ĐẠI HỌC
BÁCH KHOA

25 YEARS ANNIVERSARY
SOICT

HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

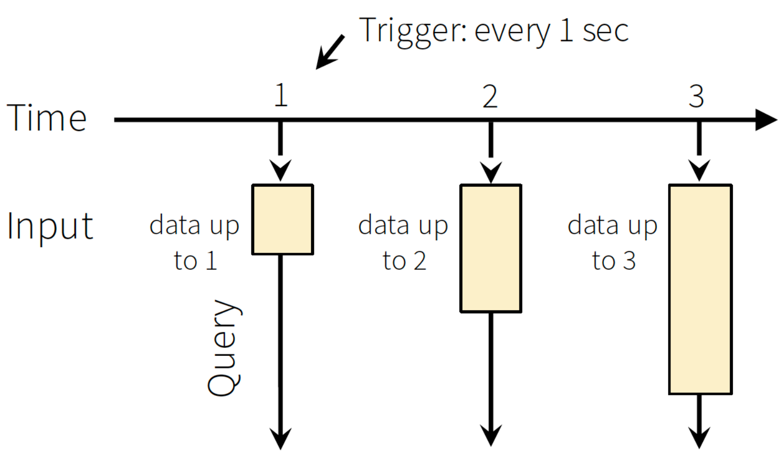
Chapter 7
Stream processing
Structured streaming in Spark

Pain points with DStreams

- Processing with event-time, dealing with late data
  - DStream API exposes batch time, hard to incorporate event-time
- Interoperate streaming with batch AND interactive
  - RDD/DStream has similar API, but still requires translation
- Reasoning about end-to-end guarantees
  - Requires carefully constructing sinks that handle failures correctly
  - Data consistency in the storage while being updated

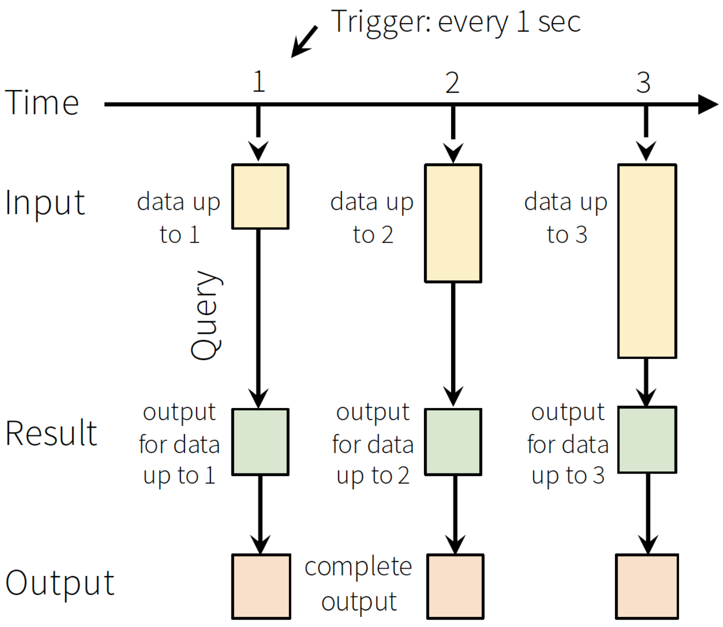
New model

- Input: data from source as an append-only table
- Trigger: how frequently to check input for new data
- Query: operations on input usual map/filter/reduce new window, session ops



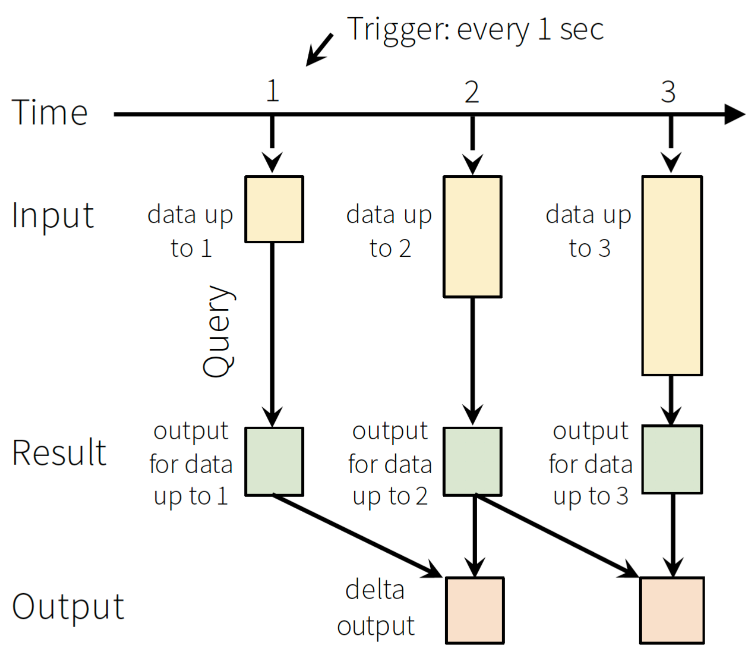
New model (2)

- Result: final operated table updated every trigger interval
- Output: what part of result to write to data sink after every trigger
- Complete output: Write full result table every time



New model (3)

- Delta output: Write only the rows that changed in result from previous batch
- Append output: Write only new rows
- *Not all output modes are feasible with all queries



Batch ETL with DataFrames

```
input = spark.read
        .format("json")
        .load("source-path")
result = input
        .select("device", "signal")
        .where("signal > 15")
result.write
        .format("parquet")
        .save("dest-path")
```

- Read from Json file
- Select some devices
- Write to parquet file

Streaming ETL with DataFrames

```
input = spark.read
        .format("json")
        .stream("source-path")
result = input
        .select("device", "signal")
        .where("signal > 15")
result.write
        .format("parquet")
        .startStream("dest-path")
```

- Read from Json file stream
  - Replace load() with stream()
- Select some devices
  - Code does not change
- Write to Parquet file stream
  - Replace save() with startStream()

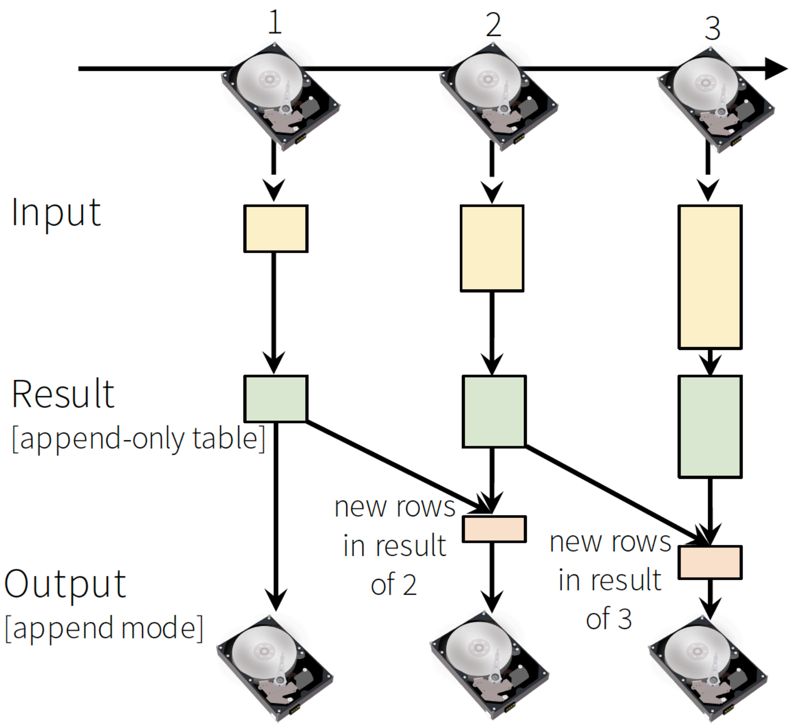
Streaming ETL with DataFrames

```
input = spark.read
        .format("json")
        .stream("source-path")
result = input
        .select("device", "signal")
        .where("signal > 15")
result.write
        .format("parquet")
        .startStream("dest-path")
```

- read…stream() creates a streaming DataFrame, does not start any of the computation
- write…startStream() defines where & how to output the data and starts the processing

Streaming ETL with DataFrames

```
input = spark.read
        .format("json")
        .stream("source-path")
result = input
        .select("device", "signal")
        .where("signal > 15")
result.write
        .format("parquet")
        .startStream("dest-path")
```



Continuous Aggregations

```
input.avg("signal")
```

- Continuously compute average signal across all devices

```
input.groupBy("device-type")
          .avg("signal")
```

- Continuously compute average signal of each type of device

Continuous Windowed Aggregations

```
input.groupBy(
        $"device-type",
        window($"event-time-col", "10 min"))
        .avg("signal")
```

- Continuously compute average signal of each type of device in last 10 minutes using event-time
- Simplifies event-time stream processing (not possible in DStreams) Works on both, streaming and batch jobs

Joining streams with static data

```
kafkaDataset = spark.read
        .kafka("iot-updates")
        .stream()

staticDataset = ctxt.read
        .jdbc("jdbc://", "iot-
device-info")

joinedDataset =
        kafkaDataset.join(
        staticDataset,
"device-type")
```

- Join streaming data from Kafka with static data via JDBC to enrich the streaming data …
- … without having to think that you are joining streaming data

Output modes

Defines what is outputted every time there is a trigger
Different output modes make sense for different queries

- Append mode with non-aggregation queries

```
input.select("device", "signal")
        .write
        .outputMode("append")
        .format("parquet")
        .startStream("dest-path")
```

- Complete mode with aggregation queries

```
input.agg(count("*"))
        .write
        .outputMode("complete"
)
        .format("parquet")
        .startStream("dest-path")
```

Query Management

```
query = result.write
              .format("parquet")
              .outputMode("append")
              .startStream("dest-path")

query.stop()
query.awaitTermination()
query.exception()

query.sourceStatuses()
query.sinkStatus()
```

- query: a handle to the running streaming computation for managing it
  - Stop it, wait for it to terminate
  - Get status
  - Get error, if terminated
- Multiple queries can be active at the same time
- Each query has unique name for keeping track

Query execution

- Logically
  - Dataset operations on table (i.e. as easy to understand as batch)
- Physically
  - Spark automatically runs the query in streaming fashion (i.e. incrementally and continuously)