Comparative Evaluation of Spark and Flink Stream Processing

February 19, 2017

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 volumes of data (e.g., processing of historical transactions), 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).

3 Experiment Setup and Implementation

This section describes the experiment workloads and their implementation in each framework.

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.