Spark Streaming
Exercise 2

Today you'll learn about Structured Streaming
(Sources: https://spark.apache.org/;
Spark: The Definitive Guide by Bill Chambers, Matei Zaharia)

Text and images are directly copied from these sources.

Spark includes two streaming APIs: the earlier DStream API which is purely micro-batch oriented and the newer Structured Streaming API which adds higher-level optimizations, event time, and support for continuous processing.

In this tutorial we are focusing on Structured Streaming API.

What is structured streaming?

Structured streaming API was released along Spark 2.2 production-ready release. By default, Structured Streaming queries are processed using a micro-batch processing engine, which processes data streams as a series of small batch jobs. However, since Spark 2.3, a new low-latency processing mode called Continuous Processing was introduced, which can achieve end-to-end latencies as low as 1 millisecond with at-least-once guarantees. Without changing the Dataset/DataFrame operations in your queries, it is possible to choose the mode based on application requirements.

Structured stream API perceives data streaming as continuously updating tables (new rows appended every moment), which makes stream processing very similar to batch processing. However, in spark streaming micro-batches are sequenced to enable the stream processing operations.

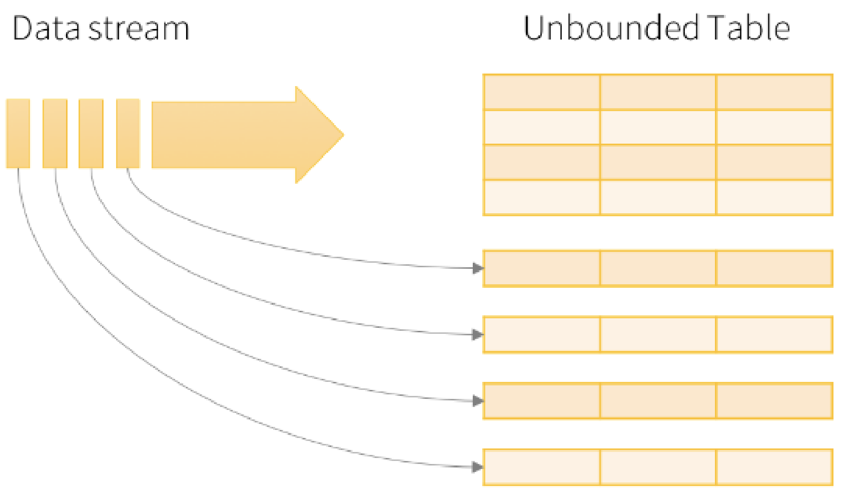
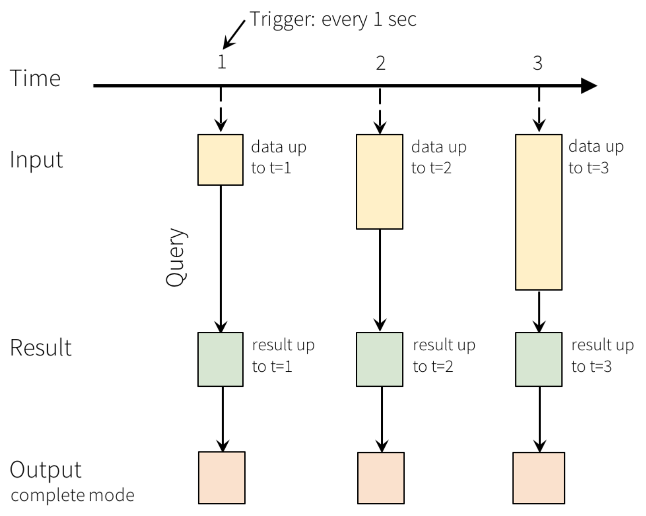


Figure source:
https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html

As can be seen from the figure, the data stream acts like a table and new rows are appended as soon as they arrive.



Programming Model for Structured Streaming

Figure source:
https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html

Basic terminology:

Input Source - allow to get streaming data into Structured Streaming. There are a few built-in sources: File source - Reads files written in a directory as a stream of data; Kafka source - Reads data from Kafka; Socket source (for testing) - Reads UTF8 text data from a socket connection; Rate source (for testing) - Generates data at the specified number of rows per second, each output row contains a timestamp and value.

Sink - specify destination for the result. A number of built-in sinks is available: File sink - Stores the output to a directory; Kafka sink - Stores the output to one or more topics in Kafka; Foreach sink - Runs arbitrary computation on the records in the output; Console sink (for debugging) - Prints the output to the console/stdout every time there is a trigger; Memory sink (for debugging) - The output is stored in memory as an in-memory table.

Output mode - defines how the result is written. Append mode (default) - only the new rows added to the Result Table since the last trigger will be outputted to the sink. Complete mode - The whole Result Table will be outputted to the sink after every trigger. Update mode - (Available since Spark 2.1.1) Only the rows in the Result Table that were updated since the last trigger will be outputted to the sink.

Triggers
The trigger settings of a streaming query defines the timing of streaming data processing, whether the query is going to be executed as micro-batch query with a fixed batch interval or as a continuous processing query. Triggers that are supported: unspecified (default) - the query will be executed in micro-batch mode, where micro-batches will be generated as soon as the previous micro-batch has completed processing. Fixed interval micro-batches - The query will be executed with micro-batches mode, where micro-batches will be kicked off at the user-specified intervals. One-time micro-batch - The query will execute *only one* micro-batch to process all the available data and then stop on its own. Continuous with fixed checkpoint interval (experimental) - The query will be executed in the new low-latency, continuous processing mode.

Event-Time Processing
Event-time is the time embedded in the data itself. For example, if you want to get the number of events generated by IoT devices every minute, then you probably want to use the time when the data was generated (that is, event-time in the data), rather than the time Spark receives them. This event-time is naturally expressed in this model – each event from the devices is a row in the table, and event-time is a column value in the row. This allows window-based aggregations (e.g. number of events every minute) to be just a special type of grouping and aggregation on the event-time column – each time window is a group and each row can belong to multiple windows/groups.

Since Spark is updating the Result Table, it has full control over updating old aggregates when there is late data, as well as cleaning up old aggregates to limit the size of intermediate state data. Since Spark 2.1, there is support for watermarking which allows the user to specify the threshold of late data, and allows the engine to accordingly clean up old state.

Let's walk through a simple example to see how structured streaming works in Spark.

1) Launching Pyspark Notebook

Please refer to Exercise 1

2) Building up Spark Session

We learned this in our last exercise. So, we will follow the same steps from Exercise 1

In our last exercise, we used example data (Ref Exercise 1) from the following link. For this exercise, the same data can be used or data of your choice (can be from API, web page etc.)
https://drive.google.com/open?id=1YLo060ccO-JNBJ7Cn_DT_y0LTkps-0Lg

3) Reading the data

In this example, we will read data similar to exercise 1. But you are free to use a different file or source.

```
df = spark.read.csv("PATH_TO_FILE", header=True, inferSchema=True)
df.show(vertical=True)
```

4) Getting schema

As we are working with time dependent data, it is always useful to know about the parameters and their type. This way, it is easier to perform data aggregation, processing etc.

```
df.createOrReplaceTempView("weatherConditions")
staticSchema = df.schema
```

Let's try to perform simple filtering

```
from pyspark.sql import functions as F
df.filter((F.col('air_temperature')<5))
```

Now that we have recalled some of the basics from past exercise. Let's move on to spark streaming

6) Structure streaming has two main operational capabilities readStream and writeStream

Let's start with reading the stream, readStream mainly requires a schema (we created it in initial steps). Secondly, we need to pass the value for how much files we want to read?.

```
streamingDF = spark.readStream\
.schema(staticSchema)\
.option("maxFilesPerTrigger", "Require numeric value i.e. if you have
more than 1 file/feed coming from source, how much you want to read ?
1,2,3?")\
.format("Require file format")\
.option("header", "true")\
.load("PATH_TO_FILE*.csv")
```

Q: Try to read a response from API or json file or try simple scraping

7) Is your data streaming?

In previous exercises, we used show() method to know if the data is read or not?

Q: Try to view the results using print() or show(). See what you get
Q: How can we know if the stream has loaded the data?

`streamingDF.isStreaming` // if its returns True , we are good to go

8) Applying basic filtering to the stream (similar to exercise 1)

```
filteredDF = streamingDF.filter((F.col('air_temperature')<5))
```

9) Writing Stream to a sink

```
df = filteredDF.writeStream.format("memory") \
.queryName("weatherQuery") \
.outputMode("append") \
.start()
```

```
df.awaitTermination()
```

Q. Which sink is used here?
Q. Which output mode is used here?
Q. Which trigger is used here?

Try to modify these to see the difference .

10) We can query the data while stream is being written and processed

```
spark.sql("""
SELECT *
FROM weatherQuery
""")\
.show(5)
```

Q: Is there any difference between methods and operations provided for structured streaming using sparksession and spark streamingcontext?

Questions for the Exercise 2 report:

Please answer the following questions. Compile your answers into a 1 page report, put there your student ID number and your name.

1. What have you learned today?
2. Is there any difference between methods and operations provided for structured streaming using sparksession and spark streamingcontext?