Reading and Writing Streams with Spark Structured Streaming

Justin Post

Recap

We'll use Spark Structured Streaming to handle our streaming data (Guide)

- Create a spark session

1. **Read in a stream**
    - Stream from a file, terminal, or use something like kafka
2. Set up transformations/aggregations to do (mostly using SQL type functions)
    - Perhaps over windows
3. Set up **writing of the query** to an output source
    - Console (for debugging)
    - File (say .csv)
    - Database
4. `query.start()` the query!
    - Continues listening until terminated (`query.stop()`)

NC STATE UNIVERSITY

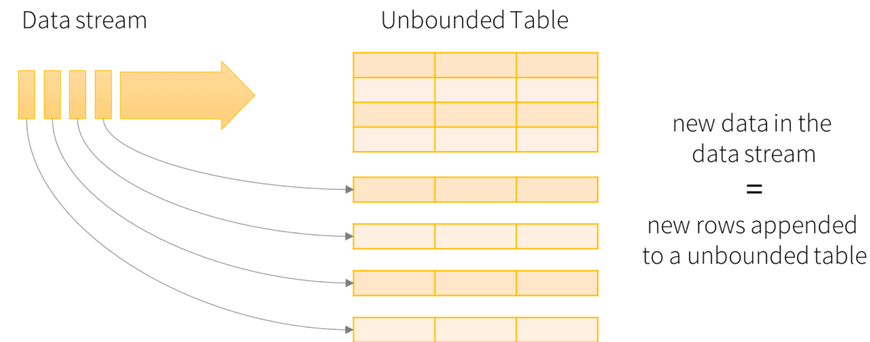
Streaming DataFrames

Stream is read into a Spark SQL data frame

- Data frames can be used to represent both static data and streaming data

Differences:

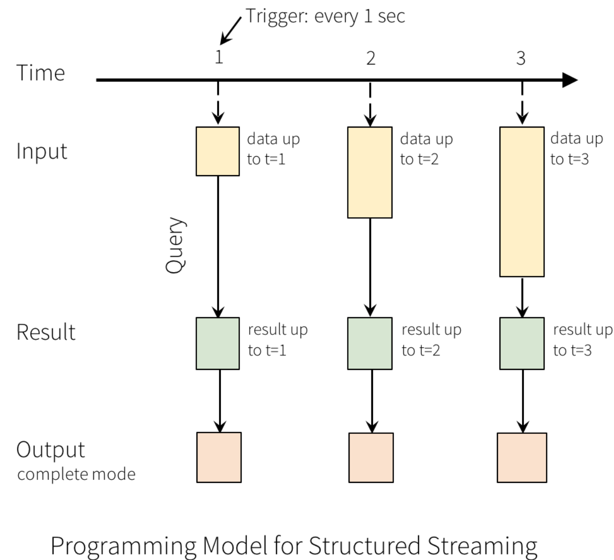
- Streaming data frames are unbounded and schema is only checked at runtime
- Rows added incrementally



Data stream as an unbounded table

NC STATE UNIVERSITY

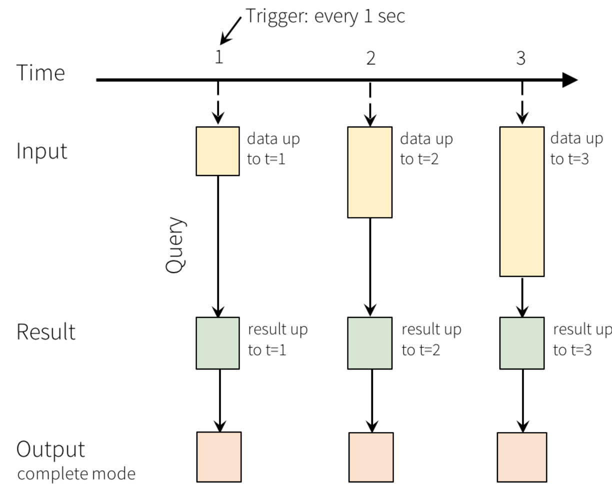
Streaming DataFrames



Programming Model for Structured Streaming

- When the query starts, Spark will check for new data (at a specified interval of time)
- 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

Streaming DataFrames



Programming Model for Structured Streaming

> 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 the minimal intermediate state data as required to update the result (e.g. intermediate counts).

NC STATE UNIVERSITY

Reading a Stream

Stream read in using the `DataStreamReader` interface (`SparkSession.readStream`)

- `readStream` has different methods to customize/set-up how to read the stream

Reading a Stream

Stream read in using the `DataStreamReader` interface (`SparkSession.readStream`)

- `readStream` has different methods to customize/set-up how to read the stream
  - `.format()` - (generic) specifies the input source
  - `.schema()` - setup what Spark should expect
  - `.option(key, value)` - allows an input option on a file source
  - `.load()` - loads a data stream and returns a DataFrame

Reading Data from a Kafka Stream

- Common syntax for reading in data

```
df = spark \
    .readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:9092") \
    .option("subscribe", "topic_name") \
    .load()
```

Reading in Testing Data

- `rate` format generates timestamp data at a specified interval of time

```
df = spark \
    .readStream \
    .format("rate") \
    .option("rowsPerSecond", 1) \
    .load()
```

Reading Data From a CSV

- Common syntax for reading in data

```
myschema = StructType().add("value", "string")
df = spark \
        .readStream \
        .schema(myschema) \
        .csv("csv_files") #automatically 'loads'
```

NC STATE UNIVERSITY

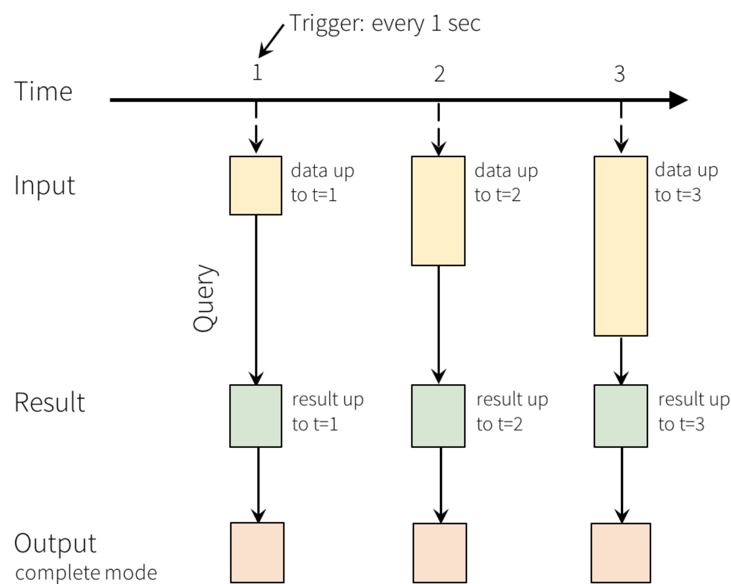
Quick Example

Let's jump into `pyspark` and use the "rate" format

- Will need to write the stream to see it (covered in more detail shortly)

Starting Streaming Queries

Notice that the process doesn't evaluate things until we use `.start()`



Programming Model for Structured Streaming

NC STATE UNIVERSITY

Starting Streaming Queries

Uses the `DataStreamWriter` interface (`df_with_transforms_etc.writeStream`)

- `writeStream` has different methods to customize output type and location

Starting Streaming Queries

Uses the `DataStreamWriter` interface (`df_with_transforms_etc.writeStream`)

- `writeStream` has different methods to customize output type and location
- Output type:
  - Complete - print entire table at each update
    - Only supported for aggregation queries
  - Append (default) - only new rows added to the Results table are outputted
    - Only applicable if rows added can never change (say from late data)
  - Update - similar to append but allows flexibility if data may change

Starting Streaming Queries

Uses the `DataStreamWriter` interface (`df_with_transforms_etc.writeStream`)

- `writeStream` has different methods to customize output type and location
- Output type:
  - Complete - print entire table at each update
    - Only supported for aggregation queries
  - Append (default) - only new rows added to the Results table are outputted
    - Only applicable if rows added can never change (say from late data)
  - Update - similar to append but allows flexibility if data may change
- Nice table in the guide to help out!

NC STATE UNIVERSITY

Starting Streaming Queries

Uses the `DataStreamWriter` interface (`df_with_transforms_etc.writeStream`)

- `writeStream` has different methods to customize output type and location
- Output sinks (location):
  - console sink for debugging
    - `query.writeStream.outputMode("append").format("console")`

Starting Streaming Queries

Uses the `DataStreamWriter` interface (`df_with_transforms_etc.writeStream`)

- `writeStream` has different methods to customize output type and location
- Output sinks (location):
  - console sink for debugging
    - `query.writeStream.outputMode("append").format("console")`
  - memory - stores output in an in-memory table that you can investigate
    - `query.writeStream.format("memory").queryName("tableName")`
  - File sink (csv, json, parquet, etc)
    - `query.writeStream.outputMode("append").format("csv").option("path", "path_to_file")`

Starting Streaming Queries

Uses the `DataStreamWriter` interface (`df_with_transforms_etc.writeStream`)

- `writeStream` has different methods to customize output type and location
- Output sinks (location):
  - console sink for debugging
    - `query.writeStream.outputMode("append").format("console")`
  - memory - stores output in an in-memory table that you can investigate
    - `query.writeStream.format("memory").queryName("tableName")`
  - File sink (csv, json, parquet, etc)
    - `query.writeStream.outputMode("append").format("csv").option("path", "path_to_file")`
  - Kafka sink
- Nice table in the guide to help out!

Starting Streaming Queries

- Updates based on **trigger** settings
  - Default uses micro-batches which are generated as soon as the previous micro-batch has completed processing

Starting Streaming Queries

- Updates based on **trigger** settings
  - Default uses micro-batches which are generated as soon as the previous micro-batch has completed processing
  - Fixed interval micro-batches (see guide for more info)
    - `writeStream....trigger(processingTime = "2 seconds")....`
  - One-time micro-batch - executes once and shuts itself down (essentially a quick update since you last ran the query)
    - `writeStream....trigger(once = True)....`
  - Continuous - experimental

Multiple Queries and Stopping Queries

- Can do multiple queries at once and they share resources
  - `spark.streams.active` gives a list of all active streaming queries
- Stop the query with `query.stop()` (where `query` is the name of the query)
- Spark has a GUI to help monitor! Doesn't work easily within our jupyterhub though
  - http://localhost:4040/

Quick Example

Let's write to a table in memory!

Recap

- Read in streams with `readStream`
- Write queries with `writeStream`
- Must `.start()` the query
- Can run multiple queries at once