Spark Streaming and Windowing

What is Spark Streaming?

- Spark Streaming is a real-time data processing engine built on Apache Spark.
- It allows processing of continuous data streams in micro-batches.
- Supports data sources like Kafka, Flume, HDFS, and more.

PySpark Code: Kafka → JSON Parsing → Parquet

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import from_json, col
from pyspark.sql.types import StructType, StringType, IntegerType

# Step 1: Create Spark session
spark = SparkSession.builder \
    .appName("KafkaJSONToParquet") \
    .getOrCreate()

# Step 2: Define JSON schema (based on expected Kafka message structure)
schema = StructType() \
    .add("id", IntegerType()) \
    .add("name", StringType()) \
    .add("city", StringType())

# Step 3: Read from Kafka
kafka_df = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:9092") \
    .option("subscribe", "test-topic") \
    .load()
Step 4: Extract value as STRING and parse JSON
json_df = kafka_df.selectExpr("CAST(value AS STRING) as message") \
    .withColumn("data", from_json(col("message"), schema)) \
    .select("data.*")  # Optional: Expand fields directly

# Step 5: Write to Parquet
query = json_df.writeStream \
    .format("parquet") \
    .option("path", "/path/to/output/parquet") \
    .option("checkpointLocation", "/path/to/checkpoint/dir") \
    .outputMode("append") \
    .start()

query.awaitTermination()
```

Checkpointing in Spark Streaming

Checkpointing is the process of saving the **intermediate state and progress** of a streaming application to **reliable storage**, enabling recovery from failures.

Why is it Needed?

- **Recover** from driver/node failures
- **Maintain state** across batches
- **Support stateful ops** like windowing, aggregation
- Ensure **exactly-once guarantees**

Types of Checkpointing

Type	Purpose
Metadata	Saves query info (DAG, offsets, config)
Data (RDD)	Stores data lineage for recomputation
State	Saves state info for stateful aggregations

How to enable Checkpointing in Spark Streaming?

Step-by-Step Setup

1.Choose a persistent storage path

- Recommended: **HDFS**, **Azure Data Lake (ADLS)**, **Amazon S3**, or **other distributed file systems**
- Avoid: local filesystem (not fault-tolerant)

2.Add .option("checkpointLocation", "/path") in writeStream

Sample Code (Kafka to Parquet with checkpointing)

```
query = parsed_df.writeStream \
   .format("parquet") \
   .outputMode("append") \
   .option("path", "/output/parquet/") \
   .option("checkpointLocation", "/checkpoint/my_stream/") \
   .start()
```

Mandatory for:

- Aggregations
- Windowed queries
- Joins
- mapGroupsWithState or flatMapGroupsWithState

Best Practices

- Use **durable storage**: HDFS / ADLS / S3
- Don’t delete checkpoint folder manually
- Keep path **unique** for each stream

What is Windowing in Spark Streaming?

- Windowing allows processing of data within a specific time range.
- Helps in analyzing time-based events efficiently.
- Commonly used for aggregations, trend analysis, and pattern detection.

Types of Windowing in Spark Structured Streaming

- **1. Tumbling Window**
- **2. Sliding Window**
- **3. Session Window**
- **4. Watermarking (Not a window type but related)**
  - Helps handle **late data**.
  - Defines how long Spark waits for late data before finalizing a window.

```
df.withWatermark("timestamp", "10 minutes")
```

Tumbling Windows

- Fixed-size, non-overlapping time intervals.
- Each window processes events occurring within that time frame.
- Example: Counting events every 10 seconds.
- Example Code:
- windowed_df = df.groupBy(window("timestamp", "10 seconds")).count()

Sliding Windows

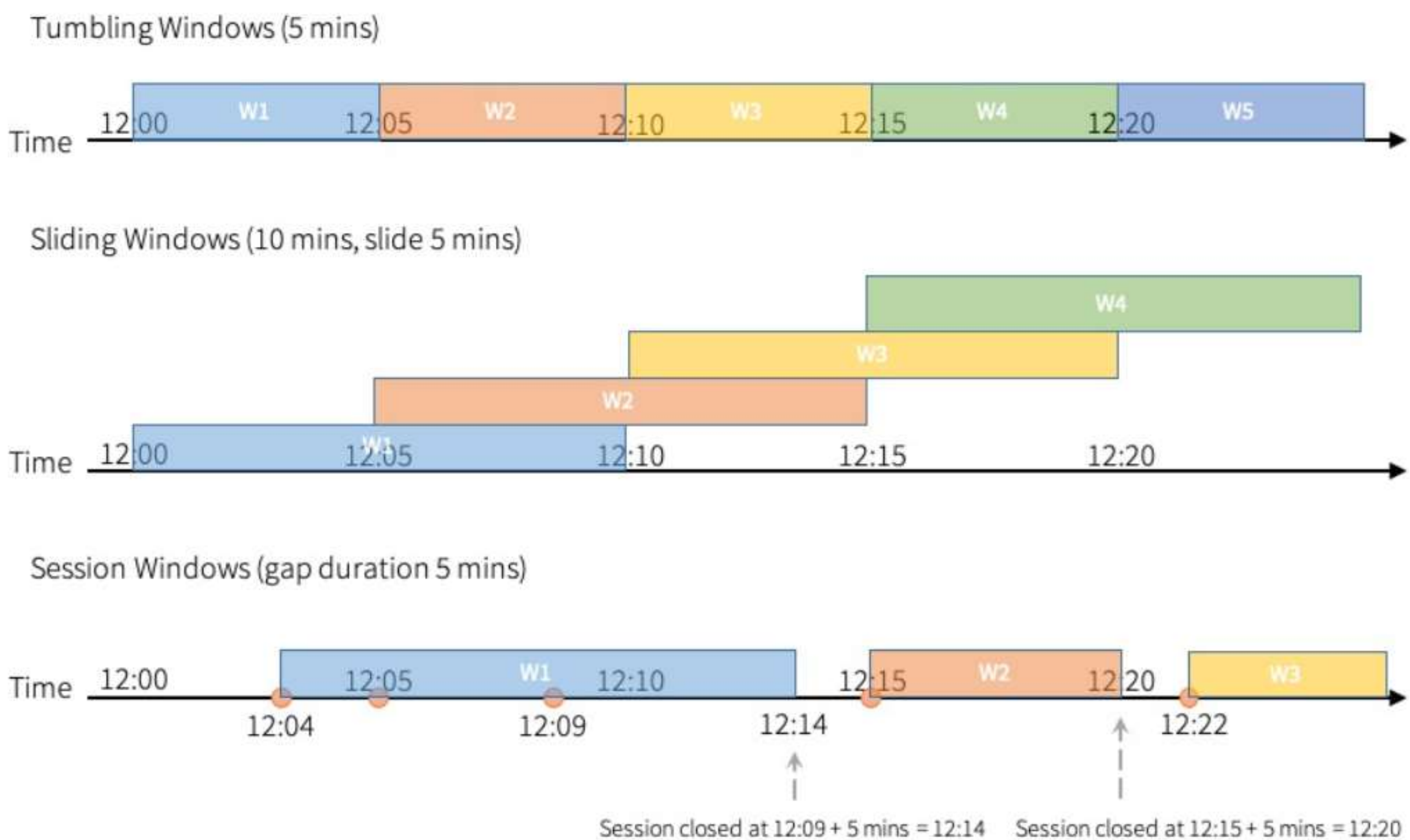
- Fixed-size windows that slide at a defined interval.
- Overlapping time windows allow event processing multiple times.
- Example: Counting events every 10 seconds with a slide interval of 5 seconds.
- Example Code:
- windowed_df = df.groupBy(window("timestamp", "10 seconds", "5 seconds")).count()

Session Windows

- Dynamic-sized windows based on user activity.
- A session window closes when there is inactivity beyond a threshold.
- Example: Tracking user sessions with a timeout of 30 minutes.
- Example Code:
- windowed_df = df.groupBy(session_window("timestamp", "30 minutes")).count()

Types of time windows

Spark supports three types of time windows: tumbling (fixed), sliding and session.



- **Tumbling windows** are a series of fixed-sized, non-overlapping and contiguous time intervals. An input can only be bound to a single window.
- **Sliding windows** are similar to the tumbling windows from the point of being “fixed-sized”, but windows can overlap if the duration of slide is smaller than the duration of window, and in this case an input can be bound to the multiple windows.
- **Tumbling and sliding window** use window function, which has been described on above examples.
- **Session windows** have different characteristic compared to the previous two types. Session window has a dynamic size of the window length, depending on the inputs. A session window starts with an input, and expands itself if following input has been received within gap duration. For static gap duration, a session window closes when there’s no input received within gap duration after receiving the latest input.
- **Session window** uses session_window function. The usage of the function is similar to the window function.

Use Cases of Windowing

- Fraud detection in real-time transactions.
- Log analysis and monitoring.
- Real-time user engagement tracking.
- IoT sensor data processing.

Thank You!

- Happy Learning ☺
- Feel free to ask any questions!