Big Data Analytics Overview with Hadoop and Spark

Harrison Carranza, MSIS
Marist College, USA, Harrison.Carranza2@marist.edu

Mentor: Aparicio Carranza, PhD
New York City College of Technology - CUNY, USA, acarranza@citytech.cuny.edu

Abstract*– In this modern era of technology the amount of data and information has increased vastly. It is essential that we acquire and utilize the appropriate software to manage our ever-growing stored data. With the use of Apache software, we can observe the distribution processing of large datasets across clusters of computers. The Hadoop software library is a framework that enables applications to work with hundreds of nodes and petabytes of data. Spark permits data analysis to be done among interactive queries and data stream processing at a very fast rate. Instead of relying solely on hardware to deliver High-Availability, these software applications are designed to detect and handle failures at the application layer. With the rapid advancement in technology, it is increasingly important to scale up applications to provide modern solutions for processing large data sets. In order to demonstrate the use of this framework, we shall describe how Apache Hadoop and Spark functions across various Operating Systems as well as how it is used for the analyses of large and diverse datasets.

Keywords — Big Data, Data Processing, Distributed Programming, HDFS, Hadoop, Parallel Computing, Spark.

I. INTRODUCTION

As the amount of data we use increases, so does the need of the analysis. In a few decades we have vastly expanded the use of technology. Personal computers originally came with a 16KB of RAM and required cassettes to load and store programs. Today, personal computers boast 8GB or more RAM with internal storage of more than 1TB.

As of 2014 Facebook has stored 300 Petabytes of data at a rate of 600 terabytes a day in order to handle its users' billions of photos [5]. These amounts of data pose problems of computing power and storage capabilities. Just 10 TB of data would take 5 nodes of parallel computing power 6 hours to process/read it. If the same data were spread across 500 nodes it would then take around 6 minutes to read. However, if we transfer this data over a network to other nodes the time would increase by some orders of magnitude.

Apache Hadoop addresses these concerns by distributing the work and data over thousands of machines in clusters. In this manner the workload is spread evenly across the cluster, which allows for effective parallel computing of data. It also ensures that only a small amount of data is transferred over the network on processing time so it suits best scenarios where data is written once and is read many times – this is known as WORM [1].

The two major pieces of Hadoop are Hadoop Distributed File System (HDFS) and MapReduce. The former is designed to scale Petabytes of storage and runs on top of the file systems of the underlying OS. It provides high-throughput access to application data. The MapReduce programming model allows the programmer to work on an abstract level without having to fiddle with any cluster specific details or communication and data flow. With the use of two functions, ***map*** and ***reduce***, we are able to receive incoming data, modify it and send it to an output channel.

Apache Spark is another form of software for performing big data analysis. It is known as a cluster computing platform designed to be fast; and used for quick processing. Spark extends beyond the MapReduce model in order to efficiently support many types of computations that are being done. Among these computations are interactive queries and data stream processing. Speed is very important when it comes to the processing of large datasets. If dataset processing is not quick enough, exploring data interactively could not take place because of the lengthy wait due to slow speed [1].

Spark was designed to facilitate the workloads that were used before as individual distributed systems. Included among these systems were batch applications, iterative algorithms, interactive queries, and streaming. Since all workloads are now performed and processed in one engine, Spark comes into play to cover the wide range of tasks as an easy and inexpensive method of combination of processes. It is also beneficial to Administrators as they are not required to manage various tools located all over a complex system [2].

II. GRID COMPUTING

Grid computing is the collection of computer resources from multiple locations to reach a common goal. It is distinguished from conventional high performance computing systems such as cluster computing in that grid computers have each node set to perform a different task/application. The general approach is to distribute the work over the nodes but leave the data on a central storage; this works well for compute-intensive tasks but eventually becomes a bottleneck for tasks, which require access to the whole dataset.

Hadoop follows a different approach by attempting to locate the work units on the same nodes as the data for this unit is hosted. This is important because in a data center the

network bandwidth becomes the most precious asset if the amount of data is large. After all, Hadoop consists of the three primary resources: HDFS, MapReduce programming platform, and the Hadoop ecosystem, which is a collection of tools that organize and store data as well as manage the machines on Hadoop.

III. HADOOP FRAMEWORK

The Hadoop Framework consists of several modules, which provide different parts of the necessary function to distribute both tasks and data across a cluster. This is explained in further detail in later sections. A cluster consists of several nodes organized into racks, each node running a task tracker for the map reduce tasks and a data node for the distributed storage system. A special node named the Master-Node runs a job tracker and a name node, which organizes the distribution of data [6]. This is shown in Figure 1.

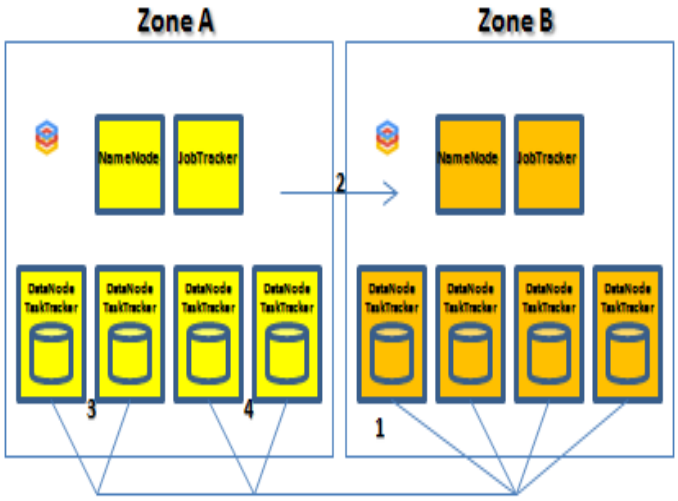

Figure 1 - Abstract view of a Cluster

Hadoop is written in Java but can run the MapReduce programs expressed in other languages such as Python, Ruby, and C++. No matter which language is implemented in; the map function will still work as long as it can read and write from standard input to standard output. In every cluster running Hadoop there is one node known as the master node. Communication from outside of the cluster is completely handled by each cluster's respective master node. It keeps all the necessary information about the cluster, usage of the nodes, disk-space and distribution of files. Each map reduce task is sent to the master node where the job tracker manages its jobs and to the name node, which is responsible for everything concerning the file system. A Hadoop cluster typically consists of thousands of nodes and is further subdivided into racks by a two-level network topology [6].

The framework's configuration should include information regarding the topology in order to allow the underlying file system the use of the location for its replication strategy. Additionally, it allows the framework to place MapReduce tasks near the data they operate on. In a distributed system with a large number of nodes of commodity hardware, failure is the biggest problem to cope with. HDFS approaches this by replicating the data to three nodes by default, one of which is to be in another rack to make sure the data is available when a network link fails and a complete rack goes offline. The file system allows for files with a size of tens of Petabytes but restricts the usage of the files to streaming-access because the "time to read the full dataset (*or large portions of it*) is more important than latency in reading the first record [7]." Other file systems such as the CloudStore or the Amazon Simple Storage Service can be used instead of HDFS too; the framework provides an abstract class and several implementations for different storage systems.

IV. HDFS STRUCTURE

A Hadoop cluster running HDFS as a file system always has a central Name Node (as shown in figure 1) that handles the data distribution and any operations on the data. It communicates with the Data Node, which is running on every single connected node and performing the disk access. Every Data Node periodically reports a list of its blocks to the Name Node so the Name Node does not need to write this information to disk all the time. The Name Node just persistently stores information about the file system-tree and the metadata of the files.

The Data Nodes are responsible for serving Read and Write requests from the file system's clients. The Data Nodes also perform block creation, deletion, and replication upon instruction from the Name Node. The read- and write-requests from clients are managed by the Name Node but executed by the Data Nodes because the Name Node has the file systems' metadata and knows which nodes store the requested blocks. The user data is never going through the Name Node because this would quickly saturate its' network connection [9].

Hadoop's Distributed File System was designed with the purpose to reliably store very large files across machines in a large cluster. It was inspired by the Google File System. HDFS stores each file as a sequence of blocks, in which all blocks in a file except the last block are the same size. Blocks belonging to a file are then replicated for fault tolerance. The block size and replication factor are configurable per file. Files in HDFS are "*write once*" and have strictly one writer at any time. The goal of the HDFS is to store large data sets, cope with the hardware failure, and emphasize streaming data access.

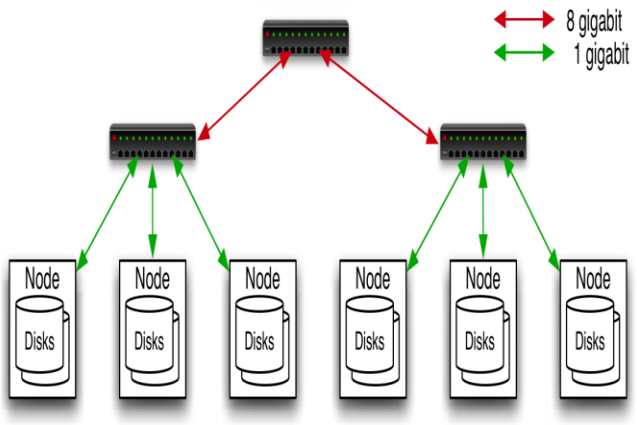

Figure 2 - Typical Hadoop Structure.

As shown in Figure 2, the commodity hardware consists of Linux PCs with 4 local disks. This is a typical level 2 architecture with 40 nodes per rack. The uplink from the rack is approximately 8 Gbps and the rack internal is 1 Gbps all-to-all. This allows for a single namespace for the whole cluster, which is optimized for streaming reads of large files. The files are broken into large blocks of about 128 MB and replicated to several data-nodes for reliability. The client then communicates with the name-node and the data-nodes. The throughput of the file system scales parallel to the number of nodes. In large clusters, some nodes might delay performance. If this occurs, then the framework re-executes failed and malfunctioning tasks. MapReduce queries HDFS for locations of input data and tasks are scheduled close to the inputs when and if possible [3].

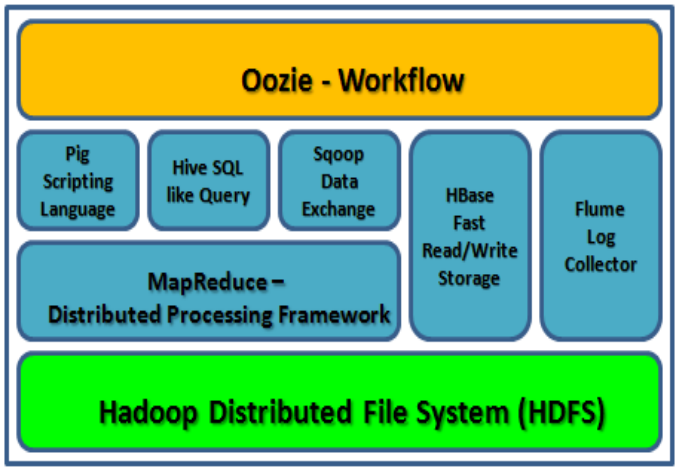

Figure 3 - Overview of the Hadoop Ecosystem

The Hadoop Ecosystem consists of many projects such as Sqoop, Flume, Oozie, Whirr, and ZooKeeper, which are all linked and work with the Hadoop Distributed File System, as show in Figure 3 [3].

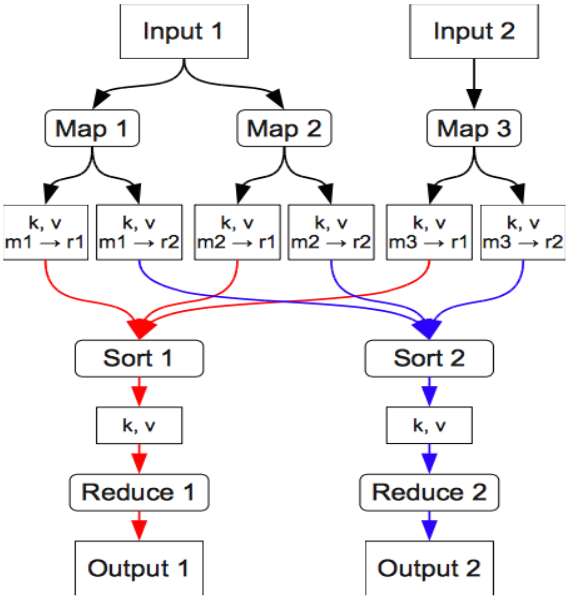

Figure 4 - Data flow in a Hadoop Cluster.

Figure 4 depicts the flow in a Hadoop Cluster – here, the input data is passed to the Map jobs, which apply their function and output key-value pairs grouped by the key and sorted. Every Reducer gets the pairs for one key and outputs the processed data into the distributed storage. They are then sorted out, reduced, and the output is obtained.

V. MAP REDUCE AND WORD COUNT

MapReduce was the first and is still the primary programming framework for developing applications in Hadoop. MapReduce functions with the use of Java. This application has jobs that consist of Java programs called mappers and reducers. The way that it works is that each mapper is provided data that is to be analyzed. Using a sentence for an example, we could analyze the data [4], [8].

To illustrate how MapReduce functions, we can use an example called "Word Count". If we create a sentence, we could analyze it from top to bottom and create name-value pairs or maps. For experimentation purposes, "The dog at the food" is going to be our sentence. Let's assume it gets a sentence: "My cat went to my bed." The maps are: "my":1, "cat":1, "went":1, "to":1, "my":1, and "bed":1. The name is the word, and the value is a count of how many times it appears on record. In this example, all the words would show up with a 1 next to it but "my" would be displayed as "my":2 once the processing is done [4].

The job of Hadoop is to process all the data, in this case, the sentence, and arranges it as needed. Then they are assigned to reducers in shuffles, sums up all the maps for each word, and produce the words listed in a document. Essentially,

mappers are programs that extract data from HDFS files into maps, and reducers as programs that take the output from the mappers and overall aggregate data [4].

Here is a thorough example involving Word Count. It is the most commonly used example to explain the usage of the map and reduce functions. It takes the input text and sums up counts for every appearing word. It can also optionally be installed as a combiner class to save on bandwidth. It sums up all counts of one map task prior to sending the result.

Here is an example source code:

```
virtual void Map(const MapInput& input)
{
      const string& text = input . value () ;
     const int n = text.size();

for (int i = 0; i < n; )
{
          // Skip past leading whitespace
          while ((i < n) && isspace(text[i]))
           i++;
          // Find word end
          int start = i;

while ((i < n) && !isspace(text[i]))
          i++;
          if (start < i)

Emit( text . substr ( start , i−start ) ,"1") ;
          }
           }
virtual void Reduce(ReduceInput∗ input)
     {
     // Iterate over all entries with the
     // same key and add the values
     int64 value = 0;
          while (!input−>done())
          {
value  +=  StringToInt(input−>value());  input−>NextValue
();
 }
          // Emit sum for input−>key()
          Emit( IntToString ( value ) ) ;
          }
```

Sorting in Hadoop normally is an implicit merge-sort, which is done by the framework and does not need to be implemented. In the examples supplied with the Hadoop source code the Map and Reduce functions are left empty, there is only a special partitioning function that tries to find evenly distributed partitions based on a sample of a small subset of the keys. The sorting takes place on the nodes that perform the (empty) mapping function and the result of this process is merged together by the reducers.

VI. HADOOP EXPERIMENTATION

This experiment explores the functionality of Hadoop across various operating systems. We have tested it with the following OS versions:

- *Windows 7*
- *Ubuntu Linux 14.04*
- *Mac OS X Yosemite*

The main goal of this initial part of the experiment was to perform simple Hadoop installation in order to play around with the software and to learn more about how it works. In order to accomplish this, I installed a single node cluster on each of the three different environments (see [10] for a complete tutorial).

After enough basic knowledge was gained in Hadoop, I decided to implement a multimode cluster in each of the operating systems in order to discover any differences that may surface amongst them.

Oracle VM Virtual Box Manager in order to create the cluster of virtual operating systems. Figure 5 demonstrates the three operating systems inside the hypervisor.

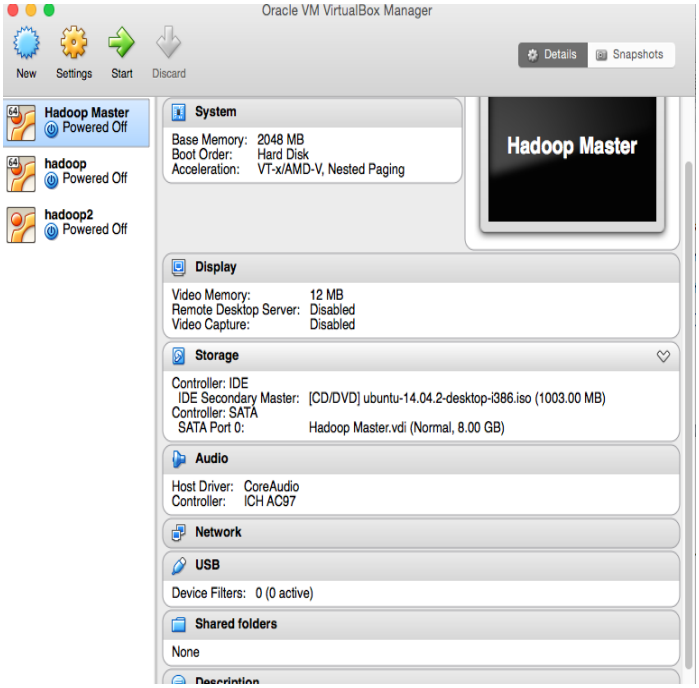

Figure 5 - Ubuntu Linux Hadoop multimode cluster.

In Figure 5, we can observe the startup menu of the Hadoop Application, ready to be used with Ubuntu Linux. From here, we can perform the experiment of processing the necessary data.

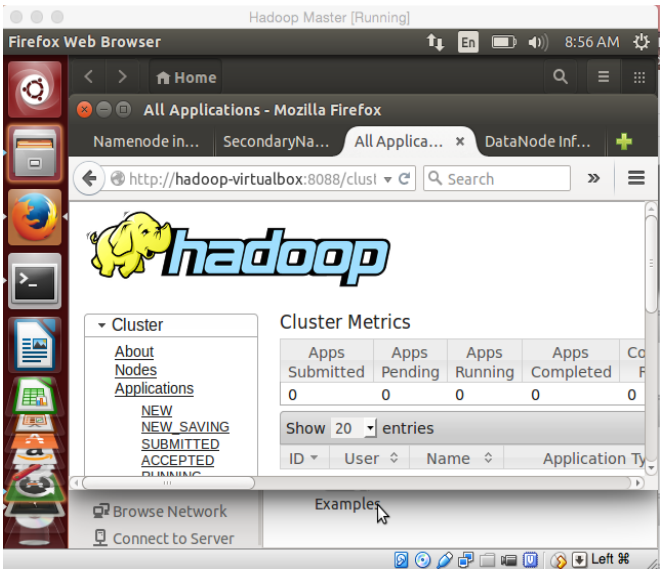

Figure 6 - Hadoop cluster running on Ubuntu

The processing of the data is carried out from Wikipedia's data dump [11] with the word count example program, as shown in Figure 7. There was a limitation to a few TB of data due to storage limitations, but this had no effect on the core principals of the experiment. In order to ensure consistency, the same exact data and amount of memory with each of the different operating systems was used.

Index of /enwiki/latest/

../		
enwiki-latest-abstract.xml	16-May-2015 10:07	4562685095
enwiki-latest-abstract.xml-rss.xml	16-May-2015 10:07	751
enwiki-latest-abstract1.xml	16-May-2015 06:07	748232458
enwiki-latest-abstract1.xml-rss.xml	16-May-2015 10:03	754
enwiki-latest-abstract10.xml	16-May-2015 05:27	160692277
enwiki-latest-abstract10.xml-rss.xml	16-May-2015 10:05	757
enwiki-latest-abstract11.xml	16-May-2015 07:08	154936835
enwiki-latest-abstract11.xml-rss.xml	16-May-2015 10:05	757
enwiki-latest-abstract12.xml	16-May-2015 08:36	172178507
enwiki-latest-abstract12.xml-rss.xml	16-May-2015 10:05	757
enwiki-latest-abstract13.xml	16-May-2015 09:50	163055214
enwiki-latest-abstract13.xml-rss.xml	16-May-2015 10:05	757
enwiki-latest-abstract14.xml	16-May-2015 09:57	156122136
enwiki-latest-abstract14.xml-rss.xml	16-May-2015 10:05	757
enwiki-latest-abstract15.xml	16-May-2015 09:06	142645608
enwiki-latest-abstract15.xml-rss.xml	16-May-2015 10:05	757
enwiki-latest-abstract16.xml	16-May-2015 09:45	153311991
enwiki-latest-abstract16.xml-rss.xml	16-May-2015 10:05	757
enwiki-latest-abstract17.xml	16-May-2015 07:28	150961156
enwiki-latest-abstract17.xml-rss.xml	16-May-2015 10:06	757
enwiki-latest-abstract18.xml	16-May-2015 10:03	138412668
enwiki-latest-abstract18.xml-rss.xml	16-May-2015 10:06	757
enwiki-latest-abstract19.xml	16-May-2015 06:45	150821976
enwiki-latest-abstract19.xml-rss.xml	16-May-2015 10:06	757

Figure 7 - Sample Index of Wikipedia entries.

We have discovered a few interesting things while conducting this research with regards to the efficiency of Hadoop across different OS platforms. When name node memory usage is higher than a certain percentage (say 75%), reading and writing HDFS files through Hadoop API will fail. This is caused by Java Hashmap as well as its collision resolution because it stores the collided elements in a linked list. Name node's QPS (query per second) can be optimized by fine-tuning the handler counts higher than the number of tasks that can be run in parallel.

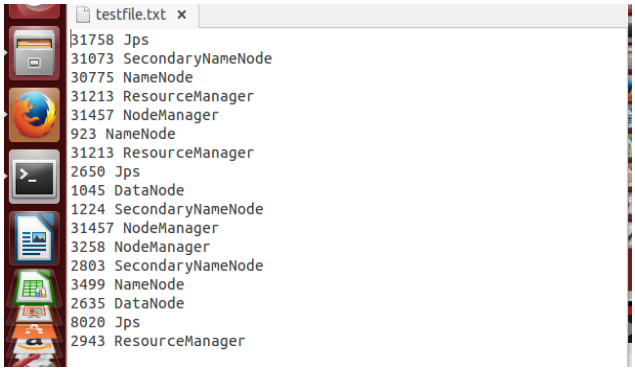

Figure 8 - Test File Processing with Hadoop

We create a test file in order to observe the processing performance of Hadoop. Keep in mind that the memory available affects the namenode's performance in each OS platform. As such you should allocate sufficient memory in order to ensure it has headroom to not fall into full garbage cycles. The results are displayed in Figure 8. Since memory has such an effect performance, it makes sense that an Operating system that utilizes less RAM for background and utility features also yields better results. In the case of this experiment Ubuntu Linux would be the ideal environment seeing as it uses the least amount of memory, followed by Mac OSX and then finally Windows.

VII. OVERVIEW OF APACHE SPARK

Spark is Apache software that is distribute and is highly scalable in the memory data analytics system. It is designed to be accessible and provides the ability to develop simple and complex application programming interfaces for many languages such as Python, Java, Scala, and SQL [1], [2]. They contain libraries within the system that have many options to perform Big Data Analysis. Spark can run in Hadoop clusters and access any Hadoop data source.

The Spark project consists of components integrated at the core. It is considered to be a computational machine that is responsible for scheduling, distributing, and monitoring applications that involve computational tasks across computer clusters. This allows machine learning to be done at a fast pace. The components are interconnected and operate together in a network, allowing all types of users to put libraries together as one framework of the project [3].

Spark is one of the leading software applications in the market when it comes to Big Data. It has one of the highest contribution and involvement rates among the current Apache top level projects. Many Apache systems use a processing engine instead of MapReduce, which is used heavily by Hadoop. Figure 9, demonstrates Spark and the variety of modules that lie within the system.

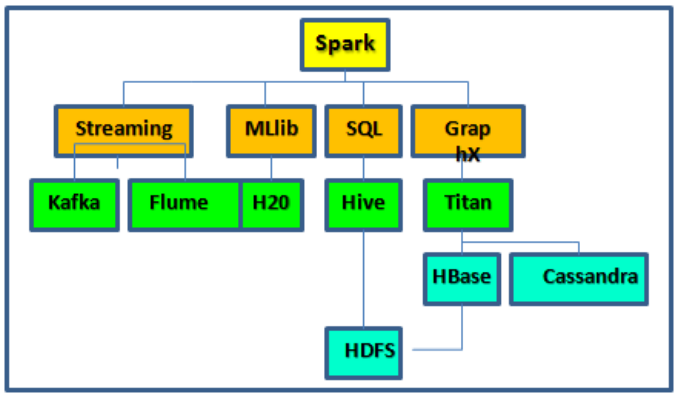

Figure 9 - Modules of the Apache Spark

Apache Spark provides four main submodules, which are SQL, MLlib, GraphX, and Streaming. The modules are interoperable, so data can be passed between them. For instance, streamed data can be passed to SQL, and create a temporary table. Above, the top two rows show Apache Spark and four submodules [1].

VIII. BENEFITS OF APACHE SPARK

Spark has several benefits that need to be taken into consideration. When Spark's core engine is optimized, the rest of the system speeds up, creating a better environment for SQL and machine learning. In addition, the cost is lower when the stacks are minimized. This demonstrates that instead of running a group or network of independent software systems, an enterprise would only use one [2]. This is considered to be a type of virtualization because the amount of physical space is reduced, benefiting enterprises.

The application would help to reduce costs in several aspects such as deployment, maintenance, testing, support, and other financial factors. When a new component is added to the software, all organizations that are running their systems on Spark are going to be required to upgrade their systems using the new components. This changes the cost of trying out a new type of data analysis from downloading, deploying, and learning a new software project to upgrading Apache Spark [2].

If core engine optimization and software component addition are considered to be beneficial to the organizations using Spark, then tight integration would be considered to be the largest advantage of the Spark application. With the use of integration, we gain the ability to construct applications that can combine different types of processing models. Just to help visualize this advantage, suppose that in Spark you can write one application that uses machine learning to classify data in real time as it gets acquired from streaming sources. Simultaneously, the resulting data can be queried by means of SQL in real time. The purpose of this process is to join the data that contains unstructured log files in the system. Here is where the Python shell would be used for further analysis on the data.

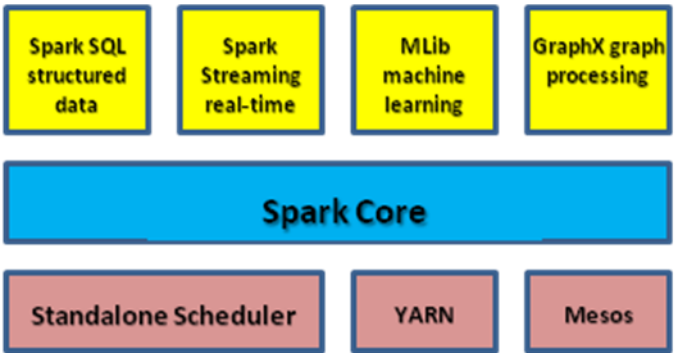

Figure 10 - Components of the Spark Stack

IX. CONCLUSION

Overall, it is evident that the growing technology that surrounds is just going to expand into new horizons. As this occurs, it is very important that we keep our storages of data secure and accessible. Hadoop and Spark do just that. Throughout this investigation, experiments have been performed with Hadoop to demonstrate that it can be used to manage large quantities of data by useful means of HDFS and MapReduce. Even though experimentation was done mostly in Hadoop and many results were obtained, Spark could be a better choice for enterprises since it was designed to facilitate the workloads used before as individual distributed systems. Now that everything is done in one engine, Spark can cover a wider range of tasks and there won't be the need of managing many tools in the system. For further research, I plan to explore Spark as well as MapReduce and HDFS in order to uncover what makes it the top option to perform Big Data processes and analyses.

REFERENCES

[1] Frampton, M.: Mastering Apache Spark. Packt Publishing, Birmingham, UK (2015)
[2] Karau, H., Konwinski, A., Wendell, P., Zaharia, M.: Learning Spark. O'Reilly, Sebastopol, CA (2015)
[3] Barot, G., Mehta, C., Patel, A.: Hadoop Backup and Recovery Solutions. Packt Publishing, Birmingham, UK (2015)
[4] Sitto, K., Presser, M.: Field Guide to Hadoop. O'Reilly, Sebastopol, CA (2015)
[5] Cohen, D: http://www.adweek.com/socialtimes/orcfile/434041, 04/11/14
[6] https://cloud.google.com/resources/articles/managing-hadoop-clusters-on-google-compute-engine2 (2014) Google Cloud Platform.
[7] White, T.: Hadoop: The Definitive Guide. Second Edition, Yahoo Press, (2009)
[8] Lin, J., Dyer, C.: Data-Intensive Text Processing with MapReduce (Synthesis Lectures on Human Language Technologies). Morgan and Claypool Publishers (2010)
[9] Apache Hadoop http://hadoop.apache.org/ (Retrieved: 2016-12-12)
[10] Noll, M. G.: http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/
[11] Wikipedia: http://dumps.wikimedia.org/enwiki/