A Gentle Introduction to Stream Processing

Srinath Perera Follow

Apr 4, 2018 · 8 min read

What is stream Processing?

Stream Processing is a Big data technology. It is used to query continuous data stream and detect conditions, quickly, within a small time period from the time of receiving the data. The detection time

period varies from few milliseconds to minutes. For example, with stream processing, you can receive an alert when the temperature has reached the freezing point, querying data streams coming from a temperature sensor.

It is also called by many names: real-time analytics, streaming analytics, Complex Event Processing, real-time streaming analytics, and event processing. Although some terms historically had differences, now tools (frameworks) have converged under term stream processing. (see this Quora Question for a list of frameworks and last section of this article for history).

It is popularized by Apache Storm, as a "technology like Hadoop but can give you results faster", after which it was adopted as a Big data technology. Now there are many contenders.

Why is stream Processing needed?

Big data established the value of insights derived from processing data. Such insights are not all created equal. Some insights are more valuable shortly after it has happened with the value diminishes very fast with time. Stream Processing enables such scenarios, providing insights faster, often within milliseconds to seconds from the trigger.

Following are some of the secondary reasons for using Stream Processing.

Reasons 1: Some data naturally comes as a never-ending stream of events. To do batch processing, you need to store it, stop data collection at some time and processes the data. Then you have to do the next batch and then worry about aggregating across multiple batches. In contrast, streaming handles neverending data streams gracefully and naturally. You can detect patterns, inspect results, look at multiple levels of focus, and also easily look at data from multiple streams simultaneously.

Stream processing naturally fit with time series data and detecting patterns over time. For example, if you are trying to detect the length of a web session in a never-ending stream (this is an example of trying to detect a sequence). It is very hard to do it with batches as some session will fall into two batches. Stream processing can handle this easily.

If you take a step back and consider, the most continuous data series are time series data: traffic sensors, health sensors, transaction logs, activity logs, etc. Almost all IoT data are time series data. Hence, it makes sense to use a programming model that fits naturally.

Reason 2: Batch processing lets the data build up and try to process them at once while stream processing process data as they come in hence spread the processing over time. Hence stream processing can work with a lot less hardware than batch processing. Furthermore, stream processing also enables approximate query processing via systematic load shedding. Hence stream processing fits naturally into use cases where approximate answers are sufficient.

Reason 3: Sometimes data is huge and it is not even possible to store it. Stream processing let you handle large fire horse style data and retain only useful bits.

Reason 4: Finally, there are a lot of streaming data available (e.g. customer transactions, activities, website visits) and they will grow faster with IoT use cases (all kind of sensors). Streaming is a much more natural model to think about and program those use cases.

However, Stream Processing is also not a tool for all use cases. One good rule of thumb is that if processing needs multiple passes through full data or have random access (think a graph data set) then it is tricky with streaming. One big missing use case in streaming is machine learning algorithms to train models. On the other hand, if processing can be done with a single pass over the data or has temporal locality (processing tend to access recent data) then it is a good fit for streaming.

How to do Stream Processing?

If you want to build an App that handles streaming data and takes real-time decisions, you can either use a tool or build it yourself. The answer depends on how much complexity you plan to handle, how much you want to scale, how much reliability and fault tolerance you need etc.

If you want to build the App yourself, place events in a message broker topic (e.g. ActiveMQ, RabbitMQ, or Kafka), write code to receive

events from topics in the broker (they become your stream) and then publish results back to the broker. Such a code is called an actor.

However, Instead of coding the above scenario from scratch, you can use a stream processing framework to save time. An event stream processor lets you write logic for each actor, wire the actors up, and hook up the edges to the data source(s). You can either send events directly to the stream processor or send them via a broker.

An event stream processor will do the hard work by collecting data, delivering it to each actor, making sure they run in the right order, collecting results, scaling if the load is high, and handling failures. Among examples are Storm, Flink, and Samza. If you like to build the app this way, please check out respective user guides.

Since 2016, a new idea called Streaming SQL has emerged (see article Streaming SQL 101 for details). We call a language that enables users to write SQL like queries to query streaming data as a "Streaming SQL" language. There are many streaming SQL languages on the rise.

Projects such as WSO2 Stream Processor and SQLStreams supported SQL for more than five years

- Apache Storm added support for Streaming SQL in 2016
- Apache Flink added support for Streaming SQL since 2016
- Apache Kafka added support for SQL (which they called KSQL) in 2017,
- Apache Samza added support for SQL in 2017

With Streaming SQL languages, developers can rapidly incorporate streaming queries into their Apps. By 2018, most of the Stream processors supports processing data via a Streaming SQL language.

Let's understand how SQL is mapped to streams. A stream is a table data in the move. Think of a never-ending table where new data appears as the time goes. A stream is such a table. One record or a row in a stream is called an event. But, it has a schema, and behave just like a database row. To understand these ideas, Tyler Akidau's talk at Strata is a great resource.

The first thing to understand about SQL streams is that it replaces tables with streams.

When you write SQL queries, you query data stored in a database. Yet, when you write a Streaming SQL query, you write them on data that is now as well as the data that will come in the future. Hence, streaming SQL queries never ends. Is it a problem? No, it works because the output of those queries are streams. The event will be placed in output streams once the event matched and output events are available right away.

A stream represents all events that can come through a logical channel and it never ends. For example, if we have a temperature sensor in boiler we can represent the output from the sensors as a stream. However, classical SQL ingest data stored in a database table, processes them, and writes them to a database table. Instead, Above query will ingest a stream of data as they come in and produce a stream of data as output. For example, let's assume there are events in the boiler stream

once every 10 minutes. The filter query will produce an event in the result stream immediately when an event matches the filter.

So you can build your App as follows. You send events to stream processor by either sending directly or by via a broker. Then you can write the streaming part of the App using “Streaming SQL”. Finally, you configure the Stream processor to act on the results. This is done by invoking a service when Stream Processor triggers or by publishing events to a broker topic and listening to the topic.

There are many stream processing frameworks available. (See Quora Question: What are the best stream processing solutions out there?).

I would recommend the one I have helped build, WSO2 Stream Processor (WSO2 SP). It can ingest data from Kafka, HTTP requests, message brokers and you can query data stream using a “Streaming SQL” language. WSO2 SP is open source under Apache license. With just two commodity servers it can provide high availability and can handle 100K+ TPS throughput. It can scale up to millions of TPS on top of Kafka and supports multi-datacenter deployments.

Who is using Stream Processing?

In general, stream processing is useful in use cases where we can detect a problem and we have a reasonable response to improve the outcome. Also, it plays a key role in a data-driven organization.

Following are some of the use cases.

- Algorithmic Trading, Stock Market Surveillance,
- Smart Patient Care
- Monitoring a production line
- Supply chain optimizations
- Intrusion, Surveillance and Fraud Detection (e.g. Uber)
- Most Smart Device Applications: Smart Car, Smart Home ..
- Smart Grid—(e.g. load prediction and outlier plug detection see Smart grids, 4 Billion events, throughout in range of 100Ks)
- Traffic Monitoring, Geofencing, Vehicle, and Wildlife tracking—e.g. TFL London Transport Management System
- Sports analytics—Augment Sports with real-time analytics (e.g. this is a work we did with a real football game (e.g. Overlaying realtime analytics on Football Broadcasts)
- Context-aware promotions and advertising
- Computer system and network monitoring
- Predictive Maintenance, (e.g. Machine Learning Techniques for Predictive Maintenance)
- Geospatial data processing

For more discussions about how to use Stream Processing, please refer to 13 Stream Processing Patterns for building Streaming and Realtime

Applications.

History of Stream Processing and its Frameworks

Stream Processing has a long history starting from active databases that provided conditional queries on data stored in databases. One of the first Stream processing framework was TelegraphCQ, which is built on top of PostgreSQL.

Then they grew in two branches.

The first branch is called Stream Processing. These frameworks let users create a query graph connecting the user's code and running the query graph using many machines. Examples are Aurora, PIPES, STREAM, Borealis, and Yahoo S4. These stream processing architectures focused on scalability.

The second branch is called Complex Event Processing. These frameworks supported query languages (such as now we have with Streaming SQL) and concerned with doing efficient matching of events against given queries, but often run on 1–2 nodes. Among examples are ODE, SASE, Esper, Cayuga, and Siddhi. These architectures focused on efficient streaming algorithms.

Stream Processing frameworks from both these branches were limited to academic research or niche applications such as the stock market. Stream processing comes back to limelight with Yahoo S4 and Apache

Storm. It was introduced as “like Hadoop, but real time”. It becomes part of the Big data movement.

In the last five years, these two branches have merged. I have discussed this in detail in an earlier post.

If you like to know more about the history of stream processing frameworks please read Recent Advancements in Event Processing and Processing flows of information: From data stream to complex event Processing.

Hope this was useful. If you enjoyed this post you might also like Stream Processing 101 and Patterns for Streaming Realtime Analytics.