Near-real time processing using Spark

RDDs, DataFrame/DataSet, Spark Streaming, and Structured Streaming

Vladimir Sivčević

Senior Data Engineer

www.vladsiv.com

May 9, 2024

Table of Contents

Table of Contents

Table of Contents

Table of Contents

RDD

RDD - Resilient Distributed Dataset

RDD

RDD - Resilient Distributed Dataset

- **Resilient**

RDD

RDD - Resilient Distributed Dataset

- **Resilient**
  - **Immutable** - Rebuild lost data

RDD

RDD - Resilient Distributed Dataset

- **Resilient**
  - **Immutable** - Rebuild lost data
  - **Fault Tolerant** - Recover data using Lineage

RDD

RDD - Resilient Distributed Dataset

- **Resilient**
  - **Immutable** - Rebuild lost data
  - **Fault Tolerant** - Recover data using Lineage
- **Distributed** - Across nodes, operated on in parallel

RDD

RDD - Resilient Distributed Dataset

- **Resilient**
  - **Immutable** - Rebuild lost data
  - **Fault Tolerant** - Recover data using Lineage
- **Distributed** - Across nodes, operated on in parallel
- Two ways to create them

RDD

RDD - Resilient Distributed Dataset

- **Resilient**
  - **Immutable** - Rebuild lost data
  - **Fault Tolerant** - Recover data using Lineage
- **Distributed** - Across nodes, operated on in parallel
- Two ways to create them
  - *Parallelized Collections*

```
data = [1, 2, 3, 4, 5]
distData = sc.parallelize(data)
```

RDD

RDD - Resilient Distributed Dataset

- **Resilient**
  - **Immutable** - Rebuild lost data
  - **Fault Tolerant** - Recover data using Lineage
- **Distributed** - Across nodes, operated on in parallel
- Two ways to create them
  - *Parallelized Collections*

    ```
    data = [1, 2, 3, 4, 5]
    distData = sc.parallelize(data)
    ```
  - *External Datasets* - Any Hadoop supported storage - HDFS, S3, HBase...

    ```
    distFile = sc.textFile("data.txt")
    ```

RDD

RDD - Resilient Distributed Dataset

- **Resilient**
  - **Immutable** - Rebuild lost data
  - **Fault Tolerant** - Recover data using Lineage
- **Distributed** - Across nodes, operated on in parallel
- Two ways to create them
  - *Parallelized Collections*

    ```
    data = [1, 2, 3, 4, 5]
    distData = sc.parallelize(data)
    ```
  - *External Datasets* - Any Hadoop supported storage - HDFS, S3, HBase...

    ```
    distFile = sc.textFile("data.txt")
    ```
- **Partitions** - How is data distributed, determined by number of cores

  ```
  sc.parallelize(data, 10)
  ```

RDD

RDD - Resilient Distributed Dataset

- **Resilient**
  - **Immutable** - Rebuild lost data
  - **Fault Tolerant** - Recover data using Lineage
- **Distributed** - Across nodes, operated on in parallel
- Two ways to create them
  - *Parallelized Collections*

    ```
    data = [1, 2, 3, 4, 5]
    distData = sc.parallelize(data)
    ```
  - *External Datasets* - Any Hadoop supported storage - HDFS, S3, HBase...

    ```
    distFile = sc.textFile("data.txt")
    ```
- **Partitions** - How is data distributed, determined by number of cores

  ```
  sc.parallelize(data, 10)
  ```
- Generally unstructured data

RDD

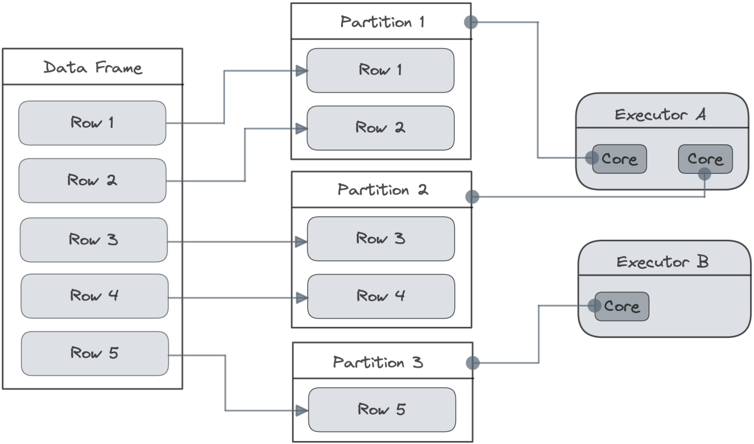

Image 1 - Partitions

RDD

RDDs support two type of operations:

- **Transformations** - create a new dataset from an existing one i.e. creates a completely new RDD

RDD

RDDs support two type of operations:

- **Transformations** - create a new dataset from an existing one i.e. creates a completely new RDD
  - `map()` - applies function to each element of a dataset and returns new RDD

RDD

RDDs support two type of operations:

- **Transformations** - create a new dataset from an existing one i.e. creates a completely new RDD
  - `map()` - applies function to each element of a dataset and returns new RDD
  - All transformations are lazy - think DAG
- **Actions** - return a value to the driver program after running a computation on the dataset

RDD

RDDs support two type of operations:

- **Transformations** - create a new dataset from an existing one i.e. creates a completely new RDD
  - `map()` - applies function to each element of a dataset and returns new RDD
  - All transformations are lazy - think DAG
- **Actions** - return a value to the driver program after running a computation on the dataset
  - `reduce()` - aggregates all the elements using some function

RDD

RDDs support two type of operations:

- **Transformations** - create a new dataset from an existing one i.e. creates a completely new RDD
  - `map()` - applies function to each element of a dataset and returns new RDD
  - All transformations are lazy - think DAG
- **Actions** - return a value to the driver program after running a computation on the dataset
  - `reduce()` - aggregates all the elements using some function
  - Transformations are invoked when an action is triggered

RDD

RDDs support two type of operations:

- **Transformations** - create a new dataset from an existing one i.e. creates a completely new RDD
  - `map()` - applies function to each element of a dataset and returns new RDD
  - All transformations are lazy - think DAG
- **Actions** - return a value to the driver program after running a computation on the dataset
  - `reduce()` - aggregates all the elements using some function
  - Transformations are invoked when an action is triggered

Note

Directly programming on RDD level is not efficient, introduces latency, lacks control, and is generally discouraged. Use it if you really have to!

RDD

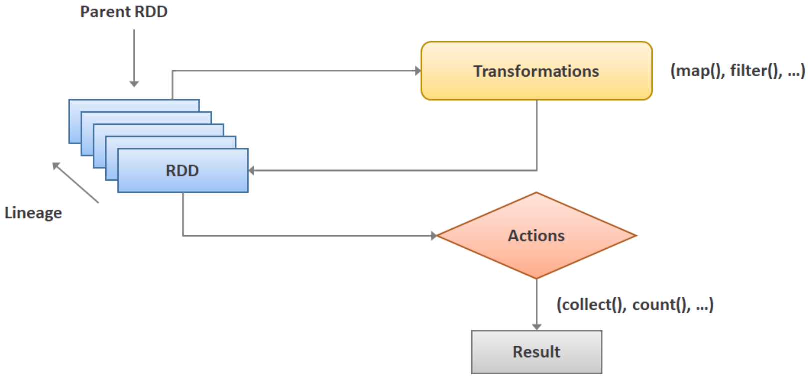

Image 2 - RDD Operations

Spark Streaming

Image 3 - Spark Streaming

Spark Streaming

Image 3 - Spark Streaming

- Mini batching - near-real time

Spark Streaming

Image 3 - Spark Streaming

- Mini batching - near-real time
- Legacy - there are better options now

Spark Streaming

Image 3 - Spark Streaming

- Mini batching - near-real time
- Legacy - there are better options now
- Abstract concept - **DStream** - Discretized Stream

Spark Streaming

Image 3 - Spark Streaming

- Mini batching - near-real time
- Legacy - there are better options now
- Abstract concept - **DStream** - Discretized Stream
  - Continuous stream of data

Spark Streaming

Image 3 - Spark Streaming

- Mini batching - near-real time
- Legacy - there are better options now
- Abstract concept - **DStream** - Discretized Stream
    - Continuous stream of data
    - Represents a continuous sequence of RDDs

Spark Streaming

Image 3 - Spark Streaming

- Mini batching - near-real time
- Legacy - there are better options now
- Abstract concept - **DStream** - Discretized Stream
    - Continuous stream of data
    - Represents a continuous sequence of RDDs
    - Operations on DStream translates to operations on underlying RDDs

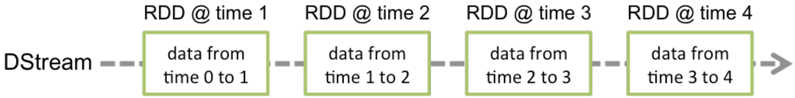

Image 4 - DStream RDDs

Spark Streaming

Image 3 - Spark Streaming

- Mini batching - near-real time
- Legacy - there are better options now
- Abstract concept - **DStream** - Discretized Stream
  - Continuous stream of data
  - Represents a continuous sequence of RDDs
  - Operations on DStream translates to operations on underlying RDDs

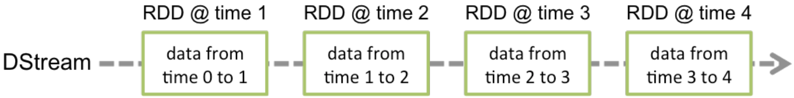

Image 4 - DStream RDDs

Spark Streaming

- Input DStreams - representing the stream of input data

Spark Streaming

- Input DStreams - representing the stream of input data
- Associted with **Receiver** - object which receives the data from a source, runs on an allocated core

Spark Streaming

- Input DStreams - representing the stream of input data
- Associted with **Receiver** - object which receives the data from a source, runs on an allocated core
    - Reliable Receiver - sends ACK to the source

Spark Streaming

- Input DStreams - representing the stream of input data
- Associted with **Receiver** - object which receives the data from a source, runs on an allocated core
    - Reliable Receiver - sends ACK to the source
    - Unrealiable Receiver - doesn't send ACK

Spark Streaming

- Input DStreams - representing the stream of input data
- Associted with **Receiver** - object which receives the data from a source, runs on an allocated core
  - Reliable Receiver - sends ACK to the source
  - Unrealiable Receiver - doesn't send ACK
  - We can create custom receivers

Spark Streaming

- Input DStreams - representing the stream of input data
- Associted with **Receiver** - object which receives the data from a source, runs on an allocated core
  - Reliable Receiver - sends ACK to the source
  - Unrealiable Receiver - doesn't send ACK
  - We can create custom receivers
- Sources

Spark Streaming

- Input DStreams - representing the stream of input data
- Associted with **Receiver** - object which receives the data from a source, runs on an allocated core
  - Reliable Receiver - sends ACK to the source
  - Unrealiable Receiver - doesn't send ACK
  - We can create custom receivers
- Sources
  - **Basic** - Available in `StreamingContext`: file systems and socket connections

Spark Streaming

- Input DStreams - representing the stream of input data
- Associted with **Receiver** - object which receives the data from a source, runs on an allocated core
  - Reliable Receiver - sends ACK to the source
  - Unrealiable Receiver - doesn't send ACK
  - We can create custom receivers
- Sources
  - **Basic** - Available in `StreamingContext`: file systems and socket connections
  - **Advanced** - Extra utility classes: Kafka, Flume, Kinesis

Spark Streaming

- Input DStreams - representing the stream of input data
- Associted with **Receiver** - object which receives the data from a source, runs on an allocated core
  - Reliable Receiver - sends ACK to the source
  - Unrealiable Receiver - doesn't send ACK
  - We can create custom receivers
- Sources
  - **Basic** - Available in `StreamingContext`: file systems and socket connections
  - **Advanced** - Extra utility classes: Kafka, Flume, Kinesis
- Drawbacks

Spark Streaming

- Input DStreams - representing the stream of input data
- Associted with **Receiver** - object which receives the data from a source, runs on an allocated core
  - Reliable Receiver - sends ACK to the source
  - Unrealiable Receiver - doesn't send ACK
  - We can create custom receivers
- Sources
  - **Basic** - Available in `StreamingContext`: file systems and socket connections
  - **Advanced** - Extra utility classes: Kafka, Flume, Kinesis
- Drawbacks
  - Directly dealing with RDDs

Spark Streaming

- Input DStreams - representing the stream of input data
- Associted with **Receiver** - object which receives the data from a source, runs on an allocated core
  - Reliable Receiver - sends ACK to the source
  - Unrealiable Receiver - doesn't send ACK
  - We can create custom receivers
- Sources
  - **Basic** - Available in `StreamingContext`: file systems and socket connections
  - **Advanced** - Extra utility classes: Kafka, Flume, Kinesis
- Drawbacks
  - Directly dealing with RDDs
  - Complicated and different API (from batch)

Spark Streaming

- Input DStreams - representing the stream of input data
- Associted with **Receiver** - object which receives the data from a source, runs on an allocated core
  - Reliable Receiver - sends ACK to the source
  - Unrealiable Receiver - doesn't send ACK
  - We can create custom receivers
- Sources
  - **Basic** - Available in `StreamingContext`: file systems and socket connections
  - **Advanced** - Extra utility classes: Kafka, Flume, Kinesis
- Drawbacks
  - Directly dealing with RDDs
  - Complicated and different API (from batch)
  - Latency

Spark Streaming

- Input DStreams - representing the stream of input data
- Associted with **Receiver** - object which receives the data from a source, runs on an allocated core
    - Reliable Receiver - sends ACK to the source
    - Unrealiable Receiver - doesn't send ACK
    - We can create custom receivers
- Sources
    - **Basic** - Available in `StreamingContext`: file systems and socket connections
    - **Advanced** - Extra utility classes: Kafka, Flume, Kinesis
- Drawbacks
    - Directly dealing with RDDs
    - Complicated and different API (from batch)
    - Latency
    - Bad Optimization

Spark Streaming

- Input DStreams - representing the stream of input data
- Associted with **Receiver** - object which receives the data from a source, runs on an allocated core
  - Reliable Receiver - sends ACK to the source
  - Unrealiable Receiver - doesn't send ACK
  - We can create custom receivers
- Sources
  - **Basic** - Available in `StreamingContext`: file systems and socket connections
  - **Advanced** - Extra utility classes: Kafka, Flume, Kinesis
- Drawbacks
  - Directly dealing with RDDs
  - Complicated and different API (from batch)
  - Latency
  - Bad Optimization
  - SQL not supported

Spark Streaming

- DStream can be combined and joined

Spark Streaming

- DStream can be combined and joined
- For example, Window operations

Spark Streaming

- DStream can be combined and joined
- For example, Window operations

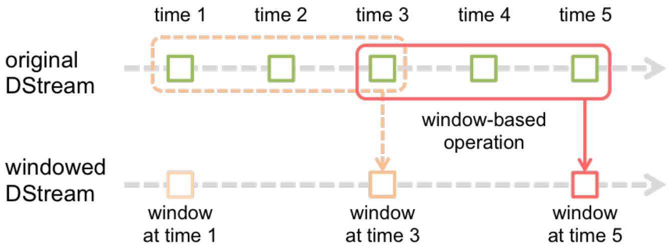

Image 5 - DStream Window

Spark Streaming

- DStream can be combined and joined
- For example, Window operations

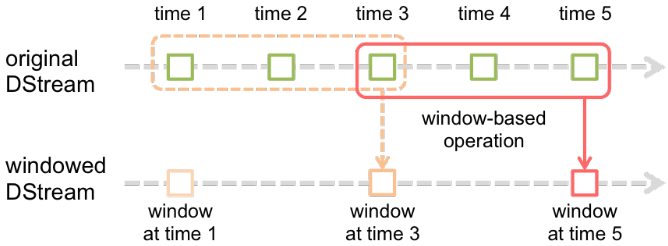

Image 5 - DStream Window

However, DStream is not great for Stateful streams and can't handle late data. There are other alternatives!

DataFrame/DataSet

- Introduced in Spark 2.0

DataFrame/DataSet

- Introduced in Spark 2.0
- Built on top of RDDs

DataFrame/DataSet

- Introduced in Spark 2.0
- Built on top of RDDs
- Imposes a structure over data - columns/rows

DataFrame/DataSet

- Introduced in Spark 2.0
- Built on top of RDDs
- Imposes a structure over data - columns/rows
- Dataset has two APIs - strongly-typed and untyped

DataFrame/DataSet

- Introduced in Spark 2.0
- Built on top of RDDs
- Imposes a structure over data - columns/rows
- Dataset has two APIs - strongly-typed and untyped
- `Row` - generic untyped JVM object

DataFrame/DataSet

- Introduced in Spark 2.0
- Built on top of RDDs
- Imposes a structure over data - columns/rows
- Dataset has two APIs - strongly-typed and untyped
- `Row` - generic untyped JVM object

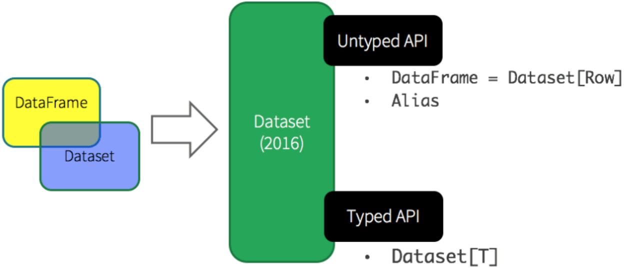

Image 6 - DataFrame/DataSet

DataFrame/DataSet

- Introduced in Spark 2.0
- Built on top of RDDs
- Imposes a structure over data - columns/rows
- Dataset has two APIs - strongly-typed and untyped
- `Row` - generic untyped JVM object

Language	Main Abstraction
Scala	Dataset[T] & DataFrame (alias for Dataset[Row])
Java	Dataset[T]
Python	DataFrame
R	DataFrame

DataFrame/DataSet

Structure Benefits:

- Syntax check, static-typing, runtime type-safety i.e. catch errors at compile time

DataFrame/DataSet

Structure Benefits:

- Syntax check, static-typing, runtime type-safety i.e. catch errors at compile time
- Simpler high-level operations - `agg`, `filter`, `groupBy`...

DataFrame/DataSet

Structure Benefits:

- Syntax check, static-typing, runtime type-safety i.e. catch errors at compile time
- Simpler high-level operations - `agg`, `filter`, `groupBy`...
- Uses Spark SQL - has Catalyst Optimizer which generates optimized logical and physical plan, together with Tungstan

DataFrame/DataSet

Structure Benefits:

- Syntax check, static-typing, runtime type-safety i.e. catch errors at compile time
- Simpler high-level operations - agg, `filter`, `groupBy`...
- Uses Spark SQL - has Catalyst Optimizer which generates optimized logical and physical plan, together with Tungstan

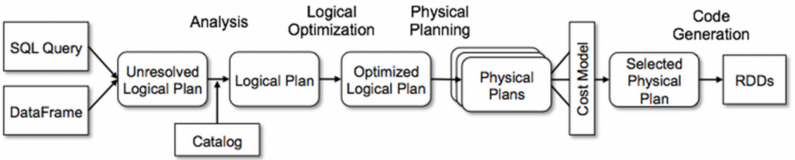

Image 7 - Catalyst Optimizer

DataFrame/DataSet

Structure Benefits:

- Syntax check, static-typing, runtime type-safety i.e. catch errors at compile time
- Simpler high-level operations - agg, `filter`, `groupBy`...
- Uses Spark SQL - has Catalyst Optimizer which generates optimized logical and physical plan, together with Tungstan

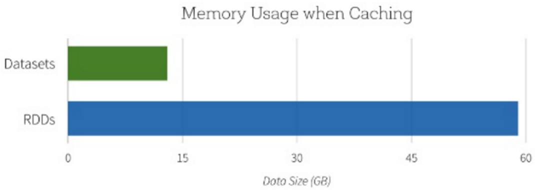

Image 8 - Memory Usage

Structured Streaming

Builds on top of structure provided by DataFrames/DataSets, which has its benefits:

- Fast and scalable

Structured Streaming

Builds on top of structure provided by DataFrames/DataSets, which has its benefits:

- Fast and scalable
- Fault-tolerant

Structured Streaming

Builds on top of structure provided by DataFrames/DataSets, which has its benefits:

- Fast and scalable
- Fault-tolerant
- End-to-end exactly-once stream

Structured Streaming

Builds on top of structure provided by DataFrames/DataSets, which has its benefits:

- Fast and scalable
- Fault-tolerant
- End-to-end exactly-once stream
- Stream and Batch are the same, we develop once

Structured Streaming

Builds on top of structure provided by DataFrames/DataSets, which has its benefits:

- Fast and scalable
- Fault-tolerant
- End-to-end exactly-once stream
- Stream and Batch are the same, we develop once
- Provides Event Time handling and Late Data processing - Watermarking

Structured Streaming

Builds on top of structure provided by DataFrames/DataSets, which has its benefits:

- Fast and scalable
- Fault-tolerant
- End-to-end exactly-once stream
- Stream and Batch are the same, we develop once
- Provides Event Time handling and Late Data processing - Watermarking
- Perform complex SQL queries

Structured Streaming

How it works:

- Unbounded Input Table

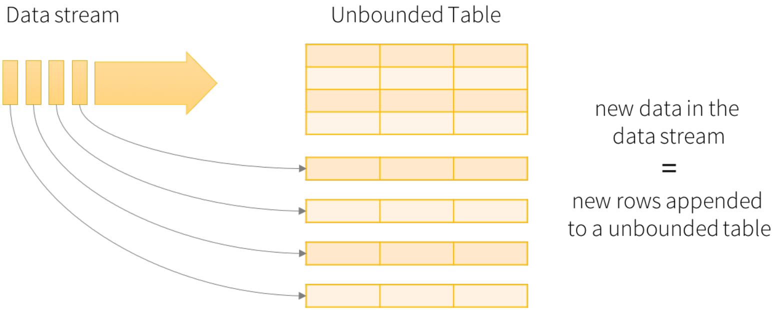

Image 9 - Unbounded Table

Structured Streaming

How it works:

- Result Table

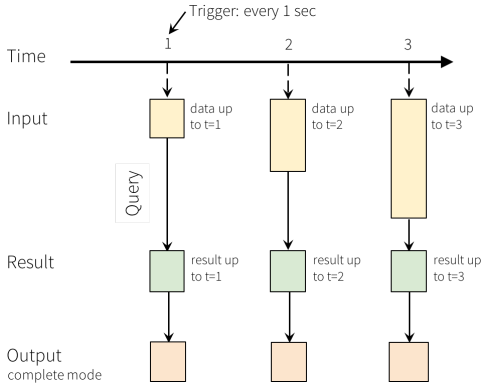

Image 10 - Result Table

Structured Streaming

- Output Modes - defines what gets written to the external storage

Structured Streaming

- Output Modes - defines what gets written to the external storage
    - **Complete** - Complete result tables are written

Structured Streaming

- Output Modes - defines what gets written to the external storage
    - **Complete** - Complete result tables are written
    - **Append** - Only rows that get appended since the last trigger

Structured Streaming

- Output Modes - defines what gets written to the external storage
  - **Complete** - Complete result tables are written
  - **Append** - Only rows that get appended since the last trigger
  - **Update** - Only rows that were updated

Structured Streaming

- Output Modes - defines what gets written to the external storage
  - **Complete** - Complete result tables are written
  - **Append** - Only rows that get appended since the last trigger
  - **Update** - Only rows that were updated

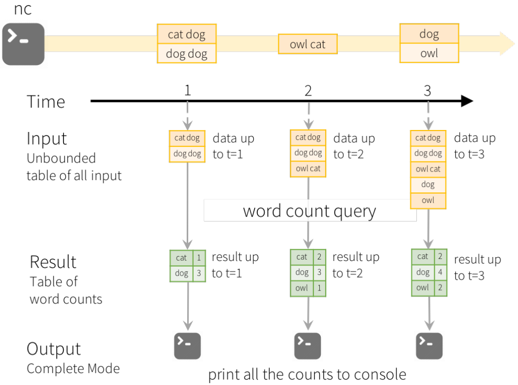

Image 11 - Query Example

Structured Streaming

Window operations

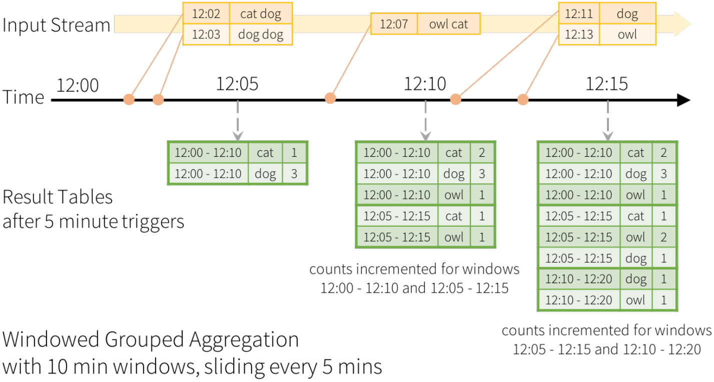

Image 12 - Window Operation

Watermarking

- Introduced in Spark 2.1

Watermarking

- Introduced in Spark 2.1
- Keeps track of event time in data

Watermarking

- Introduced in Spark 2.1
- Keeps track of event time in data
- Engine maintains state and allow late data for specific window `T`

Watermarking

- Introduced in Spark 2.1
- Keeps track of event time in data
- Engine maintains state and allow late data for specific window `T`
- Regulates state - cleaning

Watermarking

- Introduced in Spark 2.1
- Keeps track of event time in data
- Engine maintains state and allow late data for specific window `T`
- Regulates state - cleaning
- Example:

```
words = ...  # streaming DataFrame

# Group the data by window
windowedCounts = words \
    .withWatermark("timestamp", "10 minutes") \
    .groupBy(
        window(words.timestamp, "10 minutes", "5 minutes"),
        words.word) \
    .count()
```

Watermark

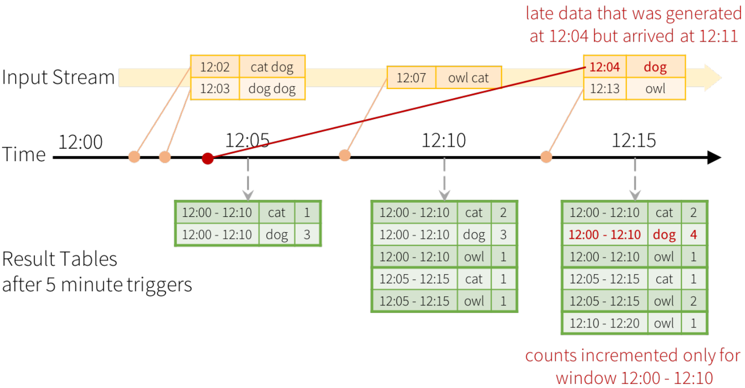

Image 13 - Watermark

THANKS!

If you have any questions, please don't hesitate to ask

References

- Spark Official Documentation
  - RDD Programming Guide
  - Streaming Programming Guide
  - Structured Streaming Programming Guide
- Databricks
  - A Tale of Three Apache Spark APIs: RDDs vs DataFrames and Datasets
  - Multiple Stateful Operators in Structured Streaming

Images

- Image 1 - Partitions - Source
- Image 2 - RDD Operations - Source
- Image 3 - Spark Streaming - Source
- Image 4 - DStream RDDs - Source
- Image 5 - DStream Window - Source
- Image 6 - DataFrame/DataSet - Source
- Image 7 - Catalyst Optimizer - Source
- Image 8 - Memory Usage - Source
- Image 9 - Unbounded Table - Source
- Image 10 - Result Table - Source
- Image 11 - Query Example - Source
- Image 12 - Window Operation - Source
- Image 13 - Watermark - Source