Big Data Analytics using Spark

Nisha Chandwani
Indiana University Bloomington
Bloomington, Indiana 47405
nchandwa@iu.edu

ABSTRACT

With Petabytes of data being generated every second, big data analytics has become one of the most talked about terms in the technological world. Many organizations are trying to use big data for deriving useful business insights in order to improve decision making. However, we need special tools and frameworks to analyze such large amounts of data. We discuss how big data can be efficiently analyzed using Apache Spark which is a memory based computing framework. We discuss the core components and the architecture of Spark along with its ecosystem that extends the capabilities of Hadoop MapReduce.

KEYWORDS

i523, HID203, Apache Spark, RDD, Big Data Analytics, Hadoop, MapReduce

1 INTRODUCTION

The growth of data has been following an exponential rate with huge amounts of data being generated every second. In today's world, having Terabytes or even Petabytes of data to deal with is not uncommon. The challenge lies not only in the volume of data but also in the large variance of the kind of data that has to be dealt with. This has led to the birth of one of the most talked about terms in today's technological world, i.e., Big Data. Most of the organizations today are collecting big data with the goal of extracting *value* from the exploratory analysis of this data and using this information to make business decisions. However, analyzing such enormous data is in itself a huge challenge and this is where big data analytics frameworks like Spark come to rescue. Spark is a general distributed computing framework that is optimized for in-memory processing. We show how Spark supports faster data analysis and is proving to be one of the most successful frameworks for Big Data Analytics.

2 SPARK

Spark is a general distributed computing framework which is based on Hadoop MapReduce algorithms[4]. However, using Hadoop MapReduce for complex tasks requires frequent disk I/O which make Hadoop less suited for low-latency tasks. To overcome this, Spark extends the capability of MapReduce by providing in-memory computing which enables it to query data much faster than disk-based engines like Hadoop[9]. Due to its memory computing capabilities, Spark is often used for iterative applications, such as Data Mining and Machine Learning[4].

Apache Spark has a well-defined architecture which is based on two main abstractions[3]:

- Resilient Distributed Datasets (RDD)
- Directed Acyclic Graph (DAG)

2.1 Resilient Distributed Datasets (RDD)

The entire framework of Spark is centered around RDD as it supports in-memory processing computation. This means it stores the state of memory in the form of an object across multiple jobs and the object is shared between these jobs[5]. RDD is a collection of data items that can be operated in parallel and is stored in memory or on disk. This parallel data computing structure is read-only and is distributed over a cluster of machines offering a restricted form of distributed shared memory. RDDs are maintained in a fault-tolerant way and can cache intermediate data across a set of nodes. Thus, RDDs enable Spark to efficiently support iterative algorithms[7].

RDD supports two types of operations[2]:

- Transformation: Operations like join, union, filter or map on existing RDDs which produce a new RDD as a result of the operation, are referred to as transformations.
- Action: Operations like count, first and reduce which evaluate an existing RDD and return values after computations are referred to as Actions.

2.2 Directed Acyclic Graph (DAG)

Spark consists of an advanced Directed Acyclic Graph (DAG) engine which allows programmers to develop complex, multi-step data pipeline[8]. Each Spark job creates a DAG of task stages to be executed on the cluster where each node in the DAG is an RDD partition and each edge represents a transformation to be applied on the data. This allows simple tasks to complete in a single stage whereas more complex tasks are completed in a single run of multiple stages, rather than splitting them into multiple jobs[10]. Thus, DAG abstraction eliminates the Hadoop MapReduce multi-stage execution model resulting in better performance[3].

3 SPARK ARCHITECTURE AND HARDWARE INTRODUCTION

Spark is built in programming language Scala and is run on Java Virtual Machine (JVM). In addition to Scala, it provides API for Java and Python as well. For running an application, Spark provides the following two options[10]:

- Interpreter in the Scala language distribution allows users to execute their queries on large data sets through Spark engine.
- Users can write their applications as Scala programs called driver programs. These driver programs can be then compiled and submitted to the cluster's master node.

Apache Spark uses a master/worker architecture as shown in Figure 1. It mainly consists of a driver program (SparkContext), workers (executors) and a cluster manager which are described below[3]:

[Figure 1 about here.]

- Driver Program: This program runs the main function of the Spark application. It is also responsible for the creation of the SparkContext object which basically coordinates the independent sets of processes running for an application on the cluster. The main components of the driver program are - DAGScheduler, TaskScheduler, BackendScheduler and BlockManager that translate the user code into Spark jobs that are executed on the cluster.
- Executor: These are the worker processes that are responsible for the execution of tasks sent by the SparkContext object. Some of these tasks include processing the data, reading from and writing data to external sources, performing computations and storing the results in in-memory cache or on hard disk drives.
- Cluster Manager: This is an external service that is responsible for acquiring resources on the Spark cluster and allocating them to the Spark jobs.

Being a memory-based computing platform, one of the most important factors of the Spark cluster is the memory. All the nodes, i.e., the driver and the executor nodes, should be equipped with at least 8 GB of memory for Spark to run well. For the cluster manager, Spark currently supports the below three deployments[4]:

- Standalone: It is a simple cluster manager included with Spark. Since Spark Standalone is available in the default configuration, it is the easiest way to set up a cluster and run applications on Spark.
- Apache Mesos: It is a general cluster manager that provides API for resource management and task scheduling across multiple nodes
- Hadoop YARN: It is the resource manager in Hadoop 2 which was added to Spark in version 0.6.

4 SPARK FOR BIG DATA ANALYTICS

With a large number of companies now looking to expand their advanced analytics capabilities, the ecosystem of Spark is right out of the box, making advanced analytics a reality. This ecosystem, as shown in Figure 2, provides an impressive set of high-level tools which include - Spark SQL for SQL, MLlib for machine learning, GraphX for graph processing and Spark Streaming[4]. Each of these components is-

[Figure 2 about here.]

- Spark Core API is the foundation of the overall ecosystem that provides task scheduling, dispatching and basic I/O functionalities. It is available through API in languages like Java, Python, Scala and R.
- Spark SQL is Apache's Spark module for supporting SQL implementation. It provides seamless integration of SQL queries with Spark programs. It provides a common way to connect to a variety of data sources such as Hive, JSON, JDBC, etc.
- Spark Streaming is an extension to the core Spark API which provides the capability to process streaming jobs along with batch jobs. The languages supported are Java, Python and Scala.
- MLlib is Spark's scalable Machine Learning library which is usable in Java, Python, Scala and R. MLlib supports high-quality algorithms such as classification, regression, clustering, recommendation, dimensionality reduction, etc. MLlib leverages Spark's excellence in iterative computing enabling it to run faster than MapReduce on huge datasets.
- GraphX is Spark's parallel computation API used for charts and graphs processing[4]. GraphX extends the capabilities of Spark RDD by introducing RDD graph which is a directed multi-graph with properties connected to each node and the edge[8].

One of the challenges of analyzing big data is that it can come in any shape or size. Thus, whether big data is to be processed offline (Spark Core) or on the fly (Spark Streaming), whether it is structured (Spark SQL) or connected in nature (GraphX), Spark ecosystem is a framework that can be widely used in big data analytics.

5 SPARK VERSUS HADOOP MAPREDUCE

Both Spark and Hadoop MapReduce are widely used in big data analytics, however, Spark has some major use cases over Hadoop[10]:

- Unlike Hadoop, Spark supports interactive data mining and data processing
- Spark outperforms Hadoop when it comes to iterative algorithms in machine learning as it keeps working sets in memory for efficient reuse
- Spark supports efficient stream processing which is one of the major advantages over Hadoop
- Spark is faster than MapReduce in execution

Though Spark has many advantages, Hadoop MapReduce can prove to be more efficient when it comes to batch processing for data with size greater than the available memory.

6 CONCLUSION

With the increasing volume of data, big data analytics is only going to become more critical for businesses decisions. Analyzing data at a huge scale presents many challenges and as we showed, Apache Spark can be very useful in overcoming these challenges. Over past few years, though Hadoop MapReduce has been one of the prime big data analytics framework, we showed how Spark has some major use cases over Hadoop. Though Apache Spark is a relatively young data project, it has already been adopted by a wide range of industries for big data analytics. We provided an introduction to Spark and discussed its architecture and the core components. As future work, we can discuss a case study and show how Spark processes big data in a more efficient manner than Hadoop MapReduce.

ACKNOWLEDGMENTS

We would like to thank Prof. Gregor von Laszewski and the teaching assistants for their helpful suggestions.

REFERENCES

[1] Apache Spark. [n. d.]. Cluster Mode Overview. ([n. d.]). http://spark.apache.org/docs/1.3.0/cluster-overview.html
[2] DeZyre. 2016. Apache Spark Ecosystem and Spark Components. (02 2016). https://www.dezyre.com/article/apache-spark-ecosystem-and-spark-components/219

[3] DeZyre. 2017. Apache Spark Architecture Explained in Detail. (03 2017). https://www.dezyre.com/article/apache-spark-architecture-explained-in-detail/338
[4] Jian Fu, Junwei Sun, and Kaiyuan Wang. 2016. SPARK–A Big Data Processing Platform for Machine Learning. In *Industrial Informatics-Computing Technology, Intelligent Technology, Industrial Information Integration (ICIICII), 2016 International Conference on*. IEEE, 48–51.
[5] V Srinivas Jonnalagadda, P Srikanth, Krishnamachari Thumati, and Sri Hari Nallamala. [n. d.]. A Review Study of Apache Spark in Big Data Processing. ([n. d.]).
[6] KDnuggets. 2016. Top Spark Ecosystem Projects. (03 2016). http://www.kdnuggets.com/2016/03/top-spark-ecosystem-projects.html
[7] Ovidiu-Cristian Marcu, Alexandru Costan, Gabriel Antoniu, and María S Pérez-Hernández. 2016. Spark versus flink: Understanding performance in big data analytics frameworks. In *Cluster Computing (CLUSTER), 2016 IEEE International Conference on*. IEEE, 433–442.
[8] Srini Penchikala. 2015. Big Data Processing with Apache Spark - Part 1. (01 2015). https://www.infoq.com/articles/apache-spark-introduction
[9] Abdul Ghaffar Shoro and Tariq Rahim Soomro. 2015. Big data analysis: Apache spark perspective. *Global Journal of Computer Science and Technology* 15, 1 (2015).
[10] Ankush Verma, Ashik Hussain Mansuri, and Neelesh Jain. 2016. Big data management processing with Hadoop MapReduce and spark technology: A comparison. In *Colossal Data Analysis and Networking (CDAN), Symposium on*. IEEE, 1–4.

7 BIBTEX ISSUES

Warning–empty year in img-arch

Warning–no journal in spark-a3

Warning–no number and no volume in spark-a3

Warning–page numbers missing in both pages and numpages fields in spark-a3

Warning–no journal in spark-a2

Warning–no number and no volume in spark-a2

Warning–page numbers missing in both pages and numpages fields in spark-a2

Warning–empty publisher in fu2016spark-p1

Warning–empty address in fu2016spark-p1

Warning–empty year in spark-j2

Warning–no journal in spark-j2

Warning–no number and no volume in spark-j2

Warning–page numbers missing in both pages and numpages fields in spark-j2

Warning–empty publisher in marcu2016spark-p2

Warning–empty address in marcu2016spark-p2

Warning–no journal in spark-a1

Warning–no number and no volume in spark-a1

Warning–page numbers missing in both pages and numpages fields in spark-a1

Warning–page numbers missing in both pages and numpages fields in spark-j1

Warning–empty publisher in verma2016big-p3

Warning–empty address in verma2016big-p3

(There were 21 warnings)

8 ISSUES

DONE:

Example of done item: Once you fix an item, change TODO to DONE

8.1 Uncaught Bibliography Errors

Bibtex labels cannot have any spaces, _ in it. We corrected them for you this time.

8.2 Citation Issues and Plagiarism

Put a space between the citation mark and the previous word

Are you sure you have quoted all the directly cited material

List of Figures

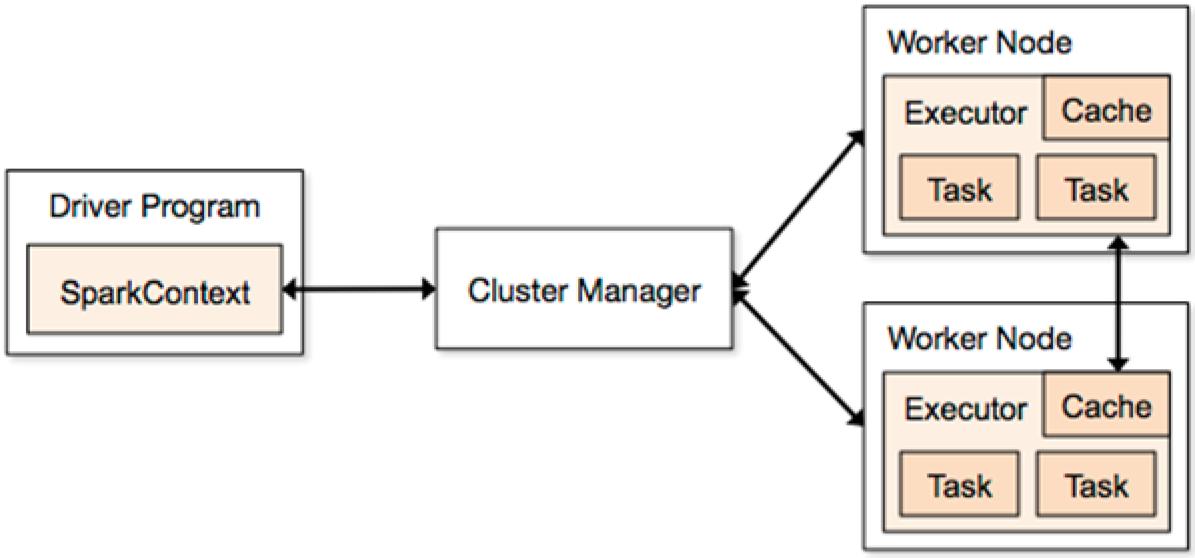

Figure 1: Spark Architecture[1]

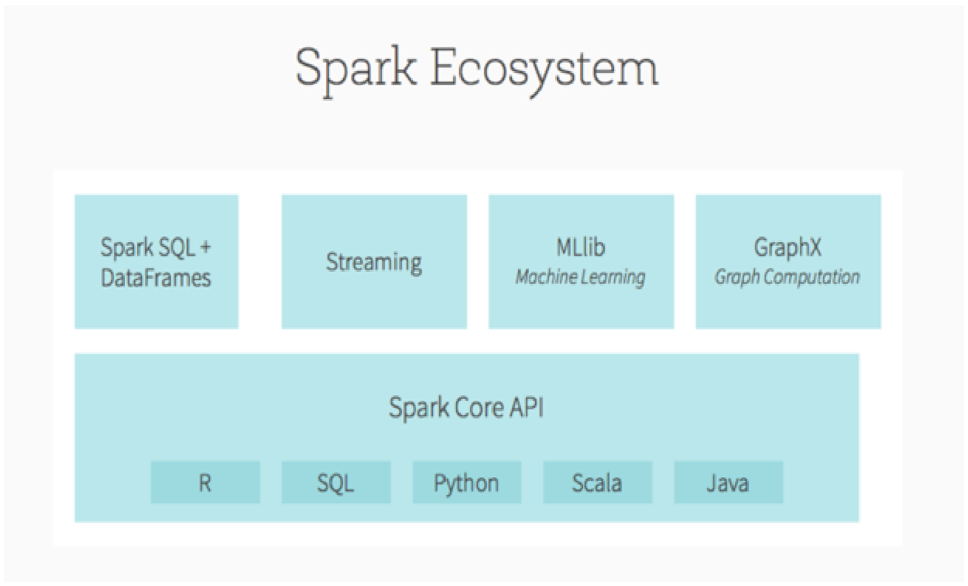

Figure 2: Spark Ecosystem[6]