), and How provenance ($How(X)$) as they represent the minimal, most expressive form of provenance that can support interval representation. Indeed, we do not use lineage (or which provenance) ($Lin(X)$) and boolean polynomial provenance ($Bool(X)$): while the former is not expressive enough to identify subsets of timestamps (sub-intervals), the latter is as expressive as $Why(X)$ when used for intervals.

We compose our framework by replacing the tuple IDs X with the set of tuple timestamps T, obtaining $Why(T)$ and $How(T)$. Given a result tuple r of a query $Q[S]$ and its Why-provenance element $x_1 + x_2x_3 \in Why(X)$ each monomial m is a minimal list of input tuple IDs necessary for generating the given result tuple r. For when provenance, the polynomial becomes $t_1 + t_2t_3 \in Why(T)$. In this case, each monomial m is a list of timestamps used to generate the tuple r.

How provenance, and the related when provenance variant ($How(T)$) is built over the same intuition, but it includes natural coefficients and exponents in the polynomial, e.g., $t_1^2 + 2t_2t_3 \in How(T)$. Such coefficients and exponents enable the counting of the multiplicity of a given interval in record generation, which may be useful in certain use cases.

Since our goal is to track generation intervals, we care about the min and the max of a given monomial m from an element in $Why(T)$ or $How(T)$. To obtain a formalism that only keeps such information, our objective is to replace each monomial m with an interval $i = (\min(m), \max(m))$. To do this, given an input tuple r and its timestamp t_r we annotate it with the interval $i_r = (t_r, t_r)$ and redefine the How and Why provenance semirings on these intervals.

More specifically, we define two new semirings $How(I)$ and $Why(I)$, which keep the same polynomial structure as $How(T)$ and $Why(T)$, respectively. We modify the original polynomial multiplication $(\cdot)$ to include an interval-by-interval multiplication $i_1 \times_I i_2 = (\min(i_{1s}, i_{2s}), \max(i_{1e}, i_{2e}))$. This operation allows intra-monomial simplification to a single interval while maintaining the same benefits of natural or boolean coefficients from $How(T)$ and $Why(T)$.

Definition 1. *Let I be the (infinite) set of intervals, which also include unbounded intervals. We define $How(I) = (\mathbb{N}[I], +, \cdot_I, 0, 1)$ as the when provenance semiring, where $\cdot_I$ is the classic polynomial multiplication enhanced with $\times_I$, which is applied when two intervals are multiplied.*

Conversely, $Why(I)$ can be defined following the same approach, but without natural coefficients in the monomials and exponents on labels. For this reason, through $Why(I)$ we would not be able to account for multiplicity in the analysis generation intervals.

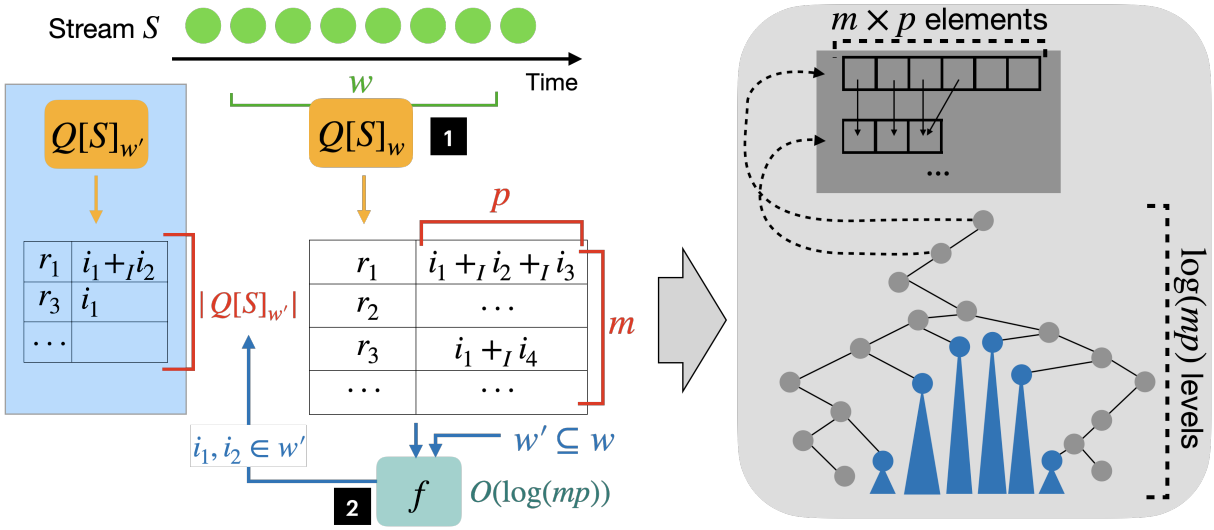

Figure 1: An overview of our approach, with highlighted the first and the second part of our query.

4. Efficient When Provenance Querying

We now discuss the design of an efficient data structure for solving our counting problem. As shown in Figure 1, our approach is twofold: **1)** The initial query Q_w, evaluated on the input stream S, with the When-provenance annotation described above and on an interval w, i.e., the interval including all data of interest. This part's complexity is that of a query with provenance labels. **2)** A function f, which takes as input the result $Q[S]_w$ and an interval $w' \subseteq w$ and returns the number of results in the query $Q[S]_{w'}$ with their interval multiplicity. We do so by taking the When-provenance data from the first part and finding all the result tuples r and their generation intervals i_k included in w'.

The time and space complexity of f depends heavily on the data structure that stores the When-provenance annotations. We approach it as a 2D orthogonal range counting problem [12] by storing all the intervals i_k contained in $Q[S]_w$ in a data structure based on balanced binary search trees and fractional cascading, shown in Figure 1. The intervals are stored and ordered by start-time at the leaves. Each node in the tree points to a separate list of all intervals in its sub-tree ordered by end-time. In a node's list, each interval keeps a pointer to the interval with the closest end-time in child node's list.

To query this data structure with an interval $w' \subseteq w$ we begin by two binary searches in the root node's list (containing all intervals), to find the first and last end-times included in w'. We then use the tree to perform the same operation on start times. At each step, we use the pointers between the lists to update the first and last end times included in w' for the new sub-tree. The goal is to find all the blue nodes of the tree as shown in Figure 1, which creates a partition of the set of intervals with start times in w'. At a blue node, we use its stored list and a prefix-sum query to count the number of intervals with both start and end times in w'. We then aggregate the results from all the blue nodes. The first two binary searches are $O(\log(pm))$ each step down is $O(1)$, and there are $O(\log(pm))$ blue nodes which can be found in $O(\log(pm))$ steps. At each blue node, we count the number of intervals in $O(1)$. With total query time $O(\log(pm))$ and pre-processing complexity $O(pm \log(pm))$ [13].

Example 1. *To concretely demonstrate how our approach can handle such streaming data scenarios, we examine a specific example involving a data stream composed of tuples, each containing three attributes: A, B, and C, accompanied by their corresponding timestamp, denoted as τ. The objective is to execute a query, as shown in Listing 1, which is formulated using the Continuous Query Language (CQL). This query is designed to perform a self-join operation based on the attribute B, within a defined time window. The time window is characterized by both a size (ι) and a slide (β) of 4 time units. After executing the self-join, the query focuses on projecting the result onto attributes A and C from the joined tuples. In line with the approach adopted by existing systems, e.g., Apache Flink and Kafka Streams [8, 14], we follow*

the convention that the result of a join operation in a continuous query will carry the timestamp of the maximum value between the timestamps of the joined tuples.

```
SELECT R1.A AS A, R2.C AS C
FROM Stream R1, Stream R2 [RANGE 4 SLIDE 4]
WHERE R1.B = R2.B,
```

Listing 1: A CQL query that performs a self-join over a tumbling window of 4 time units, projecting over values A and C.

Table 1 shows a sample stream and the result of the query over two consecutive window instances identified by intervals $[1, 5)$ *and* $[5, 9)$*. The three tables also include the How-provenance and When-provenance annotations, with the latter described through the semiring defined in Section 4.*

Once the various annotations are obtained, we can follow the approach described in Figure 1. Through When Provenance annotations, that trace the generation intervals of a given result, we can calculate how many results are produced within sub-intervals of both $[1, 5)$ *and* $[5, 9)$*. By doing so, we can intuitively see that sub-intervals* $[2, 5)$ *and* $[6, 9)$ *respectively produce the same number of results of intervals* $[1, 5)$ *and* $[5, 9)$*. This equivalence suggests that a reduction of the window size, wrt the one used by the query from Listing 1, may be needed.*

Table 1
A sample stream (1a) and the related results of the query from Listing (1), executed over records within interval $[1, 5)$ (1b) and $[5, 9)$ (1c), with the respective ID-provenance $How(X)$ and when-provenance $How(I)$ initial annotations. In gray the columns related to provenance metadata.

A	B	C	τ	X	I
...	...	...	...	...	...
μ	β	ξ	1	a	[1,1]
ζ	γ	ϕ	2	b	[2,2]
θ	γ	ν	3	c	[3,3]
ρ	γ	κ	4	d	[4,4]
δ	φ	ι	5	e	[5,5]
χ	λ	η	6	f	[6,6]
χ	λ	α	7	g	[7,7]
σ	λ	α	8	h	[8,8]
...	...	...	...	...	...

(a)

A	C	τ	$How(X)$	$How(I)$
...	...	...	...	...
μ	ξ	1	aa	[1,1]
ζ	ϕ	2	bb	[2,2]
ζ	ν	3	bc	[2,3]
θ	ϕ	3	bc	[2,3]
θ	ν	3	cc	[3,3]
ζ	κ	4	bd	[2,4]
θ	κ	4	cd	[3,4]
ρ	ϕ	4	bd	[2,4]
ρ	ν	4	bc	[3,4]
ρ	κ	4	dd	[4,4]
...	...	...	...	...

(b)

A	C	τ	$How(X)$	$How(I)$
...	...	...	...	...
δ	ι	5	ee	[5,5]
χ	η	7	ff+fg	[6,6]+[6,7]
χ	α	8	fg+fh+ gh+gg	[6,7]+[6,8]+ [7,8]+[7,7]
σ	η	8	fh	[6,8]
σ	α	8	gh+hh	[7,8]+[8,8]
...	...	...	...	...

(c)

5. Conclusion

This paper discusses the application of Green et al.'s provenance semirings to streaming data and continuous queries. We demonstrate how the result counting problem for a given query can be effectively addressed through the introduction of a novel framework. This framework reinterprets the traditional concept of provenance semirings by adapting them to operate over time intervals, which is particularly suitable for streaming data where temporal aspects are crucial. In addition to the theoretical contributions, we propose a specialized data structure designed for the efficient computation of result cardinality. This data structure leverages interval analysis techniques within the context of 2D orthogonal range counting. The use of this approach offers significant performance guarantees in terms of computational complexity and resource utilization.

Furthermore, the paper discusses the potential for future enhancements to this framework. Indeed, our data structure can be extended to its dynamic version [15] to obtain $O(\log(pm)^2)$ amortized insertion and deletion.

References

[1] D. Palyvos-Giannas, V. Gulisano, M. Papatriantafilou, Genealog: Fine-grained data streaming provenance in cyber-physical systems, Parallel Comput. (2019).

[2] B. Glavic, K. S. Esmaili, P. M. Fischer, N. Tatbul, Ariadne: managing fine-grained provenance on data streams, in: S. Chakravarthy, S. D. Urban, P. R. Pietzuch, E. A. Rundensteiner (Eds.), The 7th ACM International Conference on Distributed Event-Based Systems, DEBS '13, Arlington, TX, USA - June 29 - July 03, 2013, ACM, 2013.

[3] D. Palyvos-Giannas, K. Tzompanaki, M. Papatriantafilou, V. Gulisano, Erebus: Explaining the outputs of data streaming queries, Proc. VLDB Endow. (2022).

[4] D. Palyvos-Giannas, B. Havers, M. Papatriantafilou, V. Gulisano, Ananke: A streaming framework for live forward provenance, Proc. VLDB Endow. (2020).

[5] A. Arasu, S. Babu, J. Widom, The CQL continuous query language: semantic foundations and query execution, VLDB J. 15 (2006) 121–142. URL: https://doi.org/10.1007/s00778-004-0147-z. doi:10.1007/S00778-004-0147-Z.

[6] D. B. Terry, D. Goldberg, D. A. Nichols, B. M. Oki, Continuous queries over append-only databases, in: M. Stonebraker (Ed.), Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 2-5, 1992, ACM Press, 1992.

[7] P. Carbone, J. Traub, A. Katsifodimos, S. Haridi, V. Markl, Cutty: Aggregate sharing for user-defined windows, in: S. Mukhopadhyay, C. Zhai, E. Bertino, F. Crestani, J. Mostafa, J. Tang, L. Si, X. Zhou, Y. Chang, Y. Li, P. Sondhi (Eds.), Proceedings of the 25th ACM International Conference on Information and Knowledge Management, CIKM 2016, ACM, 2016.

[8] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, K. Tzoumas, Apache flink™: Stream and batch processing in a single engine, IEEE Data Eng. Bull. (2015).

[9] M. Armbrust, T. Das, J. Torres, B. Yavuz, S. Zhu, R. Xin, A. Ghodsi, I. Stoica, M. Zaharia, Structured streaming: A declarative API for real-time applications in apache spark, in: G. Das, C. M. Jermaine, P. A. Bernstein (Eds.), Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018, ACM, 2018, pp. 601–613. URL: https://doi.org/10.1145/3183713.3190664. doi:10.1145/3183713.3190664.

[10] J. Verwiebe, P. M. Grulich, J. Traub, V. Markl, Survey of window types for aggregation in stream processing systems, VLDB J. (2023).

[11] T. J. Green, G. Karvounarakis, V. Tannen, Provenance semirings, in: L. Libkin (Ed.), Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 11-13, 2007, Beijing, China, ACM, 2007.

[12] P. K. Agarwal, J. Erickson, et al., Geometric range searching and its relatives, Contemporary Mathematics 223 (1999) 1–56.

[13] D. E. Willard, G. S. Lueker, Adding range restriction capability to dynamic data structures, J. ACM 32 (1985) 597–617. URL: https://doi.org/10.1145/3828.3839. doi:10.1145/3828.3839.

[14] M. J. Sax, G. Wang, M. Weidlich, J. Freytag, Streams and tables: Two sides of the same coin, in: M. Castellanos, P. K. Chrysanthis, B. Chandramouli, S. Chen (Eds.), Proceedings of the International Workshop on Real-Time Business Intelligence and Analytics, BIRTE 2018, Rio de Janeiro, Brazil, August 27, 2018, ACM, 2018.

[15] G. S. Lueker, A data structure for orthogonal range queries, in: 19th Annual Symposium on Foundations of Computer Science (sfcs 1978), IEEE, 1978, pp. 28–34.