Pontifical Catholic University of Rio Grande do Sul
Graduate Program in Computer Science

Data Stream Processing with Apache Kafka and Spark Structured Streaming

Carlos Henrique Kayser
Email: carlos.kayser@edu.pucrs.br

Scalable Data Stream Processing
Prof. Dr. Dalvan Jair Griebler

May 15th, 2022

Introduction

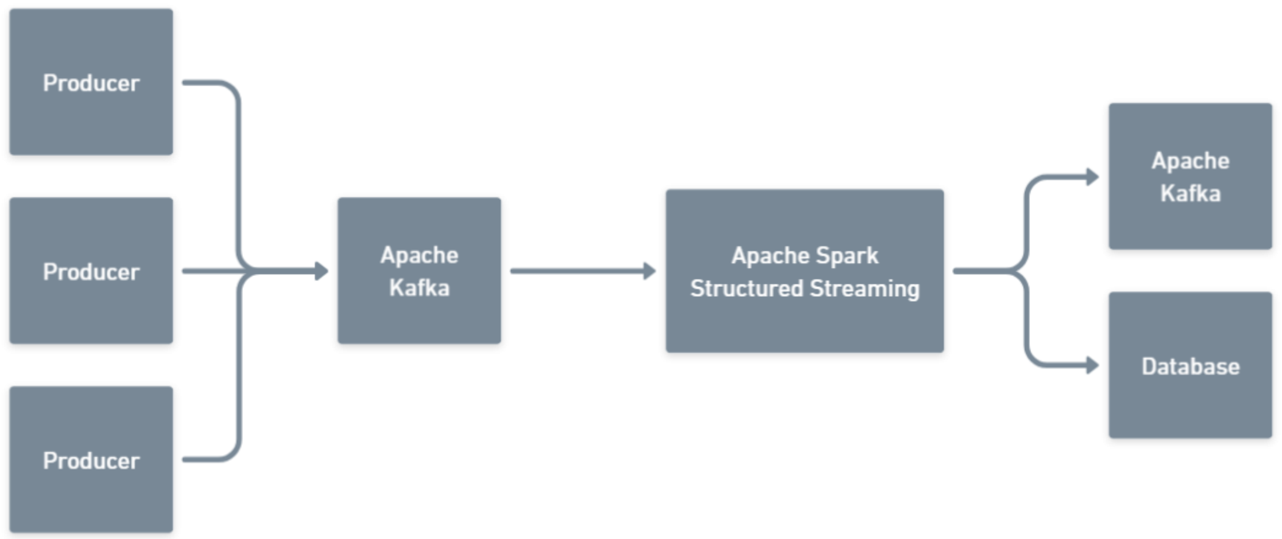
- In the big data era, large volumes of data are generated every day
- According to Marr in 2018, 90% of the data at that time were generated only in the last two years [7]
- In order to extract useful insights, this data needs to be analyzed
- In this way, efficient Data Stream Processing Systems (DSPSs) have become essential [6]

Introduction

- With all of this data, many organizations want to make their processes data-driven
  - In order to reduce costs and be more competitive
- One great example could be the **adversiment market**
- Predict **whether an ad will be clicked or not**, can reduce costs and increase profits
- However, processing a large volume of events in **real time** is a challenge

System proposal

- To takle the scenario presented, a possible architectural solution would be:
    - Apache Kafka to handle user-generated events;
    - Apache Spark Structured Streaming to apply a predictive machine learning model on the data stream
        - i.e., to predict whether an ad will be clicked or not



Apache Kafka Basics

- Kafka is a distributed event streaming plataform
- The data is organized into *topics* (e.g., tweets, orders)
    - To enable parallelism, they are split into *partitions*
- A *producer* append data/messages to *topics*
- A *consumer* read data from *topics*
    - They also can *subscribe* to a *topic* and receive incoming records as they arrive

Apache Spark Structured Streaming Basics

- Structured Streaming is a scalable and fault-tolerant stream processing engine
- It provides an unified batch and streaming API that enables us to interact with data published to Kafka as a DataFrame
  - i.e., it is possible to use the same code for batch or streaming
- It ensures end-to-end exactly-once fault-tolerance guarantees through *checkpointing* and *write-ahead logs*
- Structured Streaming queries are processed using a *micro-batch* processing engine (default)
  - i.e., *micro-batch* starts as soon as the previous one ends

Reading data from Kafka

- The following PySpark code read data from `demo` Kafka topic (*subscribe*)
- It reads the data in a streaming way (*startingOffsets*)
  - "latest" to read only the new messages
  - "earliest" to read all messages that have not been processed

```
df = spark \
  .readStream \
  .format("kafka") \
  .option("kafka.bootstrap.servers", KAFKA_HOST) \
  .option("subscribe", "demo") \
  .option("startingOffsets", "latest") \
  .load()
```

Writing data to Kafka

- When writing data, Apache Spark requires a `checkpointLocation` to store all data related to the execution
  - In case of failure or shutdown, it is possible to recover the previous progress and state
- The command below write data to `demo` Kafka topic.

```
df.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)") \
  .writeStream \
  .format("kafka") \
  .option("kafka.bootstrap.servers", KAFKA_HOST) \
  .option("topic", "demo") \
  .option("checkpointLocation", "checkpointLocation") \
  .start()
```

Writing data to not supported output sink

- Foreach Sink
  - Allows custom write logic on every row
  - Fault-tolerant: at-least-once
- ForeachBatch Sink
  - Allows arbitrary operations and custom logic on the output of each micro-batch
  - Reuse existing batch data sources
  - Write to multiple locations
  - Fault-tolerant: Depends on the implementation

Writing data to not supported output sink

Example of writing data in PostgreSQL:

```
def foreach_batch_function(df, epoch_id):

    df.write \
      .format("jdbc") \
      .option("url", "jdbc:postgresql://localhost:5432/postgres") \
      .option("driver", "org.postgresql.Driver") \
      .option("dbtable", "predictions") \
      .option("user", "postgres") \
      .option("password", "postgres") \
      .mode("append") \
      .save()

df \
  .writeStream \
  .foreachBatch(foreach_batch_function) \
  .option("checkpointLocation", "checkpointLocation") \
  .start()
```

Demonstration

- Testbed
  - minikube
- Apache Kafka on Kubernetes with Strimzi Operator
  - Apache Kafka Topics
- Producer (python application)
- Apache Spark Structured Streaming Application (jupyter)

Conclusions

- In conclusion, implementing code using Spark API is easy
  - Uses the same API for batch or streaming
- Integration with Apache Kafka is easy
- It is simple to scale the solution to support a larger number of messages
- In addition, Apache Spark Strutured Streaming is used in production envinronment of big techs [1]

References

[1] Armbrust, Michael, et al. "Structured streaming: A declarative api for real-time applications in apache spark." *Proceedings of the 2018 International Conference on Management of Data*. 2018.

[2] Apache Spark. "Structured Streaming Programming Guide". Source: https://spark.apache.org/docs/3.2.1/structured-streaming-programming-guide.html, June 2022.

[3] Apache Spark. "Structured Streaming + Kafka Integration Guide". Source: https://spark.apache.org/docs/3.2.1/structured-streaming-kafka-integration.html, June 2022.

[4] Apache Kafka. "Apache Kafka Documentation". Source: https://kafka.apache.org/32/documentation.html, June 2022.

[5] Das, Tathagata, et al. "Processing Data in Apache Kafka with Structured Streaming in Apache Spark 2.2". Source: https://databricks.com/blog/2017/01/19/real-time-streaming-etl-structured-streaming-apache-spark-2-1.html, June 2022.

[6] Eskandari, Leila, et al. "I-Scheduler: Iterative scheduling for distributed stream processing systems." *Future Generation Computer Systems* 117 (2021): 219-233.

References

[7] Marr, Bernard. “How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read”. Forbes. May 21, 2018