PySpark Streaming

Processing real-time data

Stream processing

- Stream processing is the act of continuously incorporating new data to compute a result
- In stream processing, the input data is unbounded and has no predetermined beginning or end
- It simply forms a series of events that arrive at the stream processing system
  - credit card transactions
  - clicks on a website
  - sensor readings from Internet of Things [IoT] devices
- In contrast to batch processing, in which computation runs on a fixed-input dataset
- Spark’s streamingAPI supports common stream processing

Stream processing

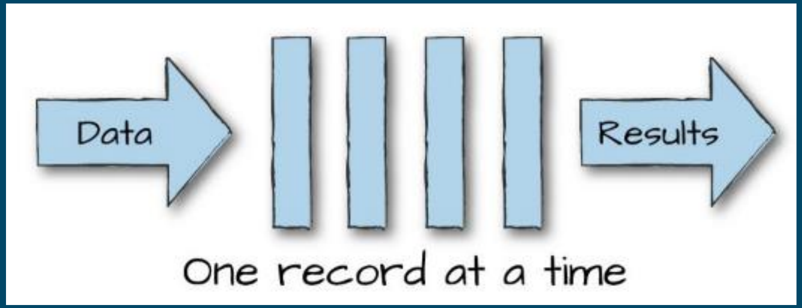
- Advantages
  - lower latency
    - when the application needs to respond quickly (on a timescale of minutes, seconds, or milliseconds)
  - Can be more efficient to update a result rather than repeated batch jobs, because is computation is incremental
- Challenges
  - Processing out-of-order data based on application timestamps (also called event time)
  - Maintaining large amounts of state (event history, e.g., for event ordering)
  - Supporting high-data throughput
  - Processing each event exactly once despite machine failures
  - Responding to events at low latency
  - Determining how to update output sinks as new events arrive
  - Writing data transactionally to output systems

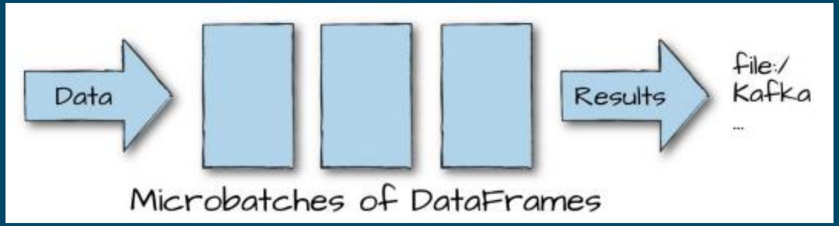
Event Time and Processing Time

- Event Time
  - Processing data based on timestamps inserted into each record at the source
  - Note that, records may arrive to the system out of order
- Processing Time
  - Processing data based the time when the record is received at the streaming application

Event batch size

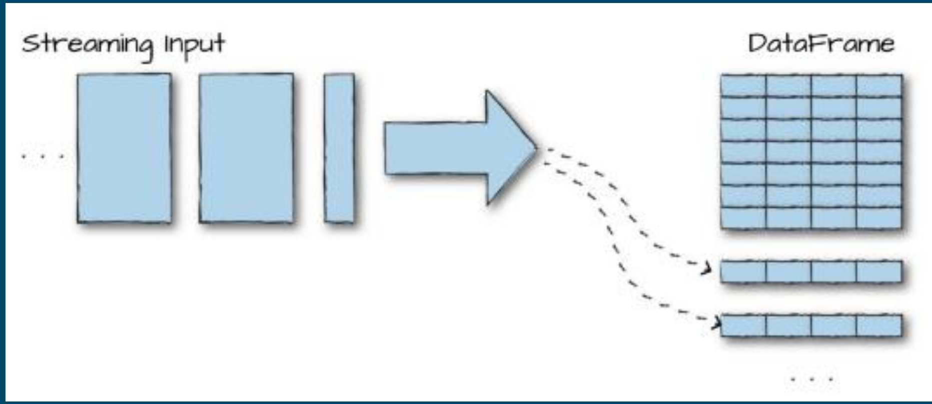
- Continuous processing-based systems
  - After an event is received, the event is passed onto other systems
  - Lowest latency
- Event batch (size > 1)
  - Accumulate events into a batch
    - Micro-batch (small size, in terms of # of event, or time passed)
    - Higher throughput, but lower latency





Structured Streaming

- Structured API for Spark
  - DataFrame based functions useful for stream processing
  - Reads common event sources
    - File, Socket, Kafka
  - Write to common event sinks
    - Output mode for append, update, complete
  - Triggers for when data is processed



A Streaming example

Time-stamped open/close events

- The events are in some files
  - We'll read the files to simulate the stream of events

```
%fs ls /databricks-datasets/structured-streaming/events/
```

path	name	size
dbfs:/databricks-datasets/structured-streaming/events/file-0.json	file-0.json	72530
dbfs:/databricks-datasets/structured-streaming/events/file-1.json	file-1.json	72961
dbfs:/databricks-datasets/structured-streaming/events/file-10.json	file-10.json	73025
dbfs:/databricks-datasets/structured-streaming/events/file-11.json	file-11.json	72999
dbfs:/databricks-datasets/structured-streaming/events/file-12.json	file-12.json	72987
dbfs:/databricks-datasets/structured-streaming/events/file-13.json	file-13.json	73006
dbfs:/databricks-datasets/structured-streaming/events/file-14.json	file-14.json	73003
dbfs:/databricks-datasets/structured-streaming/events/file-15.json	file-15.json	73007

time	
2016-07-28T04:19:28.000+0000	
2016-07-28T04:19:28.000+0000	Close
2016-07-28T04:19:29.000+0000	Open
2016-07-28T04:19:31.000+0000	Close
2016-07-28T04:19:31.000+0000	Open
2016-07-28T04:19:31.000+0000	Open
2016-07-28T04:19:32.000+0000	Close

Static analysis of the data

- Read the data into a DataFrame

```
inputPath = "/databricks-datasets/structured-streaming/events/"

# Since we know the data format already, let's define the schema to speed up processing (no need for Spark to infer schema)
jsonSchema = StructType([ StructField("time", TimestampType(), True), StructField("action", StringType(), True) ])

# Static DataFrame representing data in the JSON files
staticInputDF = (
  spark
    .read
    .schema(jsonSchema)
    .json(inputPath)
)
```

time	action
2016-07-28T04:19:28.000+0000	Close
2016-07-28T04:19:28.000+0000	Close
2016-07-28T04:19:29.000+0000	Open

Group data by action and 1 hour window

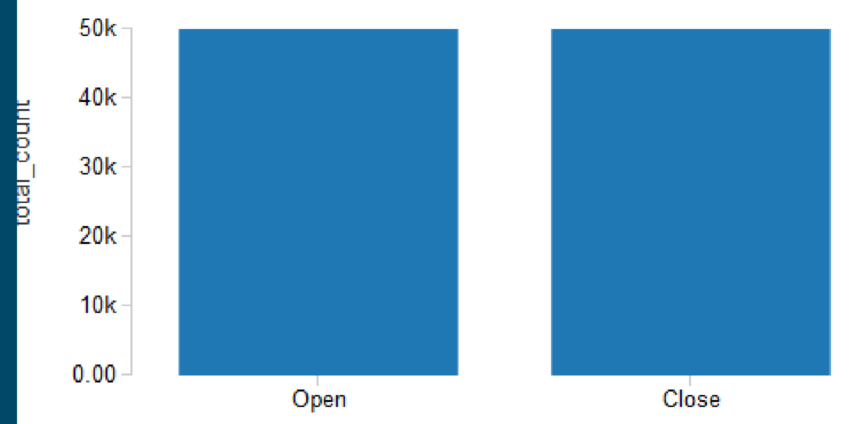
- Window function
  - Column, Time spec
  - Aggregates data

```python
from pyspark.sql.functions import *      # for window() function

staticCountsDF = (
  staticInputDF
    .groupBy(
       staticInputDF.action,
       window(staticInputDF.time, "1 hour"))
    .count()
)
```

```sql
%sql select action, sum(count) as total_count from static_counts group by action
```

(5) Spark Jobs



From static now to stream

Streaming read

- readStream instead of read

```
# Similar to definition of staticInputDF
streamingInputDF = (
  spark
    .readStream
    .schema(jsonSchema)               # S
    .option("maxFilesPerTrigger", 1)  # T
    .json(inputPath)
)
```

- Same query as for static query

```
# Same query as staticInputDF
streamingCountsDF = (
  streamingInputDF
    .groupBy(
      streamingInputDF.action,
      window(streamingInputDF.time, "1 hour"))
    .count()
)
```

- Run the process
  - continuous

```
query = (
  streamingCountsDF
    .writeStream
    .format("memory")
    .queryName("counts")
    .outputMode("complete")
    .start()
)
```

PySpark streaming API has many functions

this was a simple example

Stream processing

- Stream processing is the act of continuously incorporating new data to compute a result
- In stream processing, the input data is unbounded and has no predetermined beginning or end
- It simply forms a series of events that arrive at the stream processing system
  - credit card transactions
  - clicks on a website
  - sensor readings from Internet of Things [IoT] devices
- In contrast to batch processing, in which computation runs on a fixed-input dataset
- Spark’s streaming API supports common stream processing

Important to remember