with

Structured Streaming

Structured Streaming – Overview

- Structured Streaming is a scalable and fault-tolerant stream processing engine built on the Spark SQL engine.
- We can express the streaming computation the same way we would express a batch computation on static data.
- The Spark SQL engine will take care of running it incrementally and continuously and updating the final result as streaming data continues to arrive.
- We can use the Dataset/DataFrame API in Scala, Java, Python or R to express streaming aggregations, stream-to-batch joins, etc.
- The computation is executed on the same optimized Spark SQL engine.
- Structured Streaming provides fast, scalable, fault-tolerant, end-to-end exactly-once stream processing without the user having to reason about streaming.

Structured Streaming – Overview

- Internally, by default, Structured Streaming queries are processed using a micro-batch processing engine.
- It processes data streams as a series of small batch jobs thereby achieving end-to-end latencies as low as 100 milliseconds and exactly-once fault-tolerance guarantees.
- However, since Spark 2.3, a new low-latency processing mode called Continuous Processing is introduced.
- It can achieve end-to-end latencies as low as 1 millisecond with at-least-once guarantees.

Ref: https://spark.apache.org/docs/3.1.2/structured-streaming-programming-guide.html#continuous-processing

Structured Streaming – Overview

- When things go wrong in a stream processing application, it is possible to have either lost, or duplicated results.
- Generally streaming data platforms provide at least one if not more of the following guarantees.
- Depending on our requirements we need to use the tools and technology that suits our purpose.

at-most-once delivery

- means that for each message handed for processing, that message is delivered zero or one times; in more casual terms *it means that messages may be lost*.

at-least-once delivery

- means that for each message handed for processing, potentially multiple attempts are made at delivering it, such that at least one succeeds; again, in more casual terms *this means that messages may be duplicated but not lost*.

exactly-once delivery

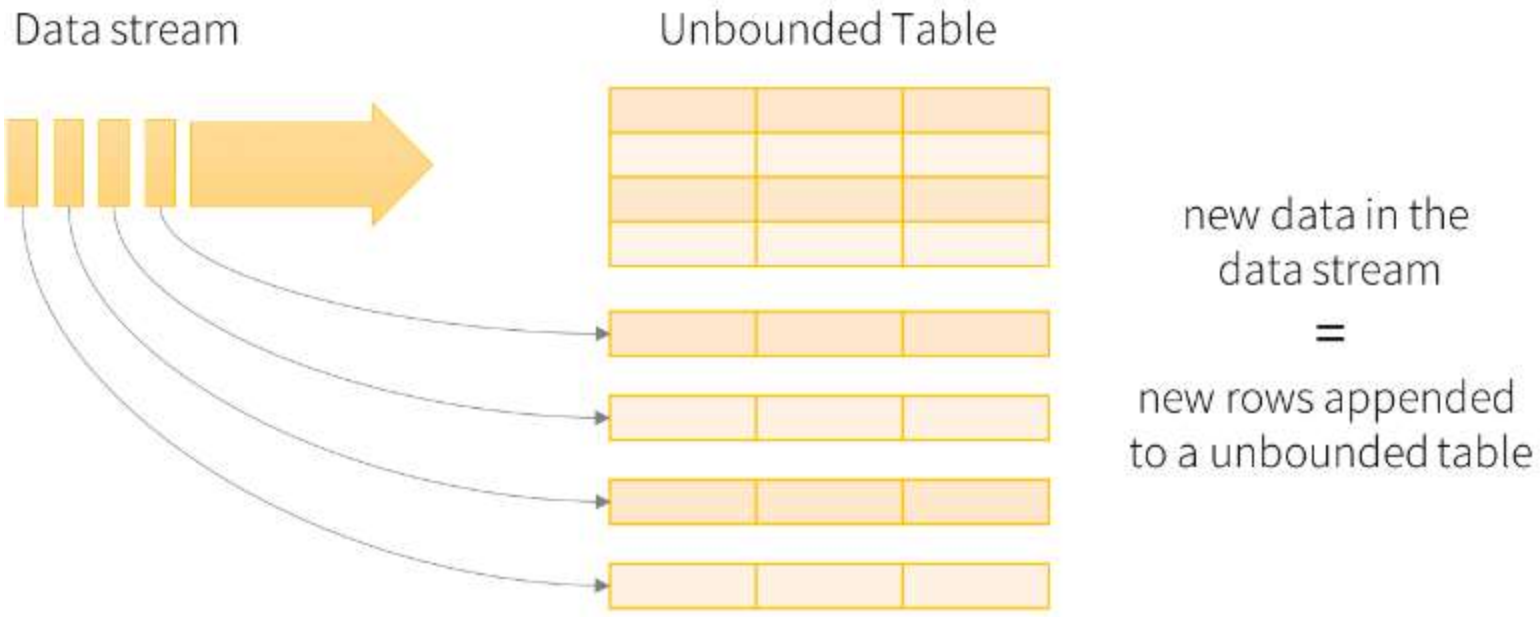
- means that for each message handed for processing, exactly one delivery is made to the recipient; the message can neither be lost nor duplicated.

Programming Model

- The key idea in Structured Streaming is to treat a live data stream as a table that is being continuously appended.
- This leads to a new stream processing model that is very similar to a batch processing model.
- We are expressing the streaming computation as standard batch-like query as on a static table.
- Spark runs it as an incremental query on the unbounded input table.
- Consider the input data stream as the “Input Table”. Every data item that is arriving on the stream is like a new row being appended to the Input Table.

Programming Model

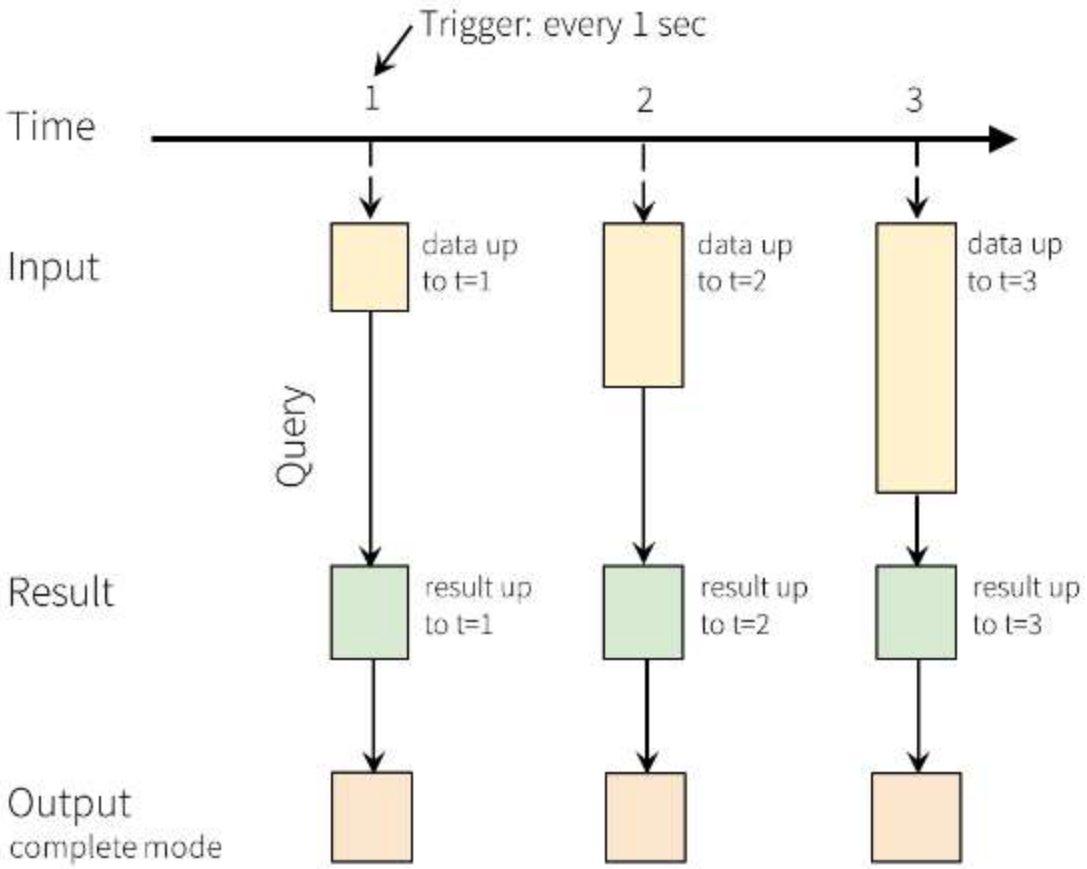
- Consider the input data stream as the “Input Table”. Every data item that is arriving on the stream is like a new row being appended to the Input Table.



Data stream as an unbounded table

Programming Model

- A query on the input will generate the "Result Table".
- Every trigger interval (say, every 1 second), new rows get appended to the Input Table, which eventually updates the Result Table.
- Whenever the result table gets updated, we would want to write the changed result rows to an external sink.

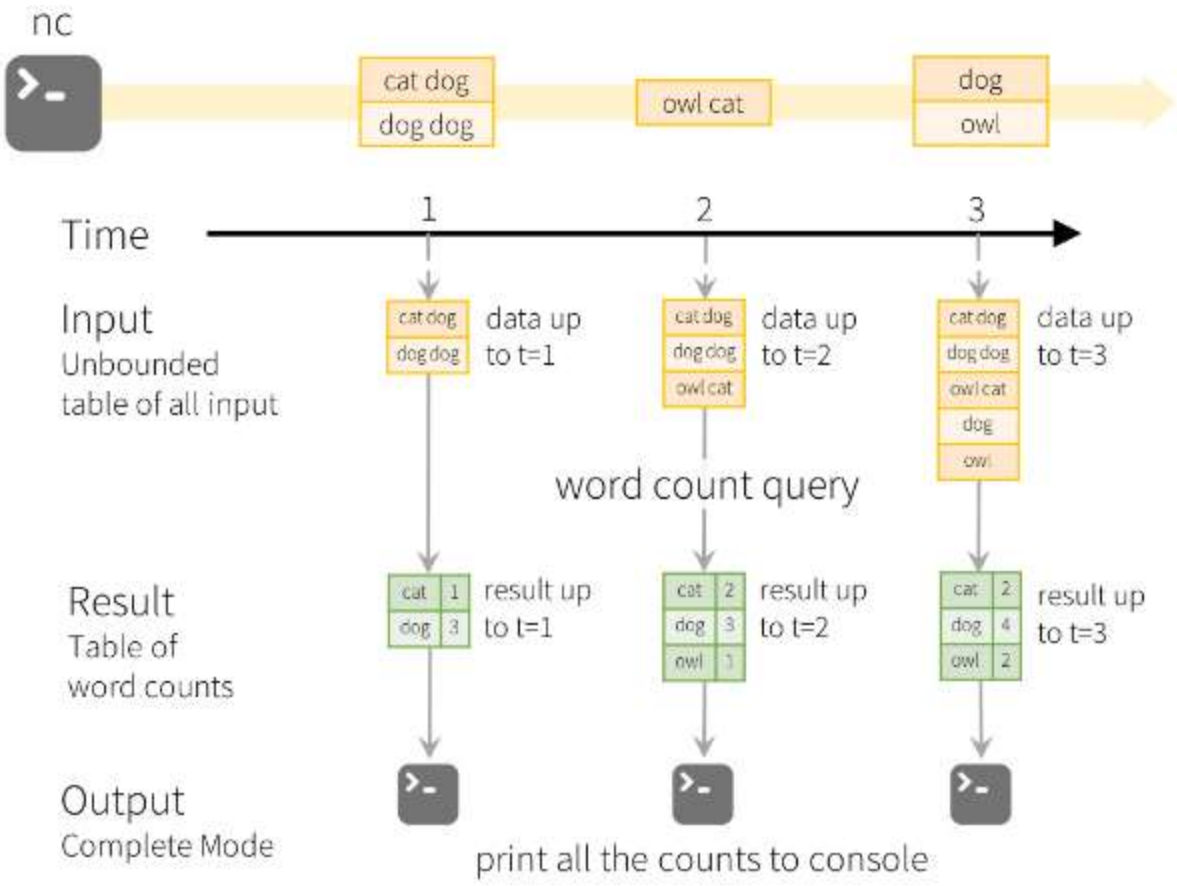


Programming Model

- The *Output* is defined as what gets written out to the external storage. There are a few types of output modes that can be specified in applications based on requirement:
- **Complete Mode -** The entire updated Result Table will be written to the external storage. It is up to the storage connector to decide how to handle writing of the entire table.
- **Append Mode -** This is the default mode. Only the new rows appended in the Result Table since the last trigger will be written to the external storage. This is applicable only on the queries where existing rows in the Result Table are not expected to change.
- **Update Mode -** Only the rows that were updated in the Result Table since the last trigger will be written to the external storage (available since Spark 2.1.1).
- Note that this is different from the Complete Mode in that this mode only outputs the rows that have changed since the last trigger. If the query doesn't contain aggregations, it will be equivalent to Append mode.

Programming Model

- The first *lines* DataFrame is the input table, and the final *wordCounts* DataFrame is the result table.
- Note that the query on streaming *lines* DataFrame to generate *wordCounts* is exactly the same as it would be a static DataFrame.
- However, when this query is started, Spark will continuously check for new data from the socket connection.
- If there is new data, Spark will run an “incremental” query that combines the previous running counts with the new data to compute updated counts.



Programming Model

- Note that Structured Streaming does not materialize the entire table.
- It reads the latest available data from the streaming data source, processes it incrementally to update the result, and then discards the source data.
- It only keeps around the minimal intermediate state data as required to update the result (e.g. intermediate counts in the earlier example).
- This model is significantly different from many other stream processing engines.
- Many streaming systems require the user to maintain running aggregations themselves, thus having to reason about fault-tolerance, and data consistency (at-least-once, or at-most-once, or exactly-once).
- In this model, Spark is responsible for updating the Result Table when there is new data, thus relieving the users from reasoning about it.