Declarative Languages for Big Streaming Data

A database Perspective

Riccardo Tommasini
University of Tartu
riccardo.tommasini@ut.ee

Sherif Sakr
Unversity of Tartu
sherif.sakr@ut.ee

Emanuele Della Valle
Politecnico di Milano
emanuele.dellavalle@polimi.it

Hojjat Jafarpour
Confluent Inc.
hojjat@confluent.com

ABSTRACT

The Big Data movement proposes data streaming systems to tame *velocity* and to enable *reactive* decision making. However, approaching such systems is still too complex due to the paradigm shift they require, i.e., moving from scalable batch processing to continuous data analysis and pattern detection.

Recently, declarative Languages are playing a crucial role in fostering the adoption of Stream Processing solutions. In particular, several key players introduce SQL extensions for stream processing. These new languages are currently playing a central role in fostering the stream processing paradigm shift. In this tutorial, we give an overview of the various languages for declarative querying interfaces big streaming data. To this extent, we discuss how the different Big Stream Processing Engines (BigSPE) interpret, execute, and optimize continuous queries expressed with SQL-like languages such as `KSQL`, `Flink-SQL`, and `Spark SQL`. Finally, we present the open research challenges in the domain.

KEYWORDS

Stream Processing, Data Stream Management Systems, Complex Event Processing, Streaming SQL

1 GOALS & OBJECTIVES

The world is accelerating; every day, hour, minute, and second the amount of data that we produce grows quicker. Initially, Big Data systems focused on scalable batch processing, and `MapReduce` [12] led the analytics market for more than a decade.

Recently, with the growing interest for (near) real-time insights, we observed a paradigm-shift, i.e., from *data at rest* and post-hoc analyses, to *data-in-motion* and continuous analyses. Thus, Big Data systems evolved to process streams with low latency and high throughput. Nowadays, the state of the art includes many alternative solutions, such as `Storm`, `Flink`, `Spark Structured Streaming`, and `Kafka Streams`) to name the most prominent ones.

Models for continuous data processing have been around for decades [15]. However, the advent of the Big-Data dramatically increased the popularity of data streams in several application domains. For example, developers aim at implementing efficient and effective analytics on massive flows of data for realizing the `Smart X` phenomena (e.g., Smart Home, Smart Hospital, Smart City). Stream processing systems are designed to support a large class of applications in which data are generated from multiple sources and are pushed asynchronously to servers which are responsible for processing them [13].

To facilitate the adoption, initially, most of the big stream processing systems provided their users with a set of API for implementing their applications. However, recently, the need for declarative stream processing languages has emerged to simplify common coding tasks; making code more readable and maintainable, and fostering the development of more complex applications. Thus, Big Data frameworks (e.g., `Flink` [9], `Spark` [3], `Kafka Streams`[1], and `Storm` [19]) are starting to develop their own SQL-like approaches (e.g., `Flink SQL`[2], `Beam SQL`[3], `KSQL`[4]) to declaratively tame data velocity.

In general, declarative languages are extremely useful when writing the optimal solution is harder than solving the problem itself. Indeed, they leverage compilers to catch programming mistakes and automatically transform and optimize code. They fill the gap between expert and normal users. They also increase the acceptance and usability of the system for the end-users. The aim of this tutorial is to provide an overview of the state-of-the-art of the ongoing research and development efforts in the domain of declarative languages for big streaming data processing. In particular, the goals of the tutorial are:

- providing an overview of the fundamental notions of processing streams with declarative languages;
- outlining the process of developing and deploying stream processing applications;
- offering an overview of state of the art for streaming query languages, with a deep-dive into optimization techniques and examples from prominent systems; and
- presenting open research challenges and future research directions in the field.

The content of this tutorial is highly relevant for EBDT-2020 attendees, as it focuses on database-related aspects that concern SQL-like domain-specific languages for stream processing.

2 TUTORIAL PROGRAM OUTLINE

The tutorial introduces the various approaches for declarative stream processing for Big Data. It is centered on providing motivations, models, and optimization techniques related to declarative Stream Processing languages, e.g., `EPL`, `KSQL`, `Flink-SQL` and `Spark Structured Streaming`. In the following, we provide an overview of program followed by a detailed description of the different lectures. Notably, we aim at preparing also practical examples and exercises for the audience to interact with the presented tools.

[1] https://kafka.apache.org/documentation/streams/
[2] https://ci.apache.org/projects/flink/flink-docs-stable/dev/table/sql.html
[3] https://beam.apache.org/documentation/dsls/sql/overview/
[4] https://www.confluent.io/product/ksql/

Figure 1: Adoption of Declarative Interfaces for Stream Processing.

(1) A Brief History of Stream Processing and Its Models.
(2) Big Stream Processing Engines
   - Apache Flink [9]
   - Apache Spark [22]
   - Apache Kafka and Kafka Streams [6]
   - Esper[5]
(3) Declarative Languages for Stream Processing
   - The Continuous Query Language [2]
   - Flink SQL (i.e.,Apache Calcite [7])
   - Spark SQL [3]
   - KSQL [14]
(4) Conclusion and Research Directions

2.1 A Brief History of Stream Processing

During this part, we will focus on describing the motivations that led to the development of stream processing [20]. Besides, we present the state of the art, surveying from seminal models [5, 8] to the more recent ones [1, 17].

`SECRET` [8] is a model to explain the operational semantics of window-based SPEs. SECRET focuses on modeling stream-to-relation operators using four abstractions:

- the *Scope* function maps an event time instant to the time interval over which the computation will occur.
- the *Content* function maps a processing time instant to the subset of stream elements that occur during the event-time interval identified by the scope function.
- the *Report* dimension identifies the set of conditions under which the window content become visible to downstream operators.
- the *Tick* dimension shows the conditions that cause a report strategy to be evaluated.

The `Dataflow Model` [1] presented a fundamental shift on the approach of stream processing, leaving the user the choice of the appropriate trade-off between correctness, latency and cost. The model describes the processing model of Google Cloud Dataflow. In particular, it operates on streams of *(key, value)* pairs using the following primitives.

- *ParDo* for generic element-wise parallel processing producing zero or more output elements per input.
- *GroupByKey* for collecting data for a given key before sending them downstream for reduction.

Being data unbounded *GroupByKey* needs windowing in order to be able to decide when it can output. Windowing is usually treated as key modifier, this way GroupByKey will group also by window. The model addresses the problem of window completeness, claiming the Watermarking an insufficient mechanism to regulate out-of-order arrival. Therefore, the Dataflow Model introduces *Triggers* to provide multiple answers for any given window. Windows and Triggers are complementary operators. The former determines *when* data are grouped together for processing using event time; the latter determines *where* the results of groupings are emitted in processing time.

The `Stream and Table Duality` [17] includes three notions, i.e., *table*, *table changelog stream* and a *record* stream. The static view of an operator's state is a *table*, updated for each input record and has a primary key attribute. *Record* streams and *changelog* streams are special cases of streams. A *changelog* stream is the dynamic view of the result of an update operation on a table. The semantics of an update is defined over both keys and timestamps. Replaying a table changelog stream allows to materialize the operator result as a table. On the other hand, a *record stream* represents facts instead of updates. A record streams model immutable items and, thus, each record has a unique key. Stream processing operators are divided in *stateless* and *stateful* ones, may have one or multiple input streams and might be defined over special types of input streams only.

2.2 Big Stream Processing Engines

During this section of the tutorial, we provide an overview of the following Big Stream Processing Engines (BigSPE). Figure 1 reports the system publication timeline, indicating in light-grey to that are in the scope of the tutorial.

`Apache Flink` [9] is an open source platform for distributed stream and batch processing. Flink uses a streaming dataflow engine that provides data distribution, communication, and fault tolerance for distributed computations. Apache Flink features two relational APIs - the Table API and SQL - for unified stream and batch processing. Flink's Streaming SQL support is based on Apache Calcite which implements the SQL standard.

`Apache Spark` [3] is a general-purpose cluster computing system. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs. Spark's main abstraction are resilient distributed datasets (RDDs). An RDD is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel.

`Apache Kafka` [21] is a distributed streaming platform. Kafka is run as a cluster on one or more servers, called brokers, that can span multiple datacenters. The Kafka cluster stores streams of records in unbounded append only logs called topics. Each record consists of a key, a value, and a timestamp. *Kafka Streams* is a stream processing library built on top of Apache Kafka producer/consumer APIs. It is based on the Stream/Table duality model, which combines the Dataflow model and CQL.

[5] http://www.espertech.com/

2.3 Declarative Stream Processing Languages

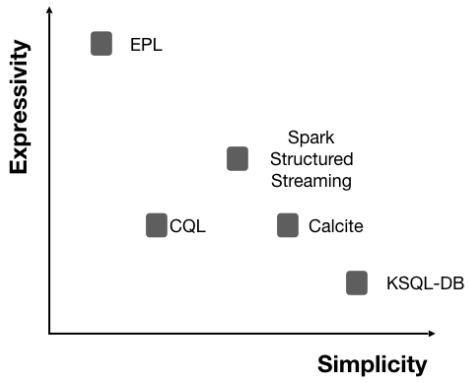

Figure 2: Expressiveness vs Simplicity w.r.t. Declarative Stream Processing Languages

In the context of Big Data Streams, declarative languages suffer from the absence of a shared formal framework that clarifies execution models and time-management approaches. Therefore, existing systems have developed their declarative languages with a primary focus on meeting specific industrial needs.

The `Continuous Query Language` (CQL) [2] is the first attempt to extend relational algebra to process streams of data. CQL defines three families of operators, i.e., Stream-to-Relation (S2R) operators that produce a relation from a stream, Relation-to-Relation (R2R) operators that produce a relation from one or more other relations, and Relation-to-Stream (R2S) operators that produce a stream from a relation. Combining these operators is it possible to define stream-to-stream transformations.

During this section, we survey existing declarative languages for stream processing. Influenced by CQL, BigSPEs' languages try to be as syntactically-close as possible to SQL focusing on usability at the cost of solid design principles like Codd's ones [10]. Following Cugola and Margara's classification [11], we will explain and compare them in terms of expressiveness and simplicity, as shown in Figure 2.

Flink SQL is based on Apache Calcite [7], i.e., a system that provides query parsing, planning, optimization, and execution of SQL queries on top of any data management system. Calcite leaves data storage and management to specialized engines. Flink relies on Calcite to offer a streaming-compliant SQL interface and an advanced query optimization layer. Any language construct used in Flink is also compliant to the SQL standard syntax. Listing 1 shows an example of Flink SQL query that counts the number of views per nation every hour and reporting every minute.

```
SELECT nation, COUNT(*)
FROM pageviews
GROUP BY HOP(rowtime,INTERVAL 1H,INTERVAL 1M), nation
```

Listing 1: Counting Page Views using Flink SQL.

Spark SQL [4] a Spark module for structured data processing. It provides a Dataframe API that can perform relational operations. Moreover, it relies on Catalyst, i.e., an extensible optimizer that allows custom optimization rules. The Dataframe API provides to Catalyst source metadata that allows it to perform optimizations. Structured Streaming [3] ports to Spark some of the Google DataFlow ideas, e.g., the separation between event-time and processing-time. Structured Streaming reuses the Spark SQL execution engine, Catalyst, and the code generator. To support streaming, Structured Streaming add the features to Spark SQL:(1) Triggers to control result reporting; (2) Time extractors to mark a column for event time;(3) Stateful operators to implement complex aggregations. Listing 2 shows the same query of Listing 1, but using Spark SQL.

```
val df = pageviews.groupBy(
  window($"timestamp", "1 hour", "1 minute"),$"nation"
).count()
```

Listing 2: Counting Page Views using Spark SQL.

KSQL [14] is a streaming SQL engine implemented on top of the Kafka Streams API. KSQL is directly based on the Stream Duality model [17]. Thus, it relies on two first-class constructs, i.e., Streams and Tables. To move between these abstractions, KSQL provides powerful stream processing capabilities such as joins, aggregations, event-time windowing, and many more. Listing **??** shows our example pageviews query using KSQL syntax.

```
CREATE TABLE analysis AS SELECT nation, COUNT(*)
FROM pageviews
WINDOW HOPPING (SIZE 1 HOUR, ADVANCE BY 1 MINUTE)
GROUP BY nation;
```

Listing 3: Counting Page Views using KSQL label

The presenters go in-depth regarding the languages above, focusing in particular on the following constructs [18]: (i) Filters and stateless operations; (ii) table-stream and stream-to-stream joins; (iii) windowing, aggregates, and stateful computations.

Finally, presenters will comment on the design of the languages above w.r.t. the Codd's principles:

- *Minimality*, i.e., a language should provide only a small set of needed language constructs so that different language constructs cannot express the same meaning;
- *Symmetry*, i.e., a language should ensure that the same language construct always expresses the same semantics regardless of the context it is used in; and
- *Orthogonality*, i.e., a language should guarantee that every meaningful combination its constructs is applicable.

2.4 Conclusion and Research Directions

This section closes the tutorial indicating ongoing research and industrial works. Moreover, we will ask the audience about the fact that SQL was not intended to be used with unbounded streams of data nor with the continuous semantics required to process them. Nevertheless, existing BigSPE are adopting it and a question naturally raises: *Can we consider BigSPE as databases?*

3 LEARNING OUTCOMES

The tutorial targets researchers, knowledge workers, and practitioners who want to understand the current state-of-the-art as well as the future directions of stream processing.

This tutorial includes relevant technologies and topics for people from IoT, as well as social media, pervasive health, ans oil industry, who have to analyze in massive amounts of streaming data. After attending this tutorial, the audience will have:

- Good understanding of the fundamental and main concepts of stream processing its models;

- An overview and detailed insight into declarative stream processing languages and the ongoing developments by big data streaming system in this domain;
- An overview of research directions in which the state-of-the-art can be improved.

The material of the tutorial will cover some of the work of the presenters on the topics of the tutorial, including [13, 16, 18]

4 PREVIOUS EDITIONS

The first version of this tutorial was presented at the DEBS 2019 conference[6]. It was a full-day tutorial, and it focused on the processing models behind declarative stream processing languages. With the tutorial, the presenters were invited to publish a companion short paper [18] and a website[7].

5 PRESENTERS & ORGANIZERS

Riccardo Tommasini is a research fellow at the University of Tartu, Estonia. Riccardo did his PhD at the Department of Electronics and Information of the Politecnico di Milano. His thesis, titled "Velocity on the Web", investigates the velocity aspects that concern the Web environment. His research interests span Stream Processing, Knowledge Graphs, Logics and Programming Languages. Riccardo's tutorial activities comprise Stream Reasoning Tutorials at ISWC 2017, ICWE 2018, ESWC 2019, and TheWebConf 2019, and DEBS 2019.

Sherif Sakr is the Head of Data Systems Group at the Institute of Computer Science, University of Tartu, Estonia. He received his PhD degree in Computer and Information Science from Konstanz University, Germany in 2007. He is currently the Editor-in-Chief of the Springer Encyclopedia of Big Data Technologies. His research interest include data and information management, big data processing systems, big data analytics and data science. Prof. Sakr has published more than 150 research papers in international journals and conferences. He delivered several tutorials in various conferences including WWW'12, IC2E'14, CAiSE'14, EDBT Summer School 2015, . The 2nd ScaDS International Summer School on Big Data 2016, The 3rd Keystone Training School on Keyword search in Big Linked Data 2017, DEBS 2019 and ISWC 2019.

Emanuele Della Valle is an Assistant Professor at the Department of Electronics and Information of the Politecnico di Milano. His research interests covered Big Data, Stream Processing, Semantic technologies, Data Science, Web Information Retrieval, and Service Oriented Architectures. His work on Stream Reasoning research filed was applied in analysing Social Media, Mobile Telecom and IoT data streams in collaboration with Telecom Italia, IBM, Siemens, Oracle, Indra, and Statoil. Emanuele presented several Stream Reasoning related tutorials at SemTech 2011, ESWC 2011, ISWC 2013, ESWC 2014, ISWC 2014, ISWC 2015, ISWC 2016, DEBS 2016, ISWC 2017 and KR 2018.

Hojjat Jafarpour is a Software Engineer and the creator of KSQL at Confluent. Before joining Confluent he has worked at NEC Labs, Informatica, Quantcast and Tidemark on various big data management projects. Hojjiat earned his PhD in computer science from UC Irvine, where he worked on scalable stream processing and publish/subscribe systems.

[6] http://debs2019.org/
[7] http://streaminglangs.io/

Acknowledgements.

The work of Sherif Sakr is funded by the European Regional Development Funds via the Mobilitas Plus programme (grant MOBTT75).

The work of Riccardo Tommasini is funded by the European Regional Funds through IT Academy programme.

REFERENCES

[1] Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, and Sam Whittle. The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *PVLDB*, 8(12):1792–1803, 2015.
[2] Arvind Arasu, Shivnath Babu, and Jennifer Widom. The CQL continuous query language: semantic foundations and query execution. *VLDB J.*, 15(2):121–142, 2006.
[3] Michael Armbrust, Tathagata Das, Joseph Torres, Burak Yavuz, Shixiong Zhu, Reynold Xin, Ali Ghodsi, Ion Stoica, and Matei Zaharia. Structured streaming: A declarative api for real-time applications in apache spark. In *SIGMOD*, 2018.
[4] Michael Armbrust, Reynold S. Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan, Michael J. Franklin, Ali Ghodsi, and Matei Zaharia. Spark SQL: relational data processing in spark. In *SIGMOD Conference*, pages 1383–1394. ACM, 2015.
[5] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *PODS*, pages 1–16. ACM, 2002.
[6] Edmon Begoli, Tyler Akidau, Fabian Hueske, Julian Hyde, Kathryn Knight, and Kenneth Knowles. One SQL to rule them all - an efficient and syntactically idiomatic approach to management of streams and tables. In *SIGMOD Conference*, pages 1757–1772. ACM, 2019.
[7] Edmon Begoli, Jesús Camacho-Rodríguez, Julian Hyde, Michael J. Mior, and Daniel Lemire. Apache calcite: A foundational framework for optimized query processing over heterogeneous data sources. In *SIGMOD Conference*, pages 221–230. ACM, 2018.
[8] Irina Botan, Roozbeh Derakhshan, Nihal Dindar, Laura M. Haas, Renée J. Miller, and Nesime Tatbul. SECRET: A model for analysis of the execution semantics of stream processing systems. *PVLDB*, 3(1):232–243, 2010.
[9] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 36(4), 2015.
[10] Edgar F Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.
[11] Gianpaolo Cugola and Alessandro Margara. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.*, 44(3):15:1–15:62, 2012.
[12] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
[13] Martin Hirzel, Guillaume Baudart, Angela Bonifati, Emanuele Della Valle, Sherif Sakr, and Akrivi Vlachou. Stream processing languages in the big data era. *SIGMOD Record*, 47(2):29–40, 2018.
[14] Hojjat Jafarpour and Rohan Desai. KSQL: streaming SQL engine for apache kafka. In *EDBT*, pages 524–533. OpenProceedings.org, 2019.
[15] Shanmugavelayutham Muthukrishnan et al. Data streams: Algorithms and applications. *Foundations and Trends® in Theoretical Computer Science*, 1(2):117–236, 2005.
[16] Sherif Sakr. *Big data 2.0 processing systems: a survey*. Springer, 2016.
[17] Matthias J. Sax, Guozhang Wang, Matthias Weidlich, and Johann-Christoph Freytag. Streams and tables: Two sides of the same coin. In *BIRTE*, pages 1:1–1:10. ACM, 2018.
[18] Riccardo Tommasini, Sherif Sakr, Marco Balduini, and Emanuele Della Valle. An outlook to declarative languages for big streaming data. In *DEBS*, pages 199–202. ACM, 2019.
[19] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, et al. Storm@ twitter. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 147–156. ACM, 2014.
[20] Emanuele Della Valle, Stefano Ceri, Frank van Harmelen, and Dieter Fensel. It's a streaming world! reasoning upon rapidly changing information. *IEEE Intelligent Systems*, 24(6):83–89, 2009.
[21] Guozhang Wang, Joel Koshy, Sriram Subramanian, Kartik Paramasivam, Mammad Zadeh, Neha Narkhede, Jun Rao, Jay Kreps, and Joe Stein. Building a replicated logging system with apache kafka. *PVLDB*, 8(12):1654–1655, 2015.
[22] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache spark: a unified engine for big data processing. *Commun. ACM*, 59(11):56–65, 2016.