Spatio-Temporal Stream Processing in Microsoft StreamInsight

Mohamed Ali[1] Badrish Chandramouli[2] Balan Sethu Raman[1] Ed Katibah[1]

[1]Microsoft Corporation, One Microsoft Way, Redmond WA 98052
[2]Microsoft Research, One Microsoft Way, Redmond WA 98052
{mali, badrishc, sethur, edwink}@microsoft.com

Abstract

Microsoft StreamInsight is a platform for developing and deploying streaming applications. StreamInsight embraces a temporal stream model to unify and further enrich query language features, handle imperfections in event delivery and define consistency guarantees on the output. With its extensibility framework, StreamInsight enables developers to integrate their domain expertise within the query pipeline as user defined functions, operators and aggregates. Also, the Microsoft SQL Server Spatial Library delivers comprehensive spatial support that enables organizations to seamlessly consume, use, and extend location-based data.

This paper covers two approaches to support spatio-temporal stream processing in StreamInsight. First, the paper describes the ongoing effort at Microsoft SQL Server to bring together the temporal aspect of StreamInsight and the spatial support of the SQL Spatial Library, through the extensibility framework, to deliver an end-to-end solution for location-aware and geospatial data streaming applications. Second, the paper provides the future vision for supporting spatial attributes natively within the pipeline of the stream query processor.

1 Introduction

Microsoft StreamInsight (StreamInsight, for brevity) is a platform for stream query processing. Thanks to its real-time low-latency output, StreamInsight monitors, analyzes and correlates stream data from multiple sources to extract meaningful patterns and trends. StreamInsight adopts a deterministic stream model that leverages a temporal algebra as the underlying basis for processing long-running continuous queries. In most streaming applications, continuous query processing demands the ability to cope with high input rates that are characterized by imperfections in event delivery (i.e., incomplete or out-of-order data). StreamInsight is architected to handle imperfections in event delivery and to provide consistency guarantees on the resultant output. Such consistency gurantees place correctness measures on the output that has been generated so far, given the fact that late and out-of-order stream events are still in transient.

Data stream systems have been widely adopted across multiple business domains. Because domain expertise is the outcome of focused experience in a specific business sector over years, it is hard to expect that a

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

single streaming engine (out-of-the-box without any specialization) can address the requirements of all domains. However, a streaming system is expected to have an extensibility mechanism that seamlessly integrates domain-specific logic into the query pipeline. Hence, StreamInsight has been designed to be an extensible system that accepts and executes user defined modules (*UDMs*) as part of the continuous query plan.

Meanwhile, Microsoft SQL Server Spatial Library [1] (SQL Spatial Library, for brevity) provides an easy to use, robust and high performance environment for persisting and analyzing spatial data. SQL Spatial Library provides data type support for point, line and polygon objects. Also, various methods are provided to handle these spatial data types. SQL Spatial Library adheres to the *Open Geospatial Consortium Simple Feature Access* specification [2] and is provided as part of the SQL Server Types Library.

This paper tackles two approaches for the spatio-temporal stream processing within StreamInsight: an extensibility approach and a native support approach. The extensibility approach combines the values of the StreamInsight extensibility framework and the SQL Spatial Library by giving the UDM writers the ability to invoke the library methods within their code. Alternatively, the native support approach deals with spatial attributes as first class citizens, reasons about the spatial properties of incoming events and, more interestingly, provides consistency guarantees over space as well as time. This paper overviews both approaches and highlights the pros and cons of each approach. The remainder of this paper is organized as follows. Section 2 provides basic background on StreamInsight. Section 3 presents the extensibility approach while Section 4 presents the native support approach for stream processing . Section 5 concludes the paper.

2 Background

A *Data Stream Management System* (*DSMS*) [3, 7, 8, 13] enables applications to issue long-running *continuous queries* (*CQs*) that monitor and process streams of data in real time. DSMSs are used for efficient real-time data processing in many applications. In this paper, we focus on Microsoft StreamInsight [4, 6], which is a DSMS based on the CEDR [5] research project.

2.1 Streams and Events

A *physical stream* is a sequence of events. An event $e_i = \langle p, c \rangle$ is a notification from the outside world that contains: (1) a payload $p = \langle p_1, \ldots, p_k \rangle$, and (2) a control parameter c that provides metadata. The control parameter includes an event generation time, and a duration that indicates the period of time over which an event can influence output. We capture these by defining $c = \langle \mathsf{LE}, \mathsf{RE} \rangle$, where the interval $[\mathsf{LE}, \mathsf{RE})$ specifies the period (or lifetime) over which the event contributes to output. The left endpoint (LE) of this interval, also called start time, is the application time of event generation, also called the event timestamp. Assuming the event lasts for x time units, the right endpoint of an event, also called end time, is simply $\mathsf{RE} = \mathsf{LE} + x$.

2.1.0.4 Compensations StreamInsight allows users to issue compensations (or corrections) for earlier reported events, by the notion of retractions [5, 10, 11], which indicates a modification of the lifetime of an earlier event. This is supported by an optional third control parameter $\mathsf{RE}_{\mathsf{new}}$, that indicates the new right endpoint of the corresponding event. Event deletion (called a full retraction) is expressed by setting $\mathsf{RE}_{\mathsf{new}} = \mathsf{LE}$ (i.e., zero lifetime).

2.1.0.5 Canonical History Table A *Canonical History Table* (*CHT*) is the logical representation of a stream. Each entry in a CHT consists of a lifetime (LE and RE) and the payload. All times are application times, as opposed to system times. Thus, StreamInsight models a data stream as a time-varying relation, motivated by early work on temporal databases by Jensen and Snodgrass [9]. Table 3 shows an example CHT. This CHT can be derived from the actual physical events (either new inserts or retractions) with control parameter

ID	LE	RE	Payload
e_0	1	5	P_1
e_1	4	9	P_2

Table 3: Canonical History Table.

ID	Type	LE	RE	$\mathsf{RE_{new}}$	Payload
e_0	Insertion	1	∞	-	P_1
e_0	Retraction	1	∞	10	P_1
e_0	Retraction	1	10	5	P_1
e_1	Insertion	4	9	-	P_2

Table 4: Physical stream corresponding to CHT.

$c = \langle \mathsf{LE}, \mathsf{RE}, \mathsf{RE_{new}} \rangle$. For example, Table 4 shows one possible physical stream with an associated logical CHT shown in Table 3. Note that a retraction event includes the new right endpoint of the modified event. The CHT (Table 3) is derived by matching each retraction in the physical stream (Table 4) with its corresponding insertion and adjusting RE of the event accordingly.

2.1.0.6 Detecting Time Progress We need to ensure that an event is not arbitrarily out-of-order; this is realized using time-based punctuations [5, 12, 15]. A time-based punctuation is a special event that is used to indicate time progress. These punctuations are called *Current Time Increments* (or *CTIs*) in StreamInsight. A CTI is associated with a timestamp t and indicates that there will be no future event in the stream that modifies any part of the time axis that is earlier than t. Note that we could still see retractions for events with LE less than t, as long as both RE and $\mathsf{RE_{new}}$ are greater than or equal to t.

2.2 Stream Queries and Operators

A CQ consists of a tree of operators, each of which performs some transformation on its input streams and produces an output stream. In StreamInsight, queries are expressed in LINQ [14]. StreamInsight operators are well-behaved and have clear semantics in terms of their effect on the CHT. This makes the underlying temporal operator algebra deterministic, even when data arrives out-of-order. Data enters the streaming system via *input adapters*, which convert external sources into events that can be processed by the streaming system. Output events exit the system via *output adapters*.

There are two main classes of operators: span-based and window-based. A span-based operator accepts events from an input, performs some computation for each event, and produces output for that event. Examples of span-based operators include filter, project, and temporal join. On the other hand, aggregation operators such as Count, Top-K, Sum, etc. report a result (or set of results) for every unique window, i.e., they are window-based. The result is computed using all events that belong to that window. StreamInsight supports several types of windows: snapshot (sliding), hopping, tumbling, and count-based windows.

Further, one stream can be output to multiple operators using an operator called multicast, while multiple streams are merged using a union operator. StreamInsight allows per-group computation using an operation called Group&Apply, where the same subplan (called the apply branch) is applied in parallel for every group (defined by a grouping key) in a stream. The results of all the groups are merged (using union) as the final operator output. In addition, StreamInsight supports user-defined operators to express custom computations; these are discussed next.

3 The Extensibility Approach

As shown in Figure 1, there are three distinct entities that collaborate to extend a streaming system. The *user defined module* (*UDM*) *writer* is the domain expert who writes and packages the code that implements domain-specific operations as libraries. The *query writer* invokes a UDM as part of the query logic. A query is expected to have one or more UDMs wired together with standard streaming operators (e.g., filter, project, joins). Note that multiple query writers may leverage the same existing repository of UDMs for accomplishing specific

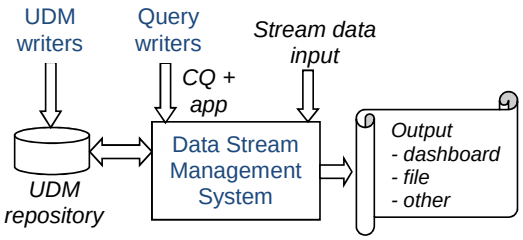


Figure 1: Entities in a streaming extensibility model.

business objectives. The *extensibility framework* is the component that connects the UDM writer and the query writer. The framework executes the UDM logic on demand based on the query to be executed. Thus, the framework provides convenience, flexibility, and efficiency for both the UDM writer and the query writer.

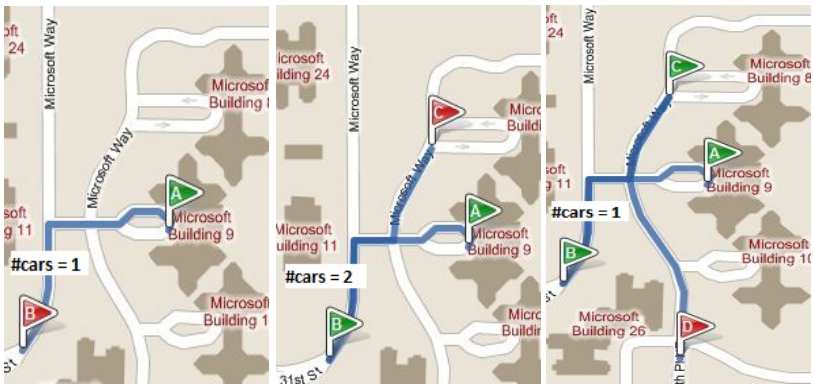
StreamInsight supports three fundamental types of UDMs to the system — *user-defined functions* (*UDFs*), *user-defined aggregates* (*UDAs*), and *user-defined operators* (*UDOs*). UDFs are method calls with arbitrary parameters and a single return value. They can be used wherever expressions (span-based stream operators) occur: filter predicates, projections, join predicates, etc. A UDA is used on top of a window specification (e.g., hopping, snapshot, or count-based window) to aggregate the events contained in each window. A UDA processes a set of events (in a window) and produces a scalar aggregation result (e.g., integer, float, string, etc). UDOs are also used on top of a window, but the result is a set of events with timestamps rather than a scalar value. Note that based on the type of extension, UDMs surface in the StreamInsight LINQ programming model either as method calls (in case of span-based operators) or as extension methods (in case of window-based stream operators).

3.1 Integrating the SQL Spatial Library within the Stream Query Processor

The SQL Spatial Library [1] provides methods to perform spatial operations on spatial data types. Thanks to the extensibility framework of StreamInsight, UDMs that perform intersection, containment, nearest neighbor and shortest route operations are implemented by invoking the appropriate methods from the SQL Spatial Library. Combining the StreamInsight extensibility model and the existing SQL Spatial library provides a solution for spatio-temporal stream processing, increases the value of existing (or *out-of-the-box*) modules, and reduces the cost to develop spatial-oriented streaming applications. Note that the SQL Spatial Library is *not* designed with the continuous stream processing paradigm in mind and, hence, is non-incremental by nature. An appealing future direction is to port the SQL Spatial Library to the streaming domain with the *incremental single-pass* model of stream processing in mind. For example, an intersection query is evaluated incrementally at time $T+1$ by reusing the computed state at time T as the moving object changes its location from time T to time $T+1$.

3.2 Breaking the Optimization Boundaries

Since a UDM is a black box to the query optimizer, the UDM stands as an optimization boundary in the query pipeline. However, there are two approaches to break the optimization boundary in the extensibility approach. The first approach is to work hand-in-hand with the UDM writer who has the option to provide several properties about the UDM through well-defined interfaces to the cost-based query optimizer. Examples of these properties include the selectivity and the expected CPU load (cycles) per input tuple. The optimizer reasons about these properties and shoots for optimization opportunities, e.g., via query plan reorganization and operator migration (in case of multiple StreamInsight instances running in parallel). In the second approach, the system automatically instruments the UDM to measure its average throughout and selectivity. While the second approach relieves the user from specifying the operator properties, the system goes through a learning period during which sub-optimal performance may be observed. With either approach, there is nothing special about spatio-temporal data streams from the extensibility framework's point of view. As we directly leverage the ex-



(a) (b) (c)

Figure 2: An example of native support for spatio-temporal streams.

tensibility framework in order to break the optimization boundary, spatio-temporal stream processing benefits from these optimizations *without* the need to specialize the system for spatio-temporal processing. This solution highlights the value and generality of the extensibility framework.

4 Native Support for Consistent Spatio-temporal Stream Processing

As described in Section 2, the canonical history table maintains the temporal endpoints LE and RE of the event's validity interval over time. Consider a spatial application where we wish to track the movement of objects (e.g., cars). The naive technique is to let the object generate an event periodically as its spatial location (detected using GPS, for example) changes. However, this can result in considerable network traffic and does not scale well as the number of objects increases.

We can instead augment the control parameters with the following information: (1) two location markers SL and DL to denote the start and destination locations of the moving object between times LE and RE, (2) the *route selection policy*, that is the route the system assumes the object takes to travel from the starting location to the destination. Each route selection policy optimizes for one or more criteria (e.g., shortest distance, shortest travel time, highway avoidance, etc) and deterministically decides on the route between the endpoints. (3) detailed temporal information of the planned trip to compute the time at which the object hits a specific point on the route — this could use a simple model such as assuming constant speed over the entire route.

Given the spatial and temporal attributes that describe the object's route over time (for every moving object in the system), the system has the ability to speculate and predict the state of the monitored environment at any point in time. For example, in a traffic management scenario, the system answers queries about the past, current, and future road conditions. Further, it suggests the best driving directions for newly added vehicles by taking future road conditions into consideration. Note that as long as the vehicle is on track, i.e., following the route planned by the system according to the expected speed, there is no need for the vehicle to transmit regular events to the system, which results in reducing transmission load over the wireless network. However, if the vehicle changes its route selection policy, makes an unexpected turn, or stops for some time, the vehicle generates retraction and insertion events to adjust its path. In response to the retraction event, the system updates the result of its CQs and possibly generates compensation events or new speculative output. Further, we could define a *spatio-temporal algebra* with new streaming operators that natively take location into consideration; for example, we may add a spatio-temporal *left-semi-join* operator that accepts a proximity metric and outputs events related to the left input object only when it overlaps in time as well as space (within the proximity metric) with a matching object on the right input. Note that such native support exposes optimization opportunities beyond those possible with black-box approaches.

As a concrete example, consider a CQ that reports the number of cars moving over "*Microsoft Way*". In Figure 2(a), the DSMS receives an insertion event that denotes the intent of car 1 to travel from point *A* to point *B* at time T. In response, it evaluates the query output to be one car traveling over *Microsoft Way* for a specific time interval, i.e., the expected time duration when car 1 is present on Microsoft Way. In Figure 2(b), the DSMS receives another insertion that denotes the intent of car 2 to travel from point *C* to point *B* at time T'. The system modifies the lifetime of the earlier event accordingly, and generates a new event for the duration when 2 cars are present on the road segment. In Figure 2(c), the DSMS receives a retraction that denotes a change in the intent of car 2 from destination *B* to destination *D*. Consequently, it retracts the previously generated event and reverts the count back to one. Although this example shows a simple query over a spatio-temporal stream of two objects, the concept is generalizable to larger road networks and more objects.

5 Conclusions

Microsoft StreamInsight is a high-performance platform for developing streaming applications. In this paper, we presented two approaches to support spatio-temporal data streams in StreamInsight. The first approach utilizes the extensibility framework of StreamInsight to invoke methods from the Microsoft SQL Server Spatial Library. The second approach supports the spatial attributes of moving objects natively as system attributes. The first approach increases the value of existing libraries and components, and gives spatio-temporal stream processing the ability to piggyback on optimizations applied within the extensibility framework as it evolves over time. The second approach extends the temporal algebra adopted by StreamInsight in the spatial direction to provide consistency guarantees over space as well as time, in addition to greater optimization opportunities.

6 Acknowledgments

We would like to thank Michael Kallay for his feedback and his help in using the SQL Spatial Library.

References

[1] SQL Server Spatial Libraries, http://www.microsoft.com/sqlserver/2008/en/us/spatial-data.aspx.

[2] Open Geospatial Consortium, http://www.opengeospatial.org/standards/sfa.

[3] D. Abadi et al. The design of the Borealis stream processing engine. In *CIDR*, 2005.

[4] M. Ali et al. Microsoft CEP Server and Online Behavioral Targeting. In *VLDB*, 2009.

[5] R. Barga et al. Consistent streaming through time: A vision for event stream processing. In *CIDR*, 2007.

[6] B. Chandramouli, J. Goldstein, and D. Maier. On-the-fly progress detection in iterative stream queries. In *VLDB*, 2009.

[7] S. Chandrasekaran et al. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *CIDR*, 2003.

[8] C. Cranor et al. Gigascope: A stream database for network applications. In *SIGMOD*, 2003.

[9] C. Jensen and R. Snodgrass. Temporal specialization. In *ICDE*, 1992.

[10] R. Motwani et al. Query processing, approximation, and resource management in a DSMS. In *CIDR*, 2003.

[11] E. Ryvkina et al. Revision processing in a stream processing engine: A high-level design. In *ICDE*, 2006.

[12] U. Srivastava and J. Widom. Flexible time management in data stream systems. In *PODS*, 2004.

[13] StreamBase Inc. http://www.streambase.com/.

[14] The LINQ Project. http://tinyurl.com/42egdn.

[15] P. Tucker et al. Exploiting punctuation semantics in continuous data streams. *IEEE TKDE*, 2003.