Apache Spark Ecosystem for Big Data Analytics

G. Kapila
Southern Methodist University
Email: gkapila@smu.edu

H. H. Nguyen
Southern Methodist University
Email: hoangnguyen@smu.edu

H. Wang
Southern Methodist University
Email: wangmichael@smu.edu

***Abstract*—In this paper, we present an overview and tutorial of the Apache Spark ecosystem used for Big Data Analytics. Apache Spark is a unified computing engine and a set of libraries for parallel data processing on computer clusters. It has emerged as a leading contender for big data processing. Big Data analytics has been one of the most active research areas in recent times. Techniques of database dealing with large data are extension from traditional relational database. Apache Spark technique can deal with different large data processing types, for example, in real-time data analysis or graph processing. It is a fast and general engine for large-scale data processing and a open source for developers or data scientists interested in Big Data Analytics.**

***Index Terms*—Apache Spark, Big Data, Data Analysis, Hadoop, Machine Learning, Stream Processing, Spark's Architecture, Keys Components**

I. Introduction to Spark

Apache Spark, since its initial release in 2014, has become one of the most popular big data tools and considered as the next generation powerful unified analytics engine for large-scale distributed data processing and machine learning, with APIs in multiple programming languages and a wealth of built-in and third-party libraries. It is an open-source distributed general-purpose cluster-computing framework, making it a highly important for any developer or data scientist interested in big data.

Apache Spark is driving data analytics excitement. One of the biggest concerns of big data and analytics is security and privacy. Apache Spark performance processes data anonymization. Real time SPAM detection [21] and cyber crime forensics [10] are two of these security and privacy applications. Apache Spark can be used in different areas, for example, in scientific research for neural response to organisms activity [9] or in genomic data processing [1].

Apache Spark Ecosystem is fast for large-scale data processing and well integrated with Hadoop. Apache Spark is around 100 times faster than MapReduce using only RAM and 10 times faster if using the disk. In addition, it can run on top of YARN and can access HDFS [2] with great set of libraries. It has become a powerful alternative to Hadoop MapReduce with rich functionality features, for example, machine learning, graph computation and real-time stream processing.

A. History of Spark

Before Spark, many organizations and large enterprises have been using Hadoop and its framework based on MapReduce extensively to analyze their datasets [18]. Hadoop was the first open source system to tackle data-parallel processing on clusters of thousands of nodes. It was the dominant parallel programming engine for clusters of the big data world. The main concern is to maintain speed in processing large datasets then Spark was introduced for speeding up the Hadoop computational computing software process [14].

Going back to 2009, Apache Spark was first developed as the Spark research project and published the following year by Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker and Ion Stoica of UC Berkeley's AMP Lab (see [4], [20]). Zaharia and his collaborators had worked with multiple early Hadoop MapReduce users to understand its main benefits of tremendous potential clusters computing and drawbacks of challenging and inefficient to build large applications.

The first version of Spark only supported batch applications but it quickly became a new candidate for the big data world.The early version of Spark (before 1.0) largely defined composable APIs in terms of functional operations. In 2014, beginning with 1.0, a new API with structured data was added as Spark SQL.

Spark has its own cluster management. It is not a modified version of Hadoop own cluster management. Spark 2.0 was released in 2016 and the project continues to make regular releases with new features.

B. Big Data World

Big data analytics is at the core of transformation of organizations and large enterprises. It can help organizations and large enterprises to better understand the information contained in data to identify what information is the most important for their business decisions. Big data is getting bigger and bigger day by day. Ever increasing complexity and appetite for big data [7] require data processing to scale to multiple nodes [21]. Big data analytics is a process of collecting, organizing and analyzing large volumes of data. The aim of this process is to discover patterns and connections of useful information that might be invisible and can provide us a better insights about who created it.

Previously, the applications developed to handle large data ran faster as the speed of the processor increased. However, over last decade, as the processor speed plateaued out, the computing power increase has relied on processing data over multiple processors, and through development of distributed computing architecture. Hence traditional software applications have to be redesigned to handle this paradigm shift in computing framework. Apache Spark was built to adapt to

this new trend. It's high-speed cluster computing technology works with the system to distribute data across the cluster and process data in parallel.

II. Apache Spark Architecture

In this section, we walk through the core architecture of Apache Spark (see [4]–[6]). Many organizations and large enterprises have considered data processing as a particularly challenging area.

A. Basic Design and Concepts

Nowadays, a single computer or machine does not have enough power and resources to perform multi-tasks required to process large datasets. They have to use cluster computing to perform their tasks.

A cluster is a type of parallel or distributed computer system pools the resources of many machines together as a single integrated computing resources. Groups of independent computers are combined into a unified system through software and networking giving us the powerful ability to use all the cumulative resources.

Apache Spark helps us to manage and coordinate the execution of tasks on data across a cluster of computers. It has a master-slave architecture with a cluster manager and two daemons, as shown in Figure 1.

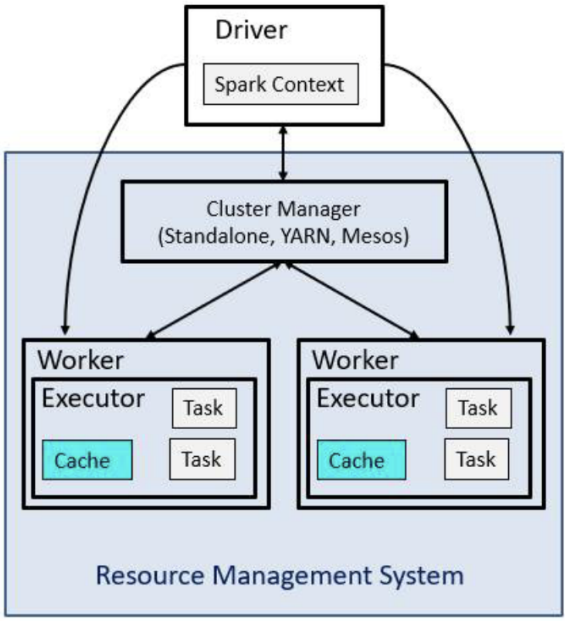

Figure 1. Apache Spark architecture

The cluster of machines is managed by a cluster manager. We submit Spark Applications to these cluster managers to perform and complete our tasks. The daemons are master and worker here. Every Spark application consists of a driver program that is responsible for running the user's main function and executing various parallel operations on the cluster.

B. Spark Applications

Spark Applications consist of a driver process that is responsible for running your *main()* function and a set of executor processes to execute various parallel operations on the cluster. The driver process is the heart of a Spark Application and the driver communicate in different ways for instance, with Spark through the Spark Context object, that is the entry point for Spark functionality. There are other ways that the driver can communicate with Cluster Manager for resource allocation to execute the tasks and with distributed workers called executors to execute tasks.

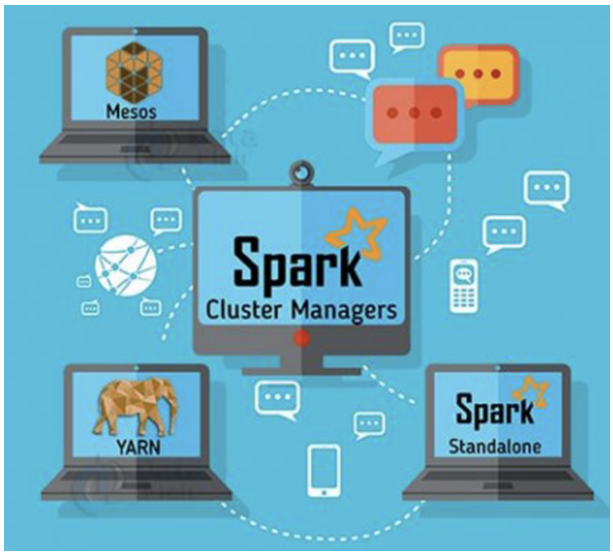

Figure 2. Apache Spark Cluster Managers

In Spark Applications, the Cluster Manager knows where the workers are located, how much memory they have and the number of CPU cores each one has. The essential responsibility of the Cluster Manager is to schedule and allocate resources across the host machines forming the cluster and to Spark Applications. There are three core Cluster Managers, YARN, Spark's standalone Cluster Manager and Mesos, that can be launched on-site or in the cloud.

The executors are responsible for actually executing the tasks and data processing in the given Spark jobs that the driver assigns them. They launch at the beginning of Spark Application and run for the lifetime of an application, read from and write to external sources, store intermediate results, offer resources (memory, CPU cores etc.)

C. Discussions

Spark's standalone Cluster Manager is one of three core Cluster Managers that Apache Spark supports. It is a simple cluster available as part of the Spark application and works on different environment (Linux, Windows or MAC OS). Its capacities for managing resources per application and for running alongside of existing Hadoop deployment and accessing Hadoop Distributed File System (HDFS) data make Spark's standalone Cluster Manager as the first choice of users.

Another Cluster Manager, Mesos, is a distributed systems kernel and has high availability for masters and slaves. Mesos has API's for Java, C++ and Python and can be run on Linux or Mac OS, support for Docker containers. This Cluster Manager can control resource per application and can run Spark jobs or Hadoop MapReduce. The third Cluster Manager, supported by Apache Spark that we will discuss now is YARN. It works on Linux and Windows. It also has high availability for masters and slaves.

III. Apache Spark Ecosystem

Apache Spark is written in Scala making it Spark's default language. Even though Spark is written in Scala, we can write code in Java. It provides high-level application programming interfaces (APIs) in Java, Scala, Python, and R. Python supports nearly all constructs that Scala supports. Apache Spark also supports a subset of the SQL standard that makes it easy for data analysts and non-programmers to take advantage of Spark big data powers. There are two common R libraries, SparkR and sparklyr. It runs on the Java Virtual Machine (JVM) either on a personal computer or on a cluster. The installation of Java is a need to run Spark.

Apache Spark continues to gain in popularity and use cases. Many new Spark Ecosystem projects have been introduced and used by Fortune 500 companies (Amazon, Ebay, Nasa Jpl, Yahoo, Uber, Slack, Netflix) to institutions (NASA, CERN, MIT, Harvard). In particular, in 2016, a new high-level streaming engine - Structure Streaming technology was made available that helps to solve massive-scale data challenges (see [11]–[16], [18]–[20]). This work serves as a tutorial on Apache Spark ecosystem, focusing on its key components, its applications and toolkit reviews. Table I provides a comparison between Spark and Hadoop MapReduce [5].

	Hadoop MapReduce	**Apache Spark**
General	Open source framework to process structured and unstructured big data	Open source big data processing framework
Data Source	Hadoop Distributed File System (HDFS)	In-memory and interface to external data source
Speed	Reads and writes from disk which slows down the data processing speed	100 times faster than Hadoop MapReduce when Spark runs in memory and 10 times faster when Spark runs on disk
Difficulty	Developers need to hand code each operation in Hadoop MapReduce	Spark provides high-level operators such as map, filter, and so on, which makes the developer's job easy
Real-time processing	Only supports batch processing	Spark supports both batch and real-time processing
Interactive mode	Does not provide an interactive shell	Spark provides an interactive shell to learn and explore data

Table I
SPARKS VS. HADOOP MAPREDUCE

Apache Stark supports different programming languages and libraries for diverse tasks ranging from SQL to streaming and machine learning. It can run anywhere from a laptop to a cluster of thousands of servers making it an easy system to start with and scale-up to big data processing or incredibly large scale. The Figure 3 illustrates all the components and libraries that Spark offers to end-users. Apache Spark has two fundamental sets of APIs, the low-level "unstructured" APIs and the higher-level structured APIs. Now we will walk through our description of Apache Spark into its key ecosystem features: Unified, Computing Engine and Libraries [4].

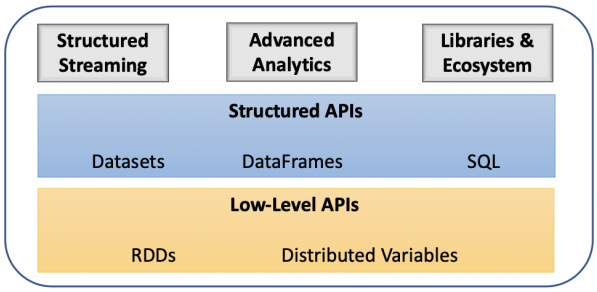

Figure 3. Components of Spark

A. Unified

The first component of Spark is unified. The design of Spark supports a wide range of data analytics tasks such that data loading, SQL queries, machine learning and streaming computation over the same computing engine and with a consistent set of APIs. Its unified platform is offered to write big data applications more easily and efficiently and for parallel data processing. By consistent and composable APIs, we can build an application out of smaller pieces or out of existing libraries with high performance by optimizing different libraries and functions composed together in a user program. To cover more workloads, Spark has continued to expand its built- in APIs .

B. Computing Engine

The second component of Spark is Computer Engine that is an infrastructure as a service that lets you run your large- scale computing workloads on virtual machines. We can use Spark with a wide variety of persistent storage systems, including cloud storage systems such that Azure Storage and Amazon S3, distributed file systems such as Apache Hadoop, key-value stores such as Apache Cassandra, and message buses such as Apache Kafka. The different key between Apache Spark and Apache Hadoop is the focus on computation of Spark, instead of the focus on a storage system and a computing system of Hadoop MapReduce.

C. Libraries

The final component of Spark is its Libraries to provide a unified APIs for common data analysis tasks. Some internal libraries are like Spark SQL for SQL and structured data, MLlib for machine learning, Spark Streaming for stream

processing and GraphX for graph analytics. There are also hundreds of open source external libraries that can be found at spark-packages.org, ranging from connectors for various storage systems to machine learning algorithms.

IV. Case Study

We now will go through some toolkits of Apache Spark.

A. Apache Spark's Toolkit Review

The focus so far in the tutorial has been on the structured APIs in Spark, specifically in context of Spark data frame. However, Spark's foundational aspects lie in low level API's like the Resilient Distributed Datasets, or RDDs, and vast suite of standard libraries for additional functionalities.

- Spark Submit: Spark makes it easy to turn our interactive program development exercise into production application through Spark Submit by allowing us to send application code to a cluster and launch and execute there. It contrasts this feature with Spark shell which is more useful for running small interactive jobs, and testing and debug the code. We can specify the resources that the application needs through command line switches. It provides support for all cluster managers including Standalone, Mesos, and YARN, and also applications in any of Spark's supported languages. Spark runs the job on our local machine, and has specifications for which class and which JAR we would like to run. By changing the master argument of Spark Submit, same application can be submitted to a cluster running Spark's standalone cluster manager, Mesos or YARN.

- Spark Datasets: It is a type-safe version of Spark's structured API used for writing statically typed code in Java and Scala. The dataset API is not available in Python and R, because those languages are dynamically typed but gives users the ability to assign a Java/Scala class to the records within a dataframe and manipulate it as a collection of typed objects, similar to a Java ArrayList or Scala Seq. The APIs available on Datasets are type-safe, meaning that one cannot accidentally view the objects in a dataset as being of another class than the class you put in initially. This makes datasets especially attractive for writing large applications, with which multiple software engineers must interact through well-defined interfaces.

- Structured Streaming: It is a high level API. The strength of this API is that one can use same operations that are run in batch mode, using structured API, and run it stream mode. This requires minimal code changes and hence lends to quick code development.

- Lower level API's: RDDs or Resilient Distributed Datasets (a key programming abstraction) is at the core of Spark. RDDs forms a simple extension to MapReduce model. At the core of Apache Spark is the notion of data abstraction as distributed collection of objects. Resilient Distributed Datasets (RDDs) are these data abstractions, allow us to write programs that transform these distributed datasets, to execute a sequence of commands independently on a cluster and enable data sharing across multiple computations then improving efficiency of iterative algorithms like K-means clustering, Page Rank, etc. RDDs constitute lower-level APIs available for manipulating raw or unprocessed unstructured data. RDDs are available in Scala and Python.

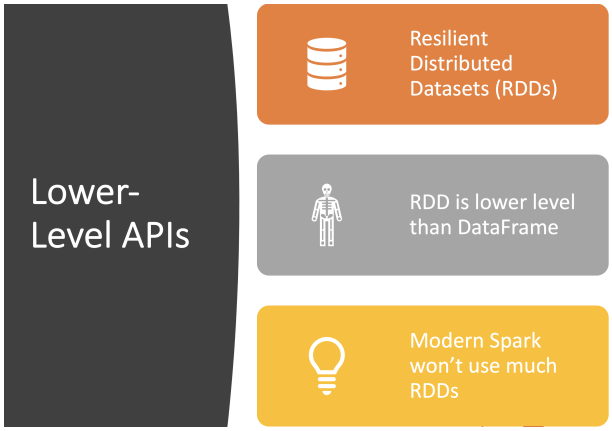

Figure 4. Lower Level APIs

- SparkR: It allows accessing Spark through an R program. SparkR package supports distributed dataframe implementation for Spark operations and distributed machine learning using MLlib. Its feature adds to the versatility of Spark to address big data analytic challenges.

B. Case Study Example for Machine Learning

In this section, we present a machine learning example using the Machine Learning and Advanced Analytics libraries using Python in Spark. Spark supports at scale machine learning analysis through MLlib, a built in library of machine learning algorithms. It has functions to handle data pre-processing, munging, training of models and last but not the least prediction. As is typical of Spark in terms of its versatility, predictions from machine learning models can be easily applied to streaming data through structured streaming. Spark provides API to handle different ML tasks like classification, regression, clustering and deep learning. We now look features of a typical machine learning application flow in Spark that uses K-means clustering algorithm.

Typically, the first step is to transform the data to numeric format, so that it can be input to machine learning algorithm in MLlib.

```
from pyspark.sql.functions
import date_format, col
preppedDataFrame = staticDataFrame
  .na.fill(0)
  .withColumn("day_of_week",
  date_format(col("InvoiceDate"), "EEEE"))
   .coalesce(5)
```

We can also create test and train data set easily, for example, in this case study, it is done with respect to invoice date being less than 1st July 2011 or not.

```
trainDataFrame
= preppedDataFrame
    .where("InvoiceDate < '2011-07-01'")
        testDataFrame
 = preppedDataFrame
    .where("InvoiceDate >= '2011-07-01'")
```

By using *transformedTraining.cache()* command, one can perform in memory computations that lead to speed improvements.

```
transformedTraining.cache()
```

Model training is typically a two-step process. First, the model is initialized

```
from pyspark.ml.clustering import KMeans
kmeans = KMeans().setK(20).setSeed(1L)
```

and then it's trained.

```
kmModel = kmeans.fit(transformedTraining)
```

Finally, the model developed is validated and tested on test data set, that can be performed using a simple command.

```
transformedTest =
    fittedPipeline.transform(testDataFrame)
kmModel.computeCost(transformedTest)
```

C. A SparkR exemple

The following code is an example of SparkR. Here SparkR library is loaded in R environment in order to use Spark in R directly.

```
// Load SparkR to use Spark from R
library(SparkR)
sparkDF <-
read.df("/data/flight-data/csv/2015-summary.csv",
source = "csv",  header = "true", inferSchema
    = "true")
take(sparkDF, 5)
collect(orderBy(sparkDF, "count"), 20)
```

D. Tableau and Apache Spark - An integration

The integration of Tableau with Spark gives us a huge value in big data world. It provides a fast, versatile data processing engine at their fingertips and also new capabilities to the Spark community. By Tableau environment, we can visually analyze data without writing a single line of Spark SQL code. Connecting Apache Spark to Tableau is one of the best approaches for processing and visualizing big data. By integrating Tableau with Spark, it requires a driver to talk to the database. Apache Spark needs to be installed either on our machine or on an accessible cluster.We can start Tableau Desktop that supports SparkSQL connector, and under Connect, we select Spark SQL. Here we used the data source from TEXAS Health and Human Services website at the following website https://dshs.texas.gov/coronavirus/AdditionalData.aspx and we load it into Tableau to create a Tableau Dashboard (Figure 5).

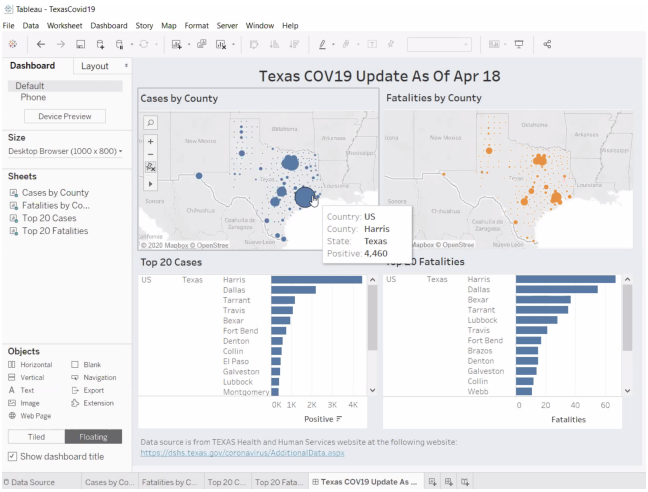

Figure 5. Tableau - Texas COV19 - Updated 04-18-2020

V. Conclusions

Apache Spark is a next-generation cluster platform with fast and efficient stream processing capacities. It supports more computation types than Hadoop MapReduce. In this tutorial, we walked through how Apache Spark can help with big data processing and analytics with its standard APIs. Key aspects of Apache Spark architecture and Ecosystem were reviewed, and a machine learning case study was highlighted. With the massive explosion of Big Data and the exponentially increasing speed of computational power, Apache Spark has moved from a talking point in the boardroom discussions to enterprise deployments. It can help handle the complexity big data with ease. Due to its open source nature, active users and developers all over the world, it has promise to grow even further, and become mainstay for big data processing in future.

VI. Acknowledgement

We would like to express our sincere gratitude to Dr. Daniel Engels (SMU) for fruitful discussions during the preparation of this paper and the Spring semester 2020.

References

[1] A. Bahmani, A.B. Sibley, M. Parsian, K. Owzar, F. Mueller, "Sparkscore: leveraging apache spark for distributed genomic inference", *IEEE International Parallel and Distributed Processing Symposium Workshops*, pp. 435–442, 2016.

[2] CloudxLab, "Apache Spark ecosystem walkthrough", *CloudxLab* [Online]. Available: https://cloudxlab.com/assessment/displayslide/126/apache-spark-ecosystem-walkthrough [Accessed: 04/15/2020].

[3] ClearPeaks, "Big Data Ecosystem - Spark and Tableau", *ClearPeaks* [Online]. Available: https://www.clearpeaks.com/big-data-ecosystem-spark-and-tableau/ [Accessed: 04/17/2020].

[4] B. Chambers, M. Zaharia, "Spark: The Definitive Guide - Big Data Processing Made Simple", *O'Reilly*, 2018.

[5] S. Chellappan, D. Ganesan, "Practical Apache Spark: Using the Scala API", *Apress*, 2018.

[6] DataFlair Team, "How Apache Spark Works - Run-time Spark Architecture", *DataFlair Blog* , 11/21/2018 [Online]. Available: https://data-flair.training/blogs/how-apache-spark-works/ [Accessed: 02/18/2020].

[7] M. Deer, "Seven Critical Predictions for Big Data in 2020", *Towards Data Science Blog* , 2020 [Online]. Available: https://towardsdatascience.com/seven-critical-predictions-for-big-data-in-2020-e9a867620 [Accessed: 01/26/2020].

[8] J. Feng, "Tableau and Spark SQL: Big data just got even more supercharged", *Tableau* , 2020 [Online]. Available:https://www.tableau.com/about/blog/2014/10/tableau-spark-sql-big-data-just-got-even-more-supercharged-33799 [Accessed: 04/17/2020].

[9] J. Freeman, N. Vladimirov, T. Kawashima, Y. Mu, N. J. Sofroniew, D. V. Bennett, J. Rosen, C-T. Yang, L.L. Looger, M.B. Ahrens, "Mapping brain activity at scale with cluster computing", *Nature Methods* 11, 9, pp. 941–950, 2014.

[10] E. E. Hemdan, D. H. Manjaiah, "Spark-based log data analysis for reconstruction of cybercrime events in cloud environment", *International Conference on Circuit ,Power and Computing Technologies (ICCPCT)*, pp. 1–8, 2017.

[11] F. Kane, "Apache Spark with Scala - Hands On with Big Data!", *Udemy*, 01/2020 [Online]. Available: https://towardsdatascience.com/a-beginners-guide-to-apache-spark-ff301cb4cd92 [Accessed: 01/26/2020].

[12] D. Kovachev, "A Beginner's Guide to Apache Spark", *Towards Data Science Blog*, 02/24/2019 [Online]. Available: https://towardsdatascience.com/a-beginners-guide-to-apache-spark-ff301cb4cd92 [Accessed: 01/26/2020].

[13] D. Lee, J, Damji, "Apache Spark Key Terms", *databricks*, 06/22/2016 [Online]. Available: https://databricks.com/blog/2016/06/22/apache-spark-key-terms-explained.html [Accessed: 02/18/2020].

[14] S. Salloum, R. Dautov, X. Chen, P. X. Peng, J. Z. Huang, "Big data analytics on Apache Spark", *Int J Data Sci Anal*, vol. 1, pp. 145–164, 2016.

[15] Sandy, "Spark Tutorial for Beginners", *YouTube*, 03/13/2016 [Online]. Available: https://www.youtube.com/watch?v=-voECgAmpTg [Accessed: 26/01/2020].

[16] J. G. Shanahan, L. Dai, "Large scale distributed data science using apache spark", *Proc. 21th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min. (KDD)*, pp. 2323–2324, Aug 2015.

[17] Tableau Desktop and Web Authoring Help, "Spark SQL", *Tableau* [Online]. Available: https://help.tableau.com/current/pro/desktop/en-us/examples_sparksql.htm [Accessed: 04/17/2020].

[18] Tutorialpoint, "Learn Apache Spark - Simply Easy Learning", *Tutorial point* [Online]. Available: https://www.tutorialspoint.com/apache_spark/ [Accessed: 04/14/2020].

[19] M. Zaharia, "An Architecture for Fast and General Data Processing on Large Clusters," Ph.D. dissertation, *U.C. Berkeley*, 2013.

[20] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoica, "Spark: Cluster Computing with Working Sets", *University of California, Berkeley*, 2010 [Online]. Available: usenix.org/legacy/event/hotcloud10/tech/full_papers/Zaharia.pdf [Accessed: 17/02/2020].

[21] M. Zaharia, et. al., "Apache Spark: A Unified Engine for Big Data Processing", *Commun. ACM*, vol. 59, no. 11, pp. 56–65, 2016.