# Streaming Data Integration: Challenges and Opportunities

Nesime Tatbul

*ETH Zurich, Switzerland*
tatbul@inf.ethz.ch

***Abstract*— In this position paper, we motivate the need for streaming data integration in three main forms including across multiple streaming data sources, over multiple stream processing engine instances, and between stream processing engines and traditional database systems. We argue that this need presents a broad range of challenges and opportunities for new research. We provide an overview of the young state of the art in this area and further discuss a selected set of concrete research topics that are currently under investigation within the scope of our MaxStream federated stream processing project at ETH Zurich.**

## I. Introduction

Stream processing systems were born nearly a decade ago, due to the need for low-latency processing of large volumes of highly dynamic, time-sensitive continuous streams of data from sensor-based monitoring devices and alike [1], [2], [3]. "Store-and-Pull" model of traditional data processing systems was simply not suitable for the high performance needs of the streaming applications, where data was much more dynamic than queries and had to be processed on the fly. Consequently, this change of roles between data and queries led to an upside down architecture, and thus, to Stream Processing Engines (SPEs).

Today, SPEs are turning into mature systems and the spectrum of applications that they serve is also widening. As part of this process, we increasingly see the need for using SPEs in combination with a variety of systems and data sources, and sometimes for porting existing applications to run on or with SPEs. However, SPEs still have a long way to go to become established in similar ways as the traditional database systems have become today. For SPEs to realize their full potential and to see stronger adoption, first and foremost, they require critical support for integration in several forms, including across streaming data sources, SPE-SPE, and SPE-DBMS.

First, as in classical data integration, continuous queries on data streams may involve combined processing over a number of streaming data sources. These sources may have different data models (e.g., relational-like or XML-based) and communication interfaces (e.g., HTTP or ODBC). To facilitate the integration of data from these sources, SPEs offer a collection of common adapters as well as SDKs for developing custom adapters. A similar facility is also provided for the output streams generated by the SPEs to enable the flow of outputs to other external systems that may be present down in the processing pipeline. For instance, leading SPE vendors such as Coral8 and StreamBase currently provide around 20 input/output adapters for connecting to messaging systems, networks, financial market data feeds, various visualization and dash-boarding tools [4], [5]. Adapter-based integration is relatively straightforward as long as the source data can be correctly represented in the data model of the SPE, and vice versa; albeit involving some manual work and inflexibility. Furthermore, unlike in the traditional setting, streaming data is usually not stored in advance to be pulled only when needed; but it is continuously streamed through the network (i.e., data is not locked in and owned by the SPE). Consequently, the input sources can be unpredictable and unreliable; the network may introduce delays, losses, disorder into the stream. These imperfections will have to be dealt with during the integration of these data sources at the SPE. On the other hand, adapters should also be very light-weight so as not to become a bottleneck by slowing down the rate at which data can get in and out of the SPE. The last but not the least, integrating streaming sources may also involve a semantic component, when schemas of inputs from different sources have to be mapped to one another and possibly to the input schemas of the already running continuous queries.

Second, there is also a need to integrate multiple SPE instances. These can be identical instances, in which case we have a homogeneous setting which requires using distributed and parallel stream processing techniques (e.g., [6]). These techniques have proven to be effective for increased scalability, high availability, and bringing computation closer to the streaming data sources which might be scattered over the network. However, more work is needed in the light of the more loosely-coupled and elastic setting of the newly emerging cloud computing environments. Another potential scenario which has not been equally explored so far is the case of integrating heterogeneous SPE instances. The main motivation for such a scenario would be to be able to exploit specialized capabilities, models, and strengths of a number of different SPEs, which would fit very well with the highly heterogeneous nature of the current stream processing landscape. Furthermore, this scenario may also arise in large-scale business intelligence applications in extended enterprises with multiple locations and companies (e.g., SAP Live Enterprise [7]). This kind of a scenario is not uncommon in new business structures with out-sourcing, acquisitions, and mergers as well as applications such as supply-chain management, where each participant in the chain might be running its own local SPE of choice and their integration might be necessary for a higher level monitoring of the business operations. The heterogeneous case is much more challenging to handle than the homogeneous case, as it is concerned not only with performance

issues, but also with model and capability differences. It also has brand new challenges compared to traditional database integration because of the strong emphasis on the need for functional integration along with data integration.

Third, there is a growing need for integrating SPEs with traditional database and data warehousing systems. For example, traditional data processing systems play a key role in providing contextual information for operational business intelligence applications [8] and continuous analytics [9]. Today there are a few architectural alternatives emerging to address this need, but more research is needed to explore how much can be accomplished with those approaches and to understand their relative merits both in terms of performance as well as persistence and querying capabilities.

An orthogonal consideration to the above listed integration needs is the way the streaming landscape is evolving today. As the application spectrum widens, SPEs keep on adding new features for increasing their competitive edge, which enlarges the heterogeneity gap further and makes it even harder to produce standards. This situation presents both opportunities as well as challenges for integration: while techniques for integrating heterogeneous SPEs can take over the burden of application development, tuning, and maintenance from the users of these systems, the integration process itself becomes much more complex. It is also important to note though, that this integration process will also likely help the field move forward in reaching consensus and standards.

The need for these different forms of streaming data integration has recently been recognized by several academic groups and commercial companies. Next we provide an overview of the young state of the art in this area.

## II. Current State of the Art

The initial generation of SPEs such as Aurora/Borealis [10], [6], STREAM [11], TelegraphCQ [12] focused on scalable processing of continuous queries over streaming sources only, so as to handle real-time inputs that usually consisted of homogeneous stream tuples (relational) with pre-defined schemas. As such, integration was not their major concern. Nevertheless, these early systems have provided a few basic ingredients, that have recently started being utilized by the current generation of academic and commercial stream processing systems as foundations for various integration tasks. Let us discuss those earlier systems first, and then examine more recent works where integration has become a more explicit goal.

First of all, STREAM's formal continuous query model takes its basis from the well-understood relational model [13]. More specifically, STREAM's Continuous Query Language (CQL) is an extension to SQL:1999. First, it introduces "stream" as a second data type in addition to "relation". Second, in addition to the "relation-to-relation" operations of the relational algebra, CQL introduces "stream-to-relation" operations for constructing windows on streams as well as "relation-to-stream" operations to convert results of relational operations back into the stream data type. Through these mappings, one can essentially reuse most of the relational algebra semantics in a rather straight-forward way. Finally, CQL has also introduced the notion of time into the relational model, which essentially adds the "time-driven" continuous query execution semantics: time advances from $t-1$ to $t$, when all data items up to $t-1$ have been processed. On the other hand, the STREAM system itself does not build on a relational engine, nor it explores the systems issues for use cases that require integrated access to streams and relations. Nevertheless, it provides a good formal model foundation for such a system. And in fact, STREAM's SQL-based language model is being adopted by several commercial systems today, such as StreamBase [14], Truviso [15], and Oracle CEP [16].

Aurora/Borealis provided "table operators" that allowed one to execute SQL statements on Berkeley DB tables for each new tuple that they receive on their input stream. With these operators, one could perform selections, insertions, deletions, and updates on a relational table upon the arrival of a new stream tuple. This is also the model that today's commercial systems like Coral8 [17] and StreamBase [14] are using to implement database access in their systems.

TelegraphCQ implemented its SPE as an extension to the PostgreSQL relational engine [12]. In this regard, it provided a potential platform for tighter integration between relations and streams, and this direction was pursued even further after TelegraphCQ was commercialized into Truviso [15], [9]. Truviso is a so-called "stream-relational" system that provides an integrated query processing approach that runs SQL queries continuously and incrementally over data before it gets stored in the database. Truviso supports queries over tables, streams, and their combinations, and as such, aims at efficiently serving continuous analytics applications.

There are a few other recent systems that follow TelegraphCQ/Truviso's design principle of building a streaming engine out of a relational database engine, but in slightly different ways.

DataCell extends the column-oriented MonetDB relational database for stream processing [18]. Like STREAM's new "stream" data type, a new data type called "basket" is introduced in addition to relational tables. Stream tuples are accumulated in baskets and are accessed by continuous queries in a periodic fashion. Baskets allow batch, out-of-order, and shared processing. The general goal of this project is to explore how much the existing relational technology can be exploited for stream processing. As such, it has the potential to naturally integrate SPE functionality with DBMS functionality as part of its future work.

DejaVu provides declarative pattern matching techniques over live and archived streams of events [19]. The project has two major goals: to efficiently process regular expression-based CEP queries, and to do this over both real-time as well as historical streams. DejaVu extends the MySQL relational database engine and exploits its pluggable storage engine API. Both streaming and historical data sources can be easily attached into a common query engine. As such, DejaVu sets an interesting example for how we can architecturally integrate streams with stores.

All of the techniques and systems summarized above fall under SPE-DBMS integration. There is relatively a smaller body of related work in integrating heterogeneous SPEs and stream data sources, as we present next.

MaxStream is a federated stream processing system that aims at seamlessly integrating multiple autonomous and heterogeneous SPEs together with traditional databases behind a common SQL-based declarative query interface and a common API, in a way to facilitate the incorporation of new functionality and requirements [20], [8], [21]. As in TelegraphCQ/Truviso, DataCell, and DejaVu, MaxStream also builds on and extends a relational database engine infrastructure. However, different from the others, MaxStream is a stream federator, not a full-fledged SPE. It has been designed to serve as a lean, light-weight middleware layer between the client applications and a bunch of underlying SPEs and DBMSs. As such, the goal is to leverage the potentially heterogeneous models and capabilities of the underlying systems rather than implementing a new stream processing engine.

Lastly, there are also a few newly emerging techniques and systems for integrating heterogeneous stream data sources.

The MDQ (Mapping Data to Queries) technique maps incoming data streams of potentially different formats and schemas to the continuous queries that should process them [22]. These queries may be written against schemas different from the inputs'. MDQ uses a set of schema mapping rules to efficiently decide at run time, which data items should be mapped to which queries. This technique would be quite useful to flexibly process data streams with heterogeneous schemas.

The ASPEN project is building a data integration platform for combining data from a variety of sources including databases, web, sensor networks, and other streaming data sources [23], [24]. The platform will serve as a single data access layer and will explore query optimization techniques for federations of stream processors that might be specialized for different types of sources and processing environments.

Despite the growing number of academic and commercial work in streaming data integration as summarized above, we are still at the very beginning and more work needs to be done to investigate the rich opportunities that this research area presents. Next, we will sketch a selection of important research challenges that lie on the path ahead of us.

## III. Selected Research Challenges

### A. Model Issues in SPE-SPE Integration

One important problem today is the lack of clean semantic models for defining streams and continuous queries to process them. There is no agreement even on the definition of basic terms such as "stream" and "window". Although CQL lays a good initial foundation for defining a formal continuous query model over streams and relations [13], it is one of the many possible points in the space of semantic models available out there. There are currently several tens of different SPEs and they widely vary in their data and query models, APIs, functionalities, and optimization capabilities. This heterogeneous and continuously evolving nature of today's streaming landscape not only introduces complexities in choosing the right engine for a given application, but also makes application development and maintenance hard. The need for standardization has recently been acknowledged and a few initiatives in this direction have been started [25], [26]. The former was a proposal on a batch-driven model in order to unify the tuple- and time-driven execution models of two specific SPEs (StreamBase and Oracle CEP, respectively), while the latter was a proposal for adding pattern matching constructs to the SQL standard. However, the whole problem is much more complex since a variety of subtle semantic differences in the execution models of a representative set of SPEs must be settled before a SQL standard can emerge.

While current SPEs differ highly in their continuous query capabilities and in their language syntaxes to express those capabilities, the implementation of a common capability can also vary from one SPE to another, due to the differences in these SPEs' query execution models. Capability differences are easier to manage since they expose themselves at the query language syntax level (e.g., an SPE can either support value-based windows or not), whereas execution model differences can be quite puzzling, since they are implicit in the internal implementation of each SPE and do not expose themselves (and therefore cannot be controlled) at the language level. On the other hand, it is important to be able to analyze and understand different systems' capabilities and execution models in comparison to one another, in reaching a formal model that is general and flexible enough to capture and explain a wide range of different behaviors that are commonplace for a core set of streaming applications. In our ongoing work in this area, we have been working on a formal model to analyze the window execution semantics of a collection of SPEs [27]. Such a model can be useful not only for analyzing the behaviors of these SPEs, but also for building a uniform model for our MaxStream federated stream processing system [20], as well as for helping with future standardization efforts in this area.

### B. Optimization Issues in SPE-SPE & SPE-DBMS Integration

In an integrated stream processing system, one of the most important questions is how to optimize the processing of a given set of continuous queries across multiple SPE instances and databases. Both performance and capability differentiators of these systems must be taken into account. The search space and cost metrics suitable for such an optimization problem need to be defined accordingly. In defining the search space, we can consider several factors: Can a given query be split across multiple SPEs or should it be processed on one SPE in its entirety? If so, which SPE instance should be chosen? This decision can also depend on the current loads of the running SPEs as well as the actual workload running on those SPEs, since continuous queries may have common processing needs and present much opportunity for shared computation. Furthermore, since queries are often times long-lived, during the lifetime of a query, certain workload and data distribution parameters may change. In this case, initial optimization decisions about query decomposition and allocation may have

to be revised. Adapting query processing to changing run-time conditions is more critical in a continuous query environment than in traditional ones, and this applies to their integration as well. In general, one can tackle optimization issues along three orthogonal dimensions: (i) hybrid queries that involve joins between streaming sources and database tables, (ii) homogeneous SPE setups with dynamic workload, and (iii) heterogeneous SPE setups with capability differences.

### C. Transactional Issues in SPEs & SPE-DBMS Integration

Another interesting research question is how to integrate traditional databases, for which clear notions of transactions and transactional properties exist, with SPEs for which no such notions have been properly defined to date. In SPEs, transactional-like concepts have only found use in high availability and fault tolerance scenarios. These scenarios focused on defining different degrees of recovery semantics for the query outputs and techniques for reducing the recovery time [28]. However, in integrated settings, other transactional properties (e.g., isolation rules) can also be important. For example, how to schedule updates to a database table while it is concurrently being joined with a continuous stream of tuples? Defining such rules can be quite tricky due to a fundamental difference in how one interacts with an SPE vs. with a DBMS: the former is data-oriented whereas the latter is operation-oriented. An integrated stream processing system must find the right abstractions to bridge this difference. But before this can be done, a robust theory for transactional stream processing must be developed.

## IV. Wrap-up

In this paper, we tried to motivate the need for streaming data integration in three main forms: streaming data source integration, SPE-SPE integration, and SPE-DBMS integration. Then we provided an overview of the related work in this broad area. Finally, we briefly presented a selection of research topics that we have been investigating as part of our own ongoing research projects at ETH Zurich. We believe that streaming data integration is a rich research area with many open challenges as well as opportunities, for which the previous work to date has only scratched the surface.

**Acknowledgements.** We would like to thank Laura Haas, Donald Kossmann, Renee Miller, and the MaxStream team for their comments and feedback.

## References

[1] S. Babu and J. Widom, "Continuous Queries over Data Streams," *ACM SIGMOD Record*, vol. 30, no. 3, September 2001.

[2] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik, "Monitoring Streams - A New Class of Data Management Applications," in *VLDB Conference*, Hong Kong, China, August 2002.

[3] S. Chandrasekaran and M. J. Franklin, "Streaming Queries over Streaming Data," in *VLDB Conference*, Hong Kong, China, August 2002.

[4] Coral8, "Coral8 Adapters," http://www.coral8.com/products/adapters.html.

[5] StreamBase, "StreamBase Adapters," http://www.streambase.com/products-StreamBaseAdapters.htm.

[6] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Çetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik, "The Design of the Borealis Stream Processing Engine," in *CIDR Conference*, Asilomar, CA, January 2005.

[7] Live Enterprise, "SAP Live Enterprise," http://wiki.sdn.sap.com/wiki/display/Img/Live+Enterprise.

[8] I. Botan, Y. Cho, R. Derakhshan, N. Dindar, L. Haas, K. Kim, and N. Tatbul, "Federated Stream Processing Support for Real-Time Business Intelligence Applications," in *VLDB International Workshop on Enabling Real-Time for Business Intelligence (BIRTE'09)*, Lyon, France, August 2009.

[9] M. J. Franklin, S. Krishnamurthy, N. Conway, A. Li, A. Russakovsky, and N. Thombre, "Continuous Analytics: Rethinking Query Processing in a Network-Effect World," in *CIDR Conference*, Asilomar, CA, January 2009.

[10] D. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora: A New Model and Architecture for Data Stream Management," *VLDB Journal, Special Issue on Best Papers of VLDB 2002*, vol. 12, no. 2, August 2003.

[11] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma, "Query Processing, Approximation, and Resource Management in a Data Stream Management System," in *CIDR Conference*, Asilomar, CA, January 2003.

[12] S. Chandrasekaran, A. Deshpande, M. Franklin, J. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. Shah, "TelegraphCQ: Continuous Dataflow Processing for an Uncertain World," in *CIDR Conference*, Asilomar, CA, January 2003.

[13] A. Arasu, S. Babu, and J. Widom, "The CQL Continuous Query Language: Semantic Foundations and Query Execution," *VLDB Journal*, vol. 15, no. 2, June 2006.

[14] StreamBase, "StreamBase Systems, Inc." http://www.streambase.com/.

[15] Truviso, "Truviso, Inc." http://www.truviso.com/.

[16] Oracle CEP, "Complex Event Processing in the Real World," http://www.oracle.com/technologies/soa/docs/oracle-complex-event-processing.pdf.

[17] Coral8, "Coral8, Inc." http://www.coral8.com/.

[18] E. Liarou, R. Goncalves, and S. Idreos, "Exploiting the Power of Relational Databases for Efficient Stream Processing," in *EDBT Conference*, Saint Petersburg, Russia, March 2009.

[19] N. Dindar, B. Güç, P. Lau, A. Özal, M. Soner, and N. Tatbul, "DejaVu: Declarative Pattern Matching over Live and Archived Streams of Events (Demonstration)," in *ACM SIGMOD Conference*, Providence, RI, June 2009.

[20] I. Botan, Y. Cho, R. Derakhshan, N. Dindar, L. Haas, K. Kim, C. Lee, G. Mundada, M.-C. Shan, N. Tatbul, Y. Yan, B. Yun, and J. Zhang, "Design and Implementation of the MaxStream Federated Stream Processing Architecture," ETH Zurich, Computer Science, Tech. Rep. 632, June 2009, ftp://ftp.inf.ethz.ch/pub/publications/tech-reports/6xx/632.pdf.

[21] I. Botan, Y. Cho, R. Derakhshan, N. Dindar, A. Gupta, L. Haas, K. Kim, C. Lee, G. Mundada, M. Shan, N. Tatbul, Y. Yan, B. Yun, and J. Zhang, "A Demonstration of the MaxStream Federated Stream Processing System (Demonstration)," in *IEEE ICDE Conference*, Long Beach, CA, March 2010.

[22] M. Hentschel, D. Kossmann, D. Florescu, L. Haas, T. Kraska, and R. J. Miller, "Scalable Data Integration by Mapping Data to Queries," ETH Zurich, Computer Science, Tech. Rep. 633, July 2009, ftp://ftp.inf.ethz.ch/pub/publications/tech-reports/6xx/633.pdf.

[23] M. Liu, S. R. Mihaylov, Z. Bao, M. Jacob, Z. G. Ives, B. T. Loo, and S. Guha, "SmartCIS: Integrating Digital and Physical Environments (Demonstration)," in *ACM SIGMOD Conference*, Providence, RI, June 2009.

[24] S. R. Mihaylov, M. Jacob, Z. G. Ives, and S. Guha, "A Substrate for In-Network Sensor Data Integration," in *VLDB Workshop on Data Management for Sensor Networks (DMSN)*, Auckland, New Zealand, August 2008.

[25] N. Jain, S. Mishra, A. Srinivasan, J. Gehrke, J. Widom, H. Balakrishnan, U. Çetintemel, M. Cherniack, R. Tibbetts, and S. Zdonik, "Towards a Streaming SQL Standard," in *VLDB Conference*, Auckland, New Zealand, August 2008.

[26] F. Zemke, A. Witkowski, M. Cherniack, and L. Colby, "Pattern Matching in Sequences of Rows, Tech. Rep. ANSI Standard Proposal, July 2007.

[27] I. Botan, R. Derakhshan, N. Dindar, L. Haas, R. Miller, and N. Tatbul, "SECRET: A Model for Analysis of the Execution Semantics of Stream Processing Systems," November 2009 (under conference submission).

[28] M. Balazinska, J.-H. Hwang, and M. A. Shah, "Fault-Tolerance and High Availability in Data Stream Management Systems," in *Encyclopedia of Database Systems*, L. Liu and M. T. Özsu, Eds. Springer, 2009.