International Journal of Fog Computing
Volume 2 • Issue 1 • January-June 2019

A Study on the Performance and Scalability of Apache Flink Over Hadoop MapReduce

Pankaj Lathar, CBP Government Engineering College, New Delhi, India

K G Srinivasa, CBP Government Engineering College, New Delhi, India

ABSTRACT

With the advancements in science and technology, data is being generated at a staggering rate. The raw data generated is generally of high value and may conceal important information with the potential to solve several real-world problems. In order to extract this information, the raw data available must be processed and analysed efficiently. It has however been observed, that such raw data is generated at a rate faster than it can be processed by traditional methods. This has led to the emergence of the popular parallel processing programming model – MapReduce. In this study, the authors perform a comparative analysis of two popular data processing engines – Apache Flink and Hadoop MapReduce. The analysis is based on the parameters of scalability, reliability and efficiency. The results reveal that Flink unambiguously outperformance Hadoop's MapReduce. Flink's edge over MapReduce can be attributed to following features – Active Memory Management, Dataflow Pipelining and an Inline Optimizer. It can be concluded that as the complexity and magnitude of real time raw data is continuously increasing, it is essential to explore newer platforms that are adequately and efficiently capable of processing such data.

KEYWORDS

Apache Flink, Big Data, Cyclic Data Flow Engine, Fog Computing, MapReduce, TeraSort

INTRODUCTION

The term Big Data is used to describe datasets so large and complex that traditional database applications prove to be inadequate to process the data. The data may be structured or unstructured in nature. Big data is best characterized by the 3Vs – Volume, Variety and Velocity (Alexandrov et al., 2014). As each of these parameters increase, it becomes more and more taxing to handle data (Currey et al., 2013). This has led to the emergence of a new data processing model – MapReduce (Dean & Ghemawat, 2008). MapReduce is a programming model for distributed computing systems. Unlike traditional data processing methods, MapReduce harnesses the power of parallel processing (Carbone et al., 2015).

The MapReduce algorithm contains two important tasks, namely Map and Reduce. The Map task tokenizes the input data set and maps each token to a relevant key. The Reduce task collects the output from a map phase and preforms an aggregate operation on the values identified by the same key. Flink (Alexandrov et al., 2014) extends the MapReduce paradigm to include several higher order operations. Flink provides a generic iterative runtime engine which is better described as a Cyclic Data Flow Engine (Lin & Liu, 2013).

DOI: 10.4018/IJFC.2019010103

SALIENT FEATUERS OF FLINK

The following section gives a deep insight into prominent architectural advancement in Apache Flink.

Common Runtime – Stream and Batch Processing

Flink offers a common runtime environment for stream and batch processing. In fact, Flink is typically a data-stream processor which treats batch data as a special case of streaming data.This is in contrast to how most data processing engines treat streaming data as micro-batches. Flink's innate ability to deal with streaming data makes it capable of dealing efficiently with real time data (O'Malley, 2008).

Active Memory Management

Several data processing engines (including Hadoop and Flink) are implemented using Java. The major concern with any JVM based implementation is efficient management of the heap. All processor intensive tasks run in memory. Hence, the datasets must be present in memory before being operated upon. However, the size of the main memory is often much less than the size of the dataset leading to *OutOfMemoryErrors*. Another major drawback associated with JVM based engines are the stalls incurred due to garbage collection. Overhead spent in the garbage collection of several objects can be taking a toll on overall system throughput. Moreover, java objects have some amount of overhead space which depletes the amount of overall memory available (Waas, 2008).

Apache Flink combats these problems associated with memory management using the concept of serialization (Stephan Ewen, 2015). Instead of burdening the heap, Flink serializes objects into a fixed number of pre-allocated memory segments. If data to be processed exceeds the size of the available memory, the serialized objects are spilled to the disk. When the need for these objects surfaces, they are de-serialized and brought back to memory. Moreover, the binary representation of objects uses far less memory. The problem associated with garbage collections is dealt with by reusing short-lived objects. Figure 1 depicts memory management in Flink

Program Optimizer

Program written in Flink, are not directly executed. Before execution, the job enters an intermediary *Cost-Based Optimization Phase*. In this phase, the Flink Optimizer, chooses the most optimum route for execution based on the dataset and the nature of the cluster. This feature makes it possible for the programmer to focus on the code and not the execution environment or the input dataset. This ensures increased productivity of the programmer and enhanced utilization of the cluster.

Figure 1. Active memory management in Flink