Data Streams

Alessandro Margara, Tilmann Rabl

Synonyms

Information Flows, Event Streams.

Definitions

A data stream is a countably infinite sequence of elements. Different models of data streams exist, that take different approaches with respect to the mutability of the stream and to the structure of stream elements. Stream processing refers to analyzing data streams on-the-fly to produce new results as new input data becomes available. Time is a central concept in stream processing: in almost all models of streams, each stream element is associated with one or more timestamps from a given time domain that might indicate for instance when the element was generated, the validity of its content, or when it became available for processing.

Alessandro Margara
Politecnico di Milano, Milano, Italy, e-mail: alessandro.margara@polimi.it,
Tilmann Rabl
TU Berlin, Berlin, Germany, e-mail: rabl@tu-berlin.de

Overview

A data stream is a countably infinite sequence of elements and is used to represent data elements that are made available over time. Examples are readings from sensors in an environmental monitoring application, stock quotes in financial applications, or network data in computer monitoring applications.

A stream-based application analyzes elements from streams as they become available to timely produce new results and enable fast reactions if needed Babcock et al (2002); Cugola and Margara (2012).

In the last decade, several technologies have emerged to address the processing of high-volume, real-time data without requiring custom code Stonebraker et al (2005). They are commonly referred to as stream processing systems, and they are the topic of this section on “Big Stream Processing”.

In the remainder of this introductory chapter, we overview the main models of data streams and stream processing systems, and we briefly introduce some central concepts in stream processing, namely time and windows. Furthermore we summarize the main requirements of stream processing.

Stream models

Streams can be structured or unstructured. In a structured stream, elements follow a certain format or schema that

allows for additional modelling of the stream. In contrast, unstructured streams can have arbitrary contents often resulting from combining streams from many sources (these are also referred to as event showers or event clouds Doblander et al (2014)).

There are three major models for structured streams, which differentiate in how the elements of the stream are related to and influence each other: the turnstile model, the cash register model, and the time series model. The most general model is the *turnstile model*. In this model, the stream is modelled as a vector of elements and each element s_i in the stream is an update (increment or decrement) to an element of the underlying vector. The size of the vector in this model is the domain of the stream elements. This model is also the model typically used in traditional database systems, where there are inserts, deletes, and updates to the database. In the *cash register model*, elements in the stream are only additions to the underlying vector, but elements can never leave the vector again. This is similar to databases recording the history of relations. Finally, the *time series model* treats every element in the stream s_i as a new independent vector entry. As a result the underlying model is a constantly increasing vector and generally unbounded vector. Because each element can be processed individually in this model, it is frequently used in current stream processing engines.

Time

Time is a central concept in many stream processing applications, either because they are concerned with updating their view of the world by taking into account *recent* data received from streams, or because they aim to detect temporal trends in the input streams.

For this reason, in most models of stream and stream processing data elements are associated with some timestamp from a given time domain. Common semantics of time include *event time*, which is the time when the element was produced, and *processing time*, which is the time when the stream processing system starts processing the element Akidau et al (2015).

Different time semantics introduce different problems in terms of order and synchronization. Intuitively, while event time and processing time should ideally be equal, in practice unsynchronized clocks at the producers as well as variable communication and processing delays produce a skew between event time and processing time which is not only non-zero, but also highly variable Akidau (2015). On the one hand, processing elements in event time order is necessary in many application scenarios. On the other hand, this requires waiting for out-of-order elements and reorder them before processing. This is a non-trivial problem, which is analyzed in detail in the chapter related to “Time management”.

Windows

Windows are one of the core building blocks of virtually all stream processing systems. They define bounded portions of elements over an unbounded stream. They are used to perform computations that would be impossible (non terminat-

ing) in the case of unbounded data, such as the computing the average value of all the elements in a stream of numbers.

The most common types of windows are *count-based* and *time-based* windows. The former define their size in terms of the number of elements that they include, while the latter define their size in terms of a time frame, and include all the elements with a timestamp included in that time frame.

In both cases, we distinguish between *sliding* windows, which continuously advance with the arrival of new elements, thus always capturing new elements, and *tumbling* windows, which can accumulate multiple elements before moving Botan et al (2010).

More recently, new types of windows have been defined to bettr capture the needs of applications. They include *session* and *data-defined* windows that are variable in size and define their boundaries based on the data elements: for instance, in a software monitoring application, a window can include all and only the elements that refer to a session opened by a specific user of that software Akidau et al (2015).

Stream Processing

Several types of stream processing systems exist that are specialized on different types of computations Cugola and Margara (2012).

Data stream management systems are designed to update the answers of *continuous queries* as new data becomes available. They typically adopt declarative abstractions similar to traditional database query languages that they enrich with constructs such as windows to deal with the unbounded nature of the input data Arasu et al (2006).

Complex event processing systems, instead, aim to detect (temporal) patterns over the input streamd of elements Etzion and Niblett (2010); Luckham (2001). These systems are discussed in the chapter on "Event recognition".

Modern Big Data stream processing systems, such as Flink Carbone et al (2015), provide general operators to transform input streams into output streams, thus offering the possibility to implement heterogeneous streaming applications by integrating these operators. These systems are designed to scale to multiple processing cores and computing nodes, and persist intermediate results for fault tolerance. Flink and other systems are discussed in several chapters within this section of the encyclopedia. The "Introduction to stream processing" chapter overviews the main classes of algorithms for stream processing.

General Requirements of Stream Processing

Data streams could theoretically be stored and processed with traditional database systems or other analytics solutions. However, real-time processing of streams introduces specific requirements that stream processing systems need to satisfy Stonebraker et al (2005). Although the following requirements are not necessarily fulfilled by all current stream processing systems, they are frequently necessary in stream processing applications.

In order to do advanced analytics on streams, which involve more than a

single operation, a stream processing system needs to integrate (pipeline) basic operations with each other. This can be done by (micro-)batching the stream, that is, splitting it in small chuncks such that each operator processes a chunck at a time, or by true streaming, where each element is processed individually.

Because streams are often generated in a distributed fashion, for example in sensor networks, a streaming system must handle small inconsistencies in the stream, such as out-of-order data, missing values, or delayed values. This is related to the topic of time management, discussed earlier. Even if a stream contains small inconsistencies, the result of the stream processing needs to be predictable. This demands for clear semantics of the operations available and for robust mechanisms that guarantee deterministic results.

A high-level interface to specify stream processing jobs is desirable. The section on stream processing languages and abstractions gives an overview of current solutions.

An area of active research is the combination of streaming and batch data. Interestingly, this combination forms the missing link between database systems and streaming systems. Early attempts to address this challenge have led to the idea of a lambda architecture, which consists of separate systems for streaming and batch processing Marz and Warren (2015). However, especially problems with robustness and model mismatches call for an integrated solution. Current streaming systems address this by enabling advanced state management Carbone et al (2017) Affetti et al (2017).

Future Directions of Research

Streaming is a very active area of research. Each of the following chapters will highlight interesing future directions of research. However, there are several topics that have found limited attention in current research but will be increasingly important for future applications and systems. Here, we will highlight two areas that found little attention so far.

Current stream processing systems frequently differentiate themselves from database systems in that they see all streams and all analytics jobs as conceptionally unbounded. However, in many scenarios, streams as well as the analytics jobs on them are bounded. Furthermore, both can arrive at high frequencies, which is a challenge for current systems.

Another challenge, not solved by current systems is transactional guarantees. Current stream processing systems provide basic guarantees like processing each stream element exactly once or at least once, but have no concept of transactions that span multiple stream elements or operations Affetti et al (2017).

Summary

In this chapter, we presented an overview on stream processing and introduced some concepts and terminology. In the following chapters, further details will be given on important topics in big stream processing.

References

Affetti L, Margara A, Cugola G (2017) Flowdb: Integrating stream processing and consistent state management. In: Proceedings of the International Conference on Distributed and Event-based Systems, ACM, DEBS '17, pp 134–145, DOI 10.1145/3093742.3093929

Akidau T (2015) The workld beyond batch: Streaming 101

Akidau T, Bradshaw R, Chambers C, Chernyak S, Fernández-Moctezuma RJ, Lax R, McVeety S, Mills D, Perry F, Schmidt E, Whittle S (2015) The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. VLDB 8(12):1792–1803, DOI 10.14778/2824032.2824076

Arasu A, Babu S, Widom J (2006) The cql continuous query language: Semantic foundations and query execution. VLDB 15(2):121–142, DOI 10.1007/s00778-004-0147-z

Babcock B, Babu S, Datar M, Motwani R, Widom J (2002) Models and issues in data stream systems. In: Proceedings of the Symposium on Principles of Database Systems, ACM, PODS '02, pp 1–16, DOI 10.1145/543613.543615

Botan I, Derakhshan R, Dindar N, Haas L, Miller RJ, Tatbul N (2010) Secret: A model for analysis of the execution semantics of stream processing systems. VLDB 3(1-2):232–243, DOI 10.14778/1920841.1920874

Carbone P, Katsifodimos A, Ewen S, Markl V, Haridi S, Tzoumas K (2015) Apache flink: Stream and batch processing in a single engine. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 36(4)

Carbone P, Ewen S, Fóra G, Haridi S, Richter S, Tzoumas K (2017) State management in apache flink®: Consistent stateful distributed stream processing. Proceedings of VLDB 10(12):1718–1729, DOI 10.14778/3137765.3137777

Cugola G, Margara A (2012) Processing flows of information: From data stream to complex event processing. ACM Computing Surveys 44(3):15:1–15:62, DOI 10.1145/2187671.2187677

Doblander C, Rabl T, Jacobsen HA (2014) Processing big events with showers and streams. In: Rabl T, Poess M, Baru C, Jacobsen HA (eds) Specifying Big Data Benchmarks, Springer, pp 60–71

Etzion O, Niblett P (2010) Event Processing in Action. Manning Publications

Luckham DC (2001) The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley

Marz N, Warren J (2015) Big Data: Principles and Best Practices of Scalable Realtime Data Systems. Manning Publications

Stonebraker M, Çetintemel U, Zdonik S (2005) The 8 requirements of real-time stream processing. SIGMOD Rec 34(4):42–47, DOI 10.1145/1107499.1107504

Cross References

- Management of Time
- Stream Processing Languages and Abstractions
- Introduction to Stream Processing