Comparative Evaluation of Spark and Flink Stream Processing

Contents

Abstract

Recent years have witnessed a big development improvements of Big Data frameworks such as Apache Spark and Apache Flink. For example, both frameworks support real-time streaming processing. While which platform is the best still an open question. In this paper, we provide a performance comparison of streaming data processing in Spark and Flink, by measuring different performance metrics, namely latency and throughput. The experiment is done using multiple streaming workloads over real-world datasets.

1 Introduction

Big Data is now one of the hot topics in Information Technology, it has gained increasing attention of professionals in industry and academics, since the amount of data we generate is increasing exponentially; generated from social media platforms, mobile phones, IoT and finical transactions. Moreover, recent years showed a rapid development of Big Data frameworks such as Spark and Flink that provide more and efficient data processing capabilities.

The Big Data is a method to process not just only just large datasets (volume), but also that arrives in high rate (velocity), and it comes in all kinds of data format (variety) (i.e., 3Vs of Big Data (Volume, Velocity, Variety))[1].

The processing models of Big Data are Batch and Stream processing, Batch processing is a way to handle large, finite volumes of data (e.g., processing of historical transaction records), is more concerned with throughput. While the Stream processing is a method to manage fast and continuously incoming data (real-time), process it as it arrives (e.g., credit card fraud detection and network monitoring applications), where the low latency is required. The variety of data (structured and semi-structured) is included in the two processing models.

The purpose of this paper is to provide a comparative experimental evaluation of throughput, latency (i.e., performance) of stream processing in Apache Spark and Apache Flink. We use a datasets of airplanes trajectories provided in the context of Datacron project[1], in order to simulate real-world use cases. Using multiple streaming data workloads we try to cover the main programming API differences between the both frameworks. Informally, this work aims to help developer to determine which platform to choose in production.

The rest of this paper is organized as follows: Section 2 presents the main characteristics, APIs, and main data abstraction of Spark and Flink. Section 3 describes the experiment workloads and their implementation in each framework. Section 4 provides and compares the performance results of the different workloads. Finally, Section 5 gives the overall conclusion.

[1] `http://www.datacron-project.eu/`

2 Technical Background

In this section, we present the architecture, key characteristics, programming APIs of Apache Spark, Apache Flink. The Apache Kafka platform is also discussed briefly.

2.1 Apache Spark

Apache Spark is an open source project that provide general framework for large-scale data processing [2]. It offers programming API in Java, Scala, Python and R. Its stack includes set of built-in modules including Spark SQL, MLlib for machine learning, GraphX for graph processing, and Spark Streaming to process real-time data streams. It can access different data sources including Hadoop Distributed File System (HDFS), Apache Cassandra database, Apache HBase database etc.

The main data abstraction in Spark is the Resilient Distributed Datasets (RDDs), which is a in-memory collection of elements partitioned across cluster of computers that can be processed in parallel [3]. RDDs supports two categories of operations: transformations that drive new RDDs from the operated ones (e.g., map). And actions which return a value (e.g., count).

//TODO: DStreams

2.2 Apache Flink

Apache Flink is an open source project that provide large-scale, distributed stream processing platform, while the batch processing treated as special case of streaming applications [4]. It offers programming API in Java, Scala. Its stack includes the core DataStream and DataSet APIs with additional libraries such as Complex event processing for Flink (FlinkCEP), Machine Learning for Flink (FlinkML), and Flink Graph API (Gelly).

The main data abstraction of Flink are DataStream and DataSet that represents read-only collection of elements of the same type. The list of elements is bounded (i.e., finite) in DataSet, while it is unbounded (i.e., infinite) in case of DataStream.

2.3 Apache Kafka

Apache Kafka is scalable, fault-tolerant distributed publish/subscribe streaming framework [5]. It manges the stream records in different categorizes (i.e., topics) that portioned and distributed over the servers in the Kafka cluster. It allows the data producers to publish stream of records to certain Kafka topic. While the consumer applications can subscribe to one or more topic to read the data stream. It's widely adopted, for example, Spark and Flink can receive data stream from Kafka.

3 Experiment Setup and Implementation

This section describes the data stream setup, design of evaluation streaming workloads, and implementation details in Spark and Flink.

3.1 Data Stream Setup

Our streaming workloads read input data stream from Kafka. In order to simulate real-world uses cases we use datasets of Automatic Dependent Surveillance Broadcast (ADS-B) messages that present the aircrafts position over time, comprise 22 fields of data such as AircraftID, Date message generated, longitude, latitude, and altitude.

In our experiment a data stream producer component reads the ADS-B datasets, then publish it to Kafka cluster to be consumed by the workloads in Spark and Flink. Figure 1 illustrates the data producer and the Kafka cluster setup. The data stream is portioned into four portions over two server, while the data producer publish the stream records randomly to Kafka partitions.

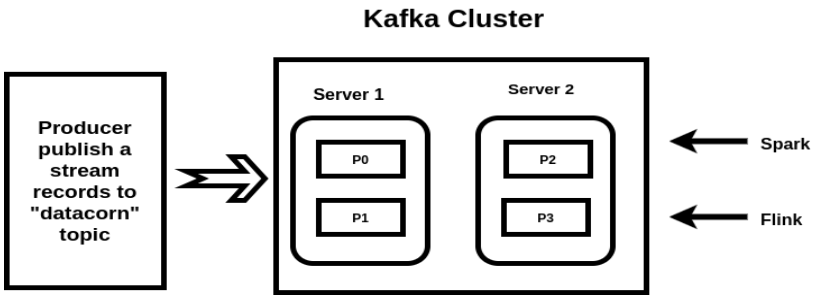

Figure 1: Data Stream Producer & Kafka Cluster Setup.

3.2 Workloads Design

In our experimental evaluation, we developed two streaming processing workloads that read the ADS-B messages stream from Kafka to perform basic trajectories analysis methods. In our design choice, we try to cover key main points related to the streaming processing, to evaluate how does each framework perform. The following are general key aspects of streaming data processing were covered by our workloads design:

- Handling parallel input streams.
- How to aggregate the state of input stream records.
- Mange the order of stream records.
- How to provide and update global data model in stream processing task.

- Evaluate the performance by measuring the latency and throughput.

We give in Section 3.3 how Spark and Flink are handling these key aspects and related issues, then we discuss the performance evaluation in Section 4.

3.2.1 Statistics Computation per Trajectory

In first stream processing workload, we perform statistics calculation for each trajectory (i.e., messages of the same ID) in ADS-B messages stream. As example of computed statistics indexes speed mean, mean of location coordinates, min and max altitude, etc. In this workload, we try to cover the parallel receiving of input data stream, state-full stream aggregation, and preserve the order of stream records.

3.3 Workloads Implementation

4 Performance Results

This section provides and compares the performance results of the different workloads.

5 Conclusion

This section gives the overall conclusion.

References

[1] Laney, D 2001 3D Data Management: Controlling Data Volume, Velocity, and Variety. META Group.

[2] Apache Spark. Available: https://spark.apache.org/ [Accessed February 2017].

[3] Zaharia, Matei, et al. "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing." Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. USENIX Association, 2012.

[4] Apache Flink. Available: https://flink.apache.org/ [Accessed February 2017].

[5] Apache Kafka. Available: https://kafka.apache.org/ [Accessed February 2017].