ISSN XXXX XXXX © 2017 IJESC

Research Article Volume 7 Issue No.10

Big Data Technologies for Batch and Real-Time Data Processing: A Review

Akshit Dhar
UG Student
Department of Information Science Engineering
Rajarajeswari College of Engineering, Bangalore, Karnataka, India

Abstract:
With the rampant growth in the generation of data from various sources, there is a necessity to analyze and process this huge amount of data effectively. Data, available in batch and real-time requires being processed actively. Big Data technologies provide effective frameworks to handle voluminous data efficiently. This paper reports review on data processing using Big Data frameworks: Apache MapReduce, Apache Spark and Apache Storm, for data available in batch and real-time. The conclusion is summarized based on the comparison between the frameworks on several variables.

Keywords: Big Data, Big Data Technologies (BDT), Google File System (GFS), Hadoop Distributed File System (HDFS), Relational Database Management Systems (RDBMS), Resilient Distributed Datasets (RDDs)

I. INTRODUCTION

Over 2.5 exabytes (2.5 billion gigabytes) of data is being generated every day from various sources such as social media, stock markets, sensor networks, Geo-spatial data, Internet blogs, health sectors etc. As forecasted by Gartner, Inc. [1], there will be roughly 20.4 billion connected things by 2020. A large stock exchange captures more than 1 terabyte of data every day whereas YouTube users upload more than 48 hours of videos every minute which narrow down to 24 terabytes of data a day.

Big data is the data that is so huge that it can't be processed using ancient data handling techniques. In other words, the traditional RDBMS techniques fail to analyze this massive pile of data that is generating at a tremendous rate. The problems faced with huge data include real time analytics, cleaning unstructured data, scalability, accessibility, the type of data and many more.

[2] Laney (2001) suggested the three dimensions of Big Data: **Variety, Velocity, and Volume.** *Variety* envelopes data in all structured, semi-structured and unstructured form which ranges from databases, XML through a text file.

Velocity records for streaming of data and movement of data at very high speed. Earlier data was processed using batch processing systems due to inefficient and expensive nature of data processing. With exponential generation of data, real-time processing has become a new standard.

Volume denotes a huge amount of data generated from E-commerce, sensors, social media and so on, ranging from terabytes to zettabytes. *Big data technologies* such as Apache Hadoop, Apache Spark, Apache Storm, Apache MapReduce and so on, provide frameworks to process data and gain insights from it.

Figure 1 shows the potential of big data technologies and why it is accepted to overcome problems posed by enormous data.

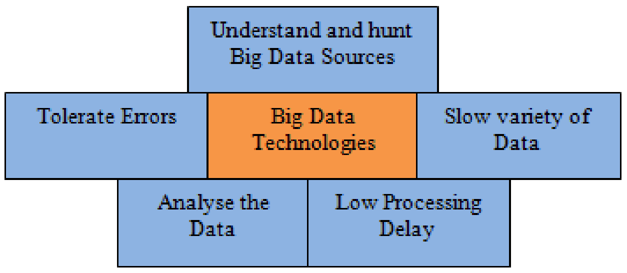

Figure.1. Big Data Potential

BDT can be used to understand and hunt down the source of big data origin which can be stock markets, hospitals, aeronautics centers to name a few. Data which is available in various forms can be stored and further can be processed faster compared to traditional processing techniques. Data can be analyzed for patterns in it such as generating insights from it. BDT can endure and handle faults effectively. In this paper, we will discuss various BDT. Section II will give an overview of different BDT frameworks. Section III will give a comparison. In Section IV, we reach to a conclusion of this paper based on the comparison.

II. OVERVIEW OF FRAMEWORKS

1. Apache Hadoop

Apache Hadoop is a framework based on Java and Google File System (GFS) [3] that supports processing of large data sets in a distributed computing environment. It possesses its own distributed file system, called Hadoop Distributed File System or HDFS [4] that allows quick data transfer among the nodes. This system is suitable only when it is written once and read multiple times. Moreover, it works flawlessly on low-end machines. *Figure 2* shows the structure of a Hadoop cluster. *Namenode* maintains the metadata of all the file system in a Hadoop Cluster. *Secondary Name Node* eyes for the proper working of the Namenode. If the Namenode fails, Secondary Namenode takes up its task. *Jobtracker* assigns the MapReduce task to the *Slave* nodes. *Datanodes* stores the data in HDFS which is replicated in at least three data nodes.

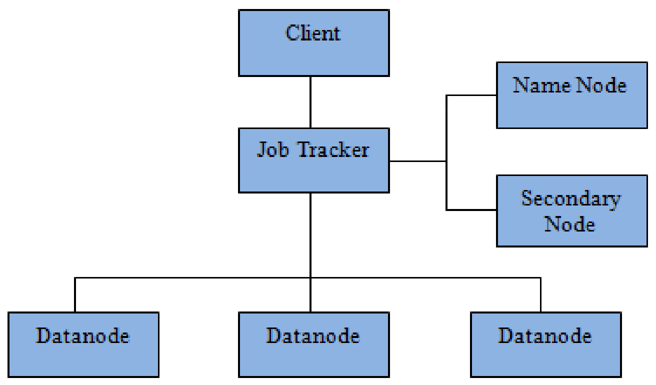

Figure.2. HDFS Cluster

The *client* that holds data analysis algorithm or map reduce program, contacts the *Namenode* that informs the client of the Data Node that holds the requested data. If *Namenode* fails, the *client* issues the same request to *Secondary Namenode*. The *client* contacts the *Job Tracker* to assign a MapReduce task to the Data Node and eventually get the result.

2. APACHE MAPREDUCE

MapReduce [5] is the batch processing model for Apache Hadoop. As the name suggests, it has two phases: Map and Reduce. In Map phase, the distributed parts of the dataset are assigned to 'mapper' that work in parallel to notch up big data computation. The results generated from map phase are sorted and shuffled. In Reduce phase, results, that in turn materialized are fed into 'reducers' which sums up the results into the solution to the original problem.

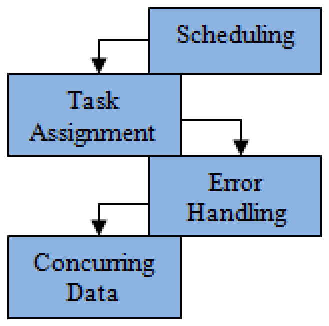

Figure.3. MapReduce Framework

Figure 3 describes the MapReduce Framework that takes care of the distributed processing and coordination. It involves four tasks: Scheduling, Task Assignment, Error Handling and Concurring Data. *Scheduling* deals with breaking large jobs into small chunks called tasks. These tasks are scheduled by Task Tracker. *Task Assignment* struggles to place tasks on the nodes where data segment to be processed, is hosted. The code is moved to the mapper that holds the data. *Error Handling* ensures tasks are executed on other machines if unexpected failures occur. *Concurring Data* is shuffled and sorted and moved between the machines. This data is further reduced to original solution to the problem much like divide and conquer algorithms.

3. APACHE SPARK

Apache Spark [6] is open source engine for efficient large-scale processing and is 100 times faster than Apache MapReduce in certain cases [7]. The general idea behind Spark is that computations are performed in-memory. Because of its exceptional performance, Spark is used for both batch processing and real time processing. Spark lacks its own distributed storage system, so it is generally installed on top of Apache Hadoop and thus uses Hadoop Distributed File System for storing data. Spark provides excellent fault tolerance and scalability. Resilient Distributed Disks (RDD) is the primary abstraction of data in Spark. These RDDs are an immutable collection of objects, records or elements separated into partitions.

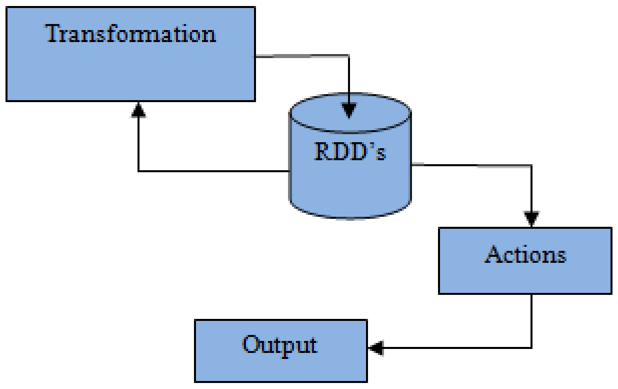

Figure.4. Spark Process

Figure 4 shows how the spark works. RDDs only support couple of operations: *Transformation* and *Actions*. A new RDD is created every time a transformation operation is carried out. Since RDDs are immutable, transformed RDDs are recomputed each time an action is run against it. The performance can be improved by caching to avoid recomputing. RDDs are computed when an action is called. The resulting data from Actions is sent to driver program.

4. APACHE STORM

Apache Storm [8] is an open source distributed framework for real time data processing. It is extremely fast, robust, highly scalable and fault tolerant system. The Storm architecture includes two kinds of nodes: Master and Worker nodes. Figure 5a shows architecture of Apache Storm. Master node of Storm cluster runs a background process (or daemon) called *NIMBUS* which is responsible for code distribution for worker nodes and oversee the failures. Similarly, each worker node runs a *Supervisor* daemon. A master node can *n* number of worker nodes. *Zookeeper Service* monitors the all the ongoing task updates by the worker nodes. NIMBUS service assigns task to worker nodes. Each worker node executes a topology.

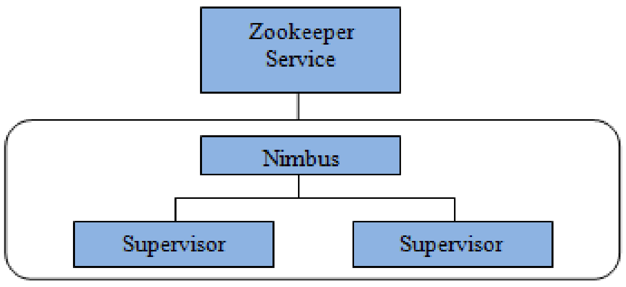

Figure.5.a. Storm Architecture

Figure 5b shows the working of *Supervisor* daemon. Each *Supervisor* executes topology: Directed Acyclic Graphs (DAG). Each topology consists of *Spout* and Bolt. *A Spout* is a stream of contiguous data from some external source. *Bolts* acts as actual processing tasks. A bolt can pass its output as input to other bolt. Thus, a bolt can receive input from a spout or other bolt.

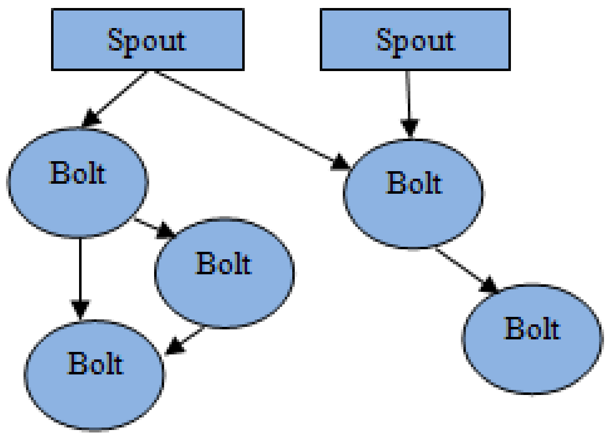

Figure.5. b: Storm Architecture

III. COMPARISON

This section provides a detailed comparison between Apache MapReduce, Apache Spark and Apache Storm based on numerous variables.

1. Speed
Apache Spark on Hadoop cluster is 100x faster in-memory and 10x faster in disk compared to Apache MapReduce for batch processing since it reduces read/write cycle to disk. Moreover, MapReduce does not support real time processing which clearly pushes it out of picture for comparison based on real-time processing. Apache Storm triumphs Apache Spark on real-time processing. Storm exhibits fairly small processing delay compared to Spark.

2. Development
MapReduce requires developers to design and hand code each operation to be performed which is annoying. Tools like *Impala* allows complete package for data querying. Spark on the other hand, offers over 80 high-level operators that aid in development of parallel applications. Spark uses Scala tuples and since it's difficult to implement Scala tuples in Java, Scala tuples can be reinforced by generic types nesting. However, compile time type safety checks are not jeopardized. Storm uses DAGs and each node in it manipulates data is some way. Streams of data are transferred between different nodes of DAGs using Storm tuples. This can be achieved by jeopardizing compile time type safety checks.

3. Latency
MapReduce is a high latency framework with few minutes delay whereas Spark is a low latency framework with few seconds delay. Storm framework outshines the other two with just a few sub seconds of delay.

4. Real-Time Analysis
MapReduce lags the ability to process real-time data since it was designed for batch processing. Spark framework uses Spark Streaming to process real-time data with low latency. Storm is specially designed for real-time processing of data and outperforms Spark Streaming with latency of few sub seconds.

5. Applicability
All BDT are widely used in every sector due to stacks of data piling up. Spark model is excellent for iterative machine learning and interactive analysis. Storm, on the other hand excels in near real-time analytics, natural language processing and various Extract Transform Load (ETL) operations. MapReduce is lagging when compared to more advance emerging BDT such as Spark, Storm and so on.

6. Contributors
From a past decade, there has been a good growth in contributions to open source technologies such as BDTs. Apache Hadoop has total of 109 contributors[9] and Apache Storm has 276 contributors[10] and wherein Apache Spark has surpassed other two with a count of 1,151 contributors[11]. This huge difference clearly indicates that the Spark is appreciated more by the developer community.

IV. CONCLUSION

From the previous section, we can reach a conclusion that Spark being better than MapReduce and Storm. Spark is generally an optimal framework that supports both batch and real-time processing unlike MapReduce and Storm which supports either. For real-time applications, Storm is optimal because of low latency.

V. ACKNOWLEDGEMENT

I would like to thank my family, teachers and friends for giving constant support and advices which proved to be very helpful and kept me going.

VI. REFERENCES

[1]. Gartner, Inc. Forecast for connected devices in the world by 2020. http://www.gartner.com/newsroom/id/3598917

[2]. Laney, D. (2001, February 6). 3D data management: Controlling data volume, velocity, and variety. META Group. Http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf

[3]. Sanjay Ghemawat, Howard Gobioff, and Shun-Tak: Google File System, 2003. https://static. Google user content. com/media/research.google.com/en//archive/gfs-sosp2003.pdf

[4]. Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler: The Hadoop Distributed File System, Yahoo! http://citeseerx.ist.psu.edu/ viewdoc/download?doi= 10. 1.1.178.989&rep=rep1&type=pdf

[5]. Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Cluster. https://static. googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf

[6]. Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Sto. Apache Spark: A Unified Engine for Big Data Processing. http://go.databricks.com/hubfs/docs/research-papers/p56-zaharia.pdf

[7] .Apache Spark. https://spark.apache.org/

[8]. Apache Storm. http://storm.apache.org/

[9]. Apache Hadoop GitHub. https://github. com/ apache/ hadoop

[10]. Apache Storm GitHub. https://github.com/apache/storm

[11]. Apache Spark GitHub. https://github.com/apache/spark