Apache Spark for Data Science

H. H. Nguyen, T. Abera, J. Coleman, G. Gonzales, S. McWhirter
Southern Methodist University
Email: { hoangnguyen, tabera, jasminecoleman, gngonzales, smcwhirter}@smu.edu

***Abstract*—This research covers the Apache Spark ecosystem, where it fits into the cloud framework, its functions underneath the hood, and machine learning examples to illustrate how the ecosystem operates. Apache Spark is a distributed computing framework that is ideal for handling big data. Spark is an ideal solution to explore, as the ability to handle big data can be a burdensome administrative task yet very necessary. This research will discuss where Spark fits into the framework of IaaS, PaaS, and SaaS, as well as its benefits and drawbacks compared to proprietary solutions. This research also provides a brief demonstration on how computing on Spark actually takes place to further illustrate its functions.**

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

I. Introduction to Spark

Modern datasets are rapidly growing in size and complexity; and there is a pressing need to develop solutions to harness this wealth of data using statistical methods. Every organization or company, be it in the healthcare, manufacturing, automotive, or software industry, needs to manage and analyze the large data volume it generates. The ability to efficiently and effectively harness big data leads to faster and more efficient operations. The growing need to manage increasingly massive data collections has led to significant interest in developing such big data tools. Research is being conducted in the areas of Big Data Infrastructure (cloud computing software systems) and Big Data Management (search and mining, algorithmic systems, data acquisition, integration and cleaning, computational modeling, graph mining, distributed and peer-to-peer search etc.).

Apache Spark has emerged as a widely used open-source engine. Spark is a fault-tolerant and general-purpose cluster computing system providing APIs in Java, Scala, Python, and R, along with an optimized engine that supports general execution graphs. Moreover, Spark is efficient at iterative computations and is thus well-suited for the development of large-scale machine learning applications.

Since Apache Spark's release almost seven years ago, it quickly became one of the most utilized tools for big data processing. Spark's ability to work with popular languages and multiple third-party libraries make it an easy choice as an effective tool. It's open-source framework provides additional benefit as incorporation of community feedback is more agile, maintaining alignment between Spark's features and the needs of the users.

The broad range of applications that Spark offers in its cluster-computing framework are highlighted across many different industries and use cases. Apache Spark is used everywhere from genomic analysis [4] and neural response to organism activity [12] to SPAM email detection [29] and cyber crime forensics [14].

Apache Spark also works well with other popular ecosystems. Spark is well-integrated with Hadoop yet runs approximately 100 times faster than Hadoop's processing technique MapReduce, making Spark an ideal alternative to MapReduce when speed and real-time processing are required [5].

II. History of Spark

Prior to Spark taking the stage, one of the primary options available for big data analysis was Hadoop's MapReduce [24]. Hadoop was the primary option as it was the first open-source approach to parallel processing on clusters containing a massive number of nodes. However, the speed of MapReduce became an issue, which was when Spark was introduced to accelerate Hadoop's computation process [20].

Spark was first developed by Monsharaf Chowdhury, Matei Zaharia, Michael J. Franklin, Ion Stoica, and Scott Shenkler of the University of California Berkley's AMP lab in 2009 (see [7], [28]). The team worked with a number of MapReduce users to understand the current benefits of cluster computing as well as the inefficiencies of MapReduce that could be improved.

Despite only supporting batch applications in the first version of Spark, it quickly caught the attention of the big data world. The idea was to build a cluster management framework, which could support different kinds of cluster computing systems. Many of the ideas behind the system were presented in various research papers over the years. After being released, Spark grew into a broad developer community, and moved to the Apache Software Foundation in 2013.

Beginning with Spark 1.0 in 2014, the team added a structured data API and Spark SQL was born. The project released version 2.0 of Spark in 2016 and version 3.1.1 in 2021. New updates and features have been regularly produced since.

III. Big Data World

In the current business environment, big data is nearly unavoidable. The ability to capture, store and analyze big data can help organizations get the most out of the data being produced by its operations. This is vital for gaining a competitive advantage and ensuring that the business remains relevant. Big data today is not what big data was ten years ago. The volume, variety, and velocity of big data is growing constantly, and organizations stand to gain if they can leverage

appropriately. The unsatiable appetite for big data, and its growing complexities, require efficient scaling in order for that data to provide utility (see [10], [29]).

Historically, the ability for big data applications to handle large datasets increased with the speed of computing processors. However, more recently the increase of processing speed has plateaued, leading to the development of distributed computing frameworks. Apache Spark was developed to help keep the ability to process and analyze big data rising upward despite the plateau of processing power. Spark is able to deliver incredible speeds with its cluster computing technology that allows for parallel processing.

IV. Cloud Computing Ecosystem and Apache Spark Architecture

A. Cloud Computing Framework

The general cloud computing framework is based around three central offerings: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Infrastructure as a Service provides users with networking, storage, servers, and virtualization to perform computations and store data, such as Amazon's S3 storage or EC2 virtualization instances. Platform as a Service providers offer additional capabilities such as an operating system and middleware. Microsoft Azure Services Platform, Amazon's Relational Database Services, and Google App Engine are examples of PaaS as they provide additional support, but ultimately leave the application assembly and customization up to the user. Software as a Service, such as Salesforce.com, provide nearly everything to the user aside from certain customizations [18].

B. Structure

Apache Spark fits into this framework as a Platform as a Service, leaving application development and customization in the hands of the developer. The underlying composition of Apache Spark is comprised of a driver node, a cluster manager, and worker nodes.

The driver node is the main interface between the developer and the Apache Spark system. This node's main function is to decompose the developer's application into smaller units known as tasks. Once the drive program has decomposed the application, these tasks are sent to the cluster manager. The cluster manager's role is to balance the various tasks to the worker nodes. These worker nodes perform the computing tasks and cache the results, providing feedback to the driver program.

C. Apache Spark Architecture

Businesses and organizations have historically considered big data processing a challenge.

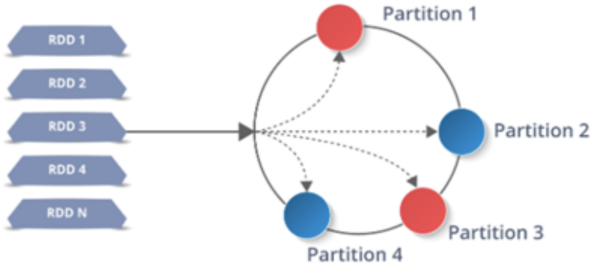

Figure 1. Resilient Distributed Datasets (RDDs)

However, the architecture of Apache Spark democratize the ability to process and analyze data at scale (see [7]–[9]). A key component of Apache Spark is the abstraction of Resilient Distributed Datasets (RDDs). RDDs are a collection of immutable and partitioned records. A key benefit of RDDs is that they do not need to be constantly materialized. The data on the RDD's derivation from other datasets provides sufficient information to compute its partitions. Another important feature is that developers are able to control the RDD's persistence and partitioning. This allows developers to specify the RDD's storage method and how each RDD is partitioned, making it ideal for dataset joins during the development cycle. RDDs are a simple extension of the MapReduce model, tying into a core component of Spark with data abstraction and distributed object collections. RDDs are immutable collections that are divided into partitions. RDDs allow spark to produce more efficient MapReduce computations.

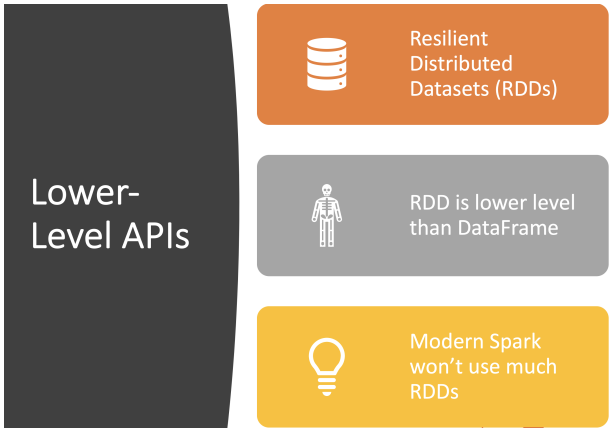

Figure 2. Lower Level APIs

There have been enormous strides in the power and abilities of single computers, but a single computer still lacks the ability and resources to execute the required tasks for big datasets. Cluster computing allows our current technology to capture these enormous datasets.

Put simply, a cluster is a group of machines working in parallel as a unified resource. Single computers or servers are combined through various networking procedures to provide the ability to harness the aggregate compute power.

Apache Spark employs this cluster framework and manages the various tasks across the individual servers. Spark uses

a master-slave architecture that contains two daemons and a cluster manager which is outlined in Figure 3.

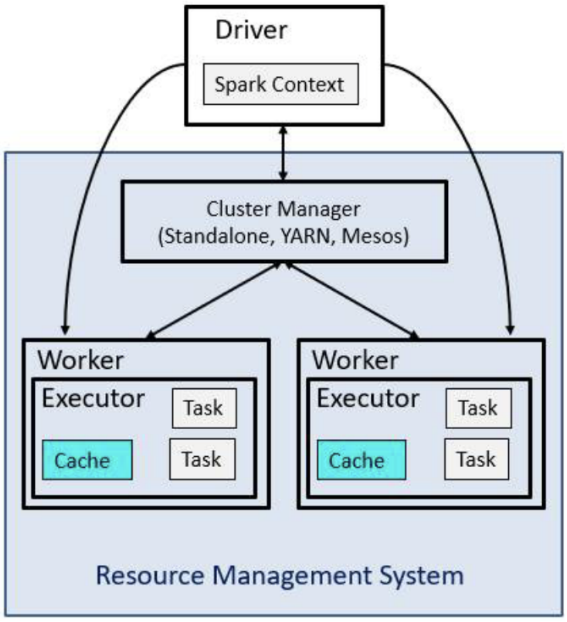

Figure 3. Apache Spark architecture

The individual machines that comprise the cluster are managed by the cluster manager after applications are submitted and ultimately carried out. The daemons, or background processes controlled by the manager, are the master and worker. Each individual Spark application contains a driver program, which is responsible for running the main function and executing the required operations in parallel within the cluster.

The driver program (driver) is at the foundation of a Spark application, and the driver can communicate with the cluster in multiple ways. A main method of communication is for the driver to interact with the Spark Context object. The driver can also communicate directly with the cluster manager to manage resource allocation in order to execute required tasks via distributed workers, known as ”executors”.

Within Spark applications, the cluster manager is cognizant of where the workers are located as well as the memory and CPU allocation of each worker. The cluster manager's core responsibility is to allocate resources across the various machines that comprise the cluster. As illustrated in Figure 3, the three main cluster managers are YARN, Mesos, and Spark's standalone cluster manger. Cluster managers can be launched on-site or virtually in the cloud.

The responsibility of the executors, the distributed workers, is to carry out the computations and processes that were assigned to them by the driver. These executors are launched at the beginning of a Spark application and remain online throughout the life of the application. The executors are able to read and write to external sources as well as store intermediate results.

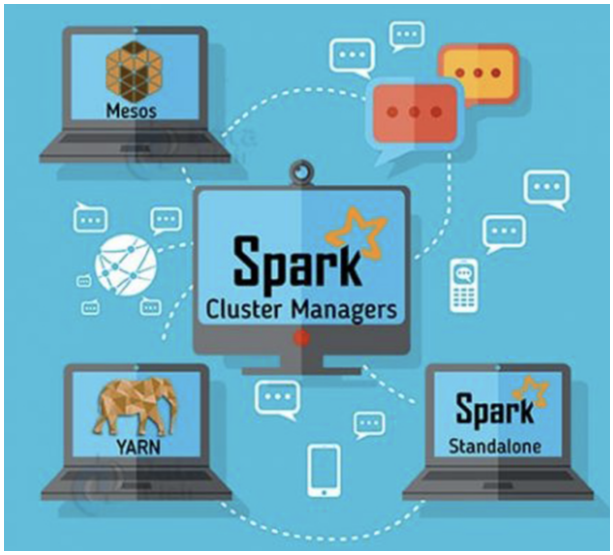

Figure 4. Apache Spark Cluster Managers

D. Directed Acyclic Graph (DAG)

The directed Acyclic Graph (DAG) component is used by the cluster manager for scheduling various jobs. Every time the developer executes an action on a RDD, the cluster manager ”examines the RDD's lineage graph to construct a DAG of stages to execute” (see [27]).

E. Apache Spark's Toolkit Review

Spark makes it easy to transform any interactive program into a production application with Spark shell and Spark Submit. Spark shell allows users to test and debug code, as well as specify resources that the application will need to run. Spark Submit allows users to easily send application code to a cluster for execution. By altering the master argument within Spark Submit, the user can specify YARN, Mesos, or a standalone cluster manager.

Spark also provides a type-save version of its structured API used for statistically typed code with Spark Datasets. Spark Datasets are not available in R or Python, due to these languages being dynamic. However, users can use Java or Scala to assign classes to records that are similar to a Java ArrayList or Scala Seq. It is important to note that Spark Datasets are type-safe, which means that users cannot view objects within the Spark Dataset a class other than what was initially defined. This feature is useful for large projects that employ multiple software engineers.

Spark also allows users to easily change from batch mode to stream-processing with Structured Streaming. This API requires minimal code changes, leading to little downtime if going from batch to stream processing.

V. Apache Spark Ecosystem

Apache Spark's default language is Scala, as Spark was constructed using the language. However, Spark also allows for code in Java. Spark also provides various high-level application programming interfaces (APIs) in Python, R, Java,

and Scala. Due to the popularity of Python in data analytics and machine learning, nearly all of the constructs that Scala supports are also supported by Python. Spark also makes it easy for analysts and those not sufficient in programming by offering a subset of the standard SQL language. However, a key dependency is Java since Spark runs on the Java Virtual Machine (JVM) whether it is on a personal computer or on a cluster. The ability to incorporate these other tools provides developers a vast number of use cases and methods. The flexibility that Spark provides has led to it being one of the most popular distributed computing systems. FINRA, Financial Industry Regulatory Authority, uses Spark to evaluate billions of real-time marketing events by moving their data processing from on-premises to the cloud via Apache Spark and Amazon Web Services (AWS). Another example of an ideal application of Spark is its use by Zillow, the poplar online real estate company. Zillow uses Spark to process enormous amounts of data to produce real-time home price estimates for customers [1].

The adoption of Apache Spark continues to rise, leading to many new projects in the Apache Spark ecosystem. Many of these new projects are used by Fortune 500 companies and prestigious institutions such as NASA, CERN, MIT, Harvard, Amazon, Uber, and Netflix. In 2016, Structure Streaming technology was made available which allowed users to handle data challenges on a truly massive scale (see [15]–[17], [20]–[22], [24], [27], [28]).

Apache Spark is able to execute on nearly any data-related task, ranging from real-time stream processing and machine learning to basic SQL queries. Spark's scalability allows users to run Spark from a single local machine, but can easily scale up to thousands of servers as the user's data consumption and needs change.

The scalability is a primary component of Spark's utility, but its ecosystem also provides value. The three main ecosystem features are a unified platform, Spark's computing engine, and various libraries available to users (see [7]).

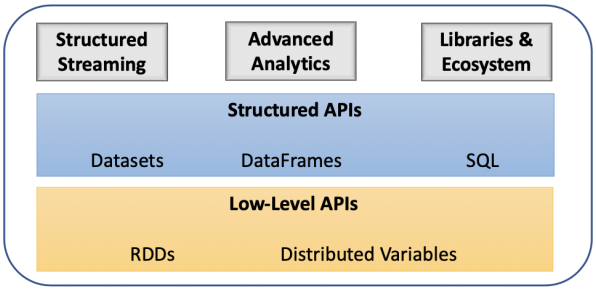

Figure 5. Components of Spark

A. Unified Platform

Apache Spark provides a unified platform that incorporates a wide range of tools that can help users tackle nearly any analytic task. Spark can manage a majority of the data analytics pipeline starting with data loading and ETL via SQL queries to real-time stream processing and machine learning all with a manageable and consistent set of APIs. These consistent APIs allow Spark users to compose big data applications more efficiently by allowing users to combine smaller custom pieces, or by using the various libraries available. To provide even more benefit, Spark's unified platform has continued to expand its constructed-in APIs.

B. Spark's Computing Engine

A key differentiator of Apache Spark is its focus on computing power instead of storage being its main concern. Spark helps bring the compute to the storage, not the storage to the compute, which speeds up the process considerably. Spark's computing engine works with a diverse array of cloud storage options including AWS, Microsoft Azure, Google Cloud, Hadoop, Cassandra, and Apache Kafka.

C. Libraries

The libraries that Spark utilizes allow for efficient execution of data-related tasks in common languages and procedures. Spark SQL is an internal library that allows for the handling of structured data, GraphX for graph analytics, and Spark Streaming for real-time stream processing. MLlib is Spark's library for almost any machine learning task. In addition to these native libraries, there are hundreds of external libraries ranging from data storage to machine learning that can be found on spark-packages.org.

VI. Building Machine Learning Algorithms on Apache Spark

Apache Spark has transformed the world of Big Data. It is the most active big data tool reshaping the big data market. This open-source distributed computing platform offers more powerful advantages than any other proprietary solutions. The diverse advantages of Apache Spark make it an attractive big data framework.

Apache Spark has huge potential to contribute to the Big Data-related business in the industry. Some of the common benefits of Apache Spark are its speed, multilingualism, advanced analytics, and open-source community.

A. Speed

Although Apache Spark and Hadoop serve different purposes, Hadoop's MapReduce is often compared to Apache Spark to illustrate Spark's potential benefits for big data computing. One area where Spark shines is its blazing speed, as Spark brings the compute to the data source rather than vice versa. Compared to Hadoop's MapReduce, Spark is 100 times faster when run in memory, and 10 times faster when run on hard disk. Given that these are both big data platforms, that illustration of speed is quite significant. In addition to speed, Spark can also support real-time processing in addition to batch processing, whereas MapReduce can only support the latter [8]. Table I provides a comparison between Spark and Hadoop MapReduce (see [8], [13]).

	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

B. Multilingualism and MLlib

In addition to its speed, Spark also has the advantage of being easily accessible to a diverse range of users. MapReduce requires each process be coded individually. Spark, on the other hand, operates by providing high-level functions such as "map," which can be used with a variety of programming language API's such as Java, Scala, Python, and R (see [8]).

This accessibility via high-level API's allows for broader access by the Data Science community, which is one reason employers have been adding Apache Spark to job requirements. Spark's machine learning library MLlib provides Java, Scala, Python, and R programmers the ability to utilize a vast array of machine learning algorithms and workflow utilities on almost any data source. MLlib is able to run machine learning pipelines against all major data sources such as Hadoop, Mesos, Kubernetes, Cassandra, and many more. Spark is also able to run on its own standalone cluster, on the local machine, Hadoop YARN, Kubernetes, or an AWS EC2 instance. The flexibility of host and data source options provides a vast number of use cases [19].

C. Open-source Community

The power and flexibility of Spark attracts a large number of contributors to its open-source concept. Spark currently lists 1,628 contributors on its GitHub for its MLlib library alone. While those are just the contributors to the actual code, there are likely countless others that are part of message boards and other feedback systems that provide constant feedback to ensure Spark remains powerful and easy to use [19].

VII. APACHE SPARK EXAMPLES

A. Python Machine Learning and Apache Spark

Here we will cover an example using machine learning and advanced analytics libraries offered by Spark in Python. MLlib is a Spark-native library that allows for running machine learning algorithms at scale. MLlib also incorporates functions that allow for the handling of data preprocessing, data munging, machine learning model testing, and model application. Spark also provides APIs for various types of machine learning problems like regression, classification, clustering, and deep learning. The example below will focus on using k-means clustering with Apache Spark.

When clustering, the first step is to transform any categorical data into numeric, so that it can be processed by the algorithm.

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

Once our data is transformed, MLlib also allows us to easily split our dataset up into a training set and a testing set based on the use case. For this example, the data has a temporal component, so we are splitting the data based on the July 1^{st}, 2011 invoice date.

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

To fully utilize the speed that Apache Spark offers, we use the *.cache()* command on our *transformedTraining* data to load the data into memory.

```
transformedTraining.cache()
```

We then begin the two-step process of our clustering model, beginning with initializing the KMeans algorithm

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

and then training it.

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

Lastly, the trained model is run on the testing set for validation. This step is completed using two simple commands.

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

B. A SparkR exemple

The ease of using Spark in Python can also be realized in R. There are two main ways to use Spark in R: SparkR and SparklyR. This example focuses on the former.

Within the R environment, we load the SparkR library in order to use Spark.

```
// 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)
```

C. Tableau Integration with and Apache Spark

Apache Spark more than covers the data loading, preprocessing, and analysis phases of the data pipeline. A key component that realizes the work for stakeholders is the visualization of the data and the analysis. To accomplish this, Spark integrates with Tableau – a leading data visualization platform. In order to integrate Spark with Tableau, it is necessary that Spark is on an accessible cluster, or installed locally on the machine. Tableau Desktop (the component of Tableau in which one builds the visualizations) provides an out-of-the-box Spark SQL connector.

D. Deep Learning and Apache Spark

Deep learning has gained its popularity in both academic and industrial fields. It is a subset of Machine Learning based on artificial neural networks and representation learning by mimicking the way of human brain works to learn from large scale data for different machine learning types such as supervised, semi-supervised or unsupervised learning. By using Apache Spark, many previously difficult deep learning problems with unstructured data can be now solved. There are three main ways to use deep learning in Spark. The first way, and the simplest, is to use pre-trained machine learning models for a large scale dataset in parallel using Spark. The next level to solve a deep learning model is to use an existing model such as a featurizer and the most complex level is to use Spark to train a new deep learning model from scratch [7]. Apache Spark is one of the easiest ways to solve and process computationally heavy processing Deep Learning models. There are many Deep Learning libraries, for instance, MLlib Neural Network Support, TensorFrames, BigDL, TensorFlowOnSpark, DeepLearning4J and Deep Learning Pipelines open source package from Databricks.

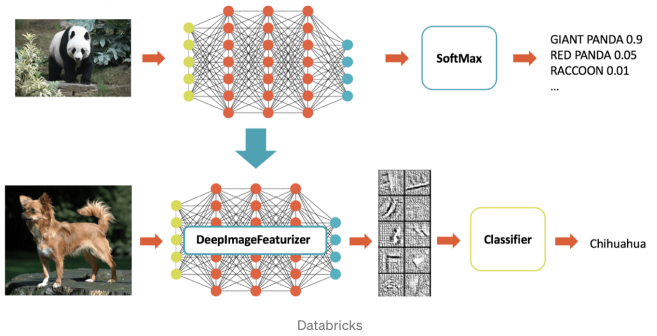

Figure 6. Transfer learning

One example of using Deep Learning Pipelines is to perform transfer learning on an image. It supports running pre-trained models with Apache Spark. By using DeepImageFeaturizer transformer, the pre-trained model ”peels off the last layer of a pre-trained neural network and uses the output from all the previous layers as features for the logistic regression algorithm”, for instance (see [25], [26]).

E. AWS EMR and Apache Spark

AWS is the most popular on-demand cloud computing platform having various available services that are well documented. Amazon EMR, a big data platform, is the ”industry-leading cloud big data platform for processing vast amounts of data using open source tools such as Apache Spark, Apache Hive, Apache HBase, Apache Flink, Apache Hudi, and Presto” [2]. We can install Apache Spark on EMR cluster along with other Hadoop applications and the data stored in Amazon S3 can be accessed directly by the EMR file system to perform a large scale machine learning model.

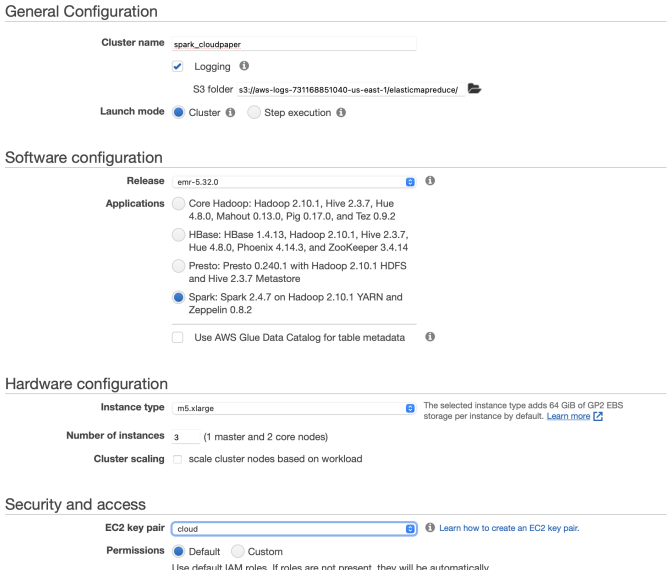

Figure 7. EMR cluster for Spark

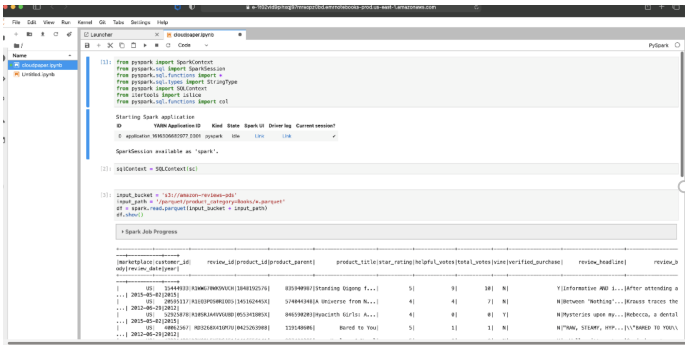

Figure 8. PySpark notebook in AWS EMR

Figure 7 refers to how to create an EMR cluster for a Spark application in AWS. We can also create a notebook in Amazon

EMR and use the PySpark kernel to run a Spark job to perform a Machine Learning model in AWS (see Figure 8 for the reference).

AWS EMR supports auto scaling to add more nodes to the cluster to give us a perfect and dynamic solution when a large Spark job is submitted or the clusters are connected by more developers. In addition to Amazon EMR, Google Cloud, Databricks and Cloudera also offer similar services for the Apache Spark community.

VIII. Conclusions

Apache Spark's user-oriented development, along with its open-source community, have made Spark essential for efficient processing of big data. Its unique architecture and ecosystem allow for an effective alternative to Hadoop's MapReduce not only for batch processing, but also real-time stream processing. Spark's high-level APIs give programmers from multiple languages, such as Python and R, have access to this powerful tool. Apache Spark has gone from a discussion point behind the scenes to a must-have capability for businesses and organizations that deal with big data in real-time. Given Spark's popularity and open-source community, it will be interesting to see how Apache Spark continues to define real-time big data processing.

IX. Acknowledgement

We would like to express our sincere gratitude to Dr. Sohail Rafiqi (SMU) for fruitful discussions during the preparation of this paper.

References

[1] AWS, "Introduction to Apache Spark", *AWS* [Online]. Available: https://aws.amazon.com/big-data/what-is-spark/ [Accessed: 03/21/2021].

[2] AWS, "Apache Spark", *AWS* [Online]. Available: https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-spark.html [Accessed: 03/21/2021].

[3] AWS, "Large-Scale Machine Learning with Spark on Amazon EMR", *AWS* [Online]. Available: https://aws.amazon.com/blogs/big-data/large-scale-machine-learning-with-spark-on-amazon-emr/[Accessed: 03/21/2021].

[4] 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.

[5] CloudxLab, "Apache Spark ecosystem walkthrough", *CloudxLab* [Online]. Available: https://cloudxlab.com/assessment/displayslide/126/apache-spark-ecosystem-walkthrough [Accessed: 03/15/2021].

[6] ClearPeaks, "Big Data Ecosystem - Spark and Tableau", *ClearPeaks* [Online]. Available: https://www.clearpeaks.com/big-data-ecosystem-spark-and-tableau/ [Accessed: 03/17/2021].

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

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

[9] 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: 02/18/2021].

[10] 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: 01/26/2021].

[11] 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: 03/17/2021].

[12] 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.

[13] G. Kapila, H. H. Nguyen, H.Wang, "Apache Spark Ecosystem for Big Data Analytics", *MSDS SMU*, 2020.

[14] 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.

[15] 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: 01/26/2021].

[16] 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: 01/26/2021].

[17] 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: 02/18/2021].

[18] S. R. Marston, Z. Li, S. Bandyopadhyay, A. Ghalsasi, J. Zhang, "Cloud computing: The business perspective", *SSRN Electronic Journal*, 2009.

[19] MLlib, [https://spark.apache.org/mllib/].

[20] 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.

[21] Sandy, "Spark Tutorial for Beginners", *YouTube*, 03/13/2016 [Online]. Available: https://www.youtube.com/watch?v=-voECgAmpTg [Accessed: 01/26/2021].

[22] 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.

[23] Tableau Desktop and Web Authoring Help, "Spark SQL", *Tableau* [Online]. Available: https://help.tableau.com/current/pro/desktop/en-us/examples_sparksql.htm [Accessed: 03/17/2021].

[24] Tutorialpoint, "Learn Apache Spark - Simply Easy Learning", *Tutorial point* [Online]. Available: https://www.tutorialspoint.com/apache_spark/ [Accessed: 03/14/2021].

[25] F. Vázquez, "Deep Learning With Apache Spark — Part 1", *Towards Data Science Blog*, 09/04/2018 [Online]. Available: https://towardsdatascience.com/deep-learning-with-apache-spark-part-1-6d397c16abd [Accessed: 03/22/2021].

[26] F. Vázquez, "Deep Learning With Apache Spark — Part 1", *Towards Data Science Blog*, 10/05/2018 [Online]. Available: https://towardsdatascience.com/deep-learning-with-apache-spark-part-2-2a2938a36d35 [Accessed: 03/22/2021].

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

[28] 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: 02/17/2021].

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