Comparative Analysis of Spark with different Storage Layer

Maulik Patel
Illinois Institute of Technology
Mpate111@hawk.iit.edu

Sushma Mahadevan
Illinois Institute of Technology
smahade4@hawk.iit.edu

Ioan Raicu
Illinois Institute of Technology
iraicu@cs.iit.edu

ABSTRACT

Rapid development in technology have increased the data exponentially in last decade. Currently the biggest challenge in the Data Science and Distributed Systems field is analyzing the huge amount of data with higher performance and lower latency. Various Framework are developed that helps in computation and storage of real time data across multiple system. One such framework that helps faster data processing is Spark. Spark consist of in memory storage RDDS that help in faster data computation but the in memory storage is limited. For purpose of storing large data in distributed environment and gaining high speed up various storage layer like file system HDFS and No-SQL database like Cassandra are integrated with Spark. In this paper we will compare performance of spark with various storage layer like No-SQL database (Cassandra) and different file system (hdfs, fusion fs). Further extending our work we dig into the datastax-connector code for seeing the actual implementation.

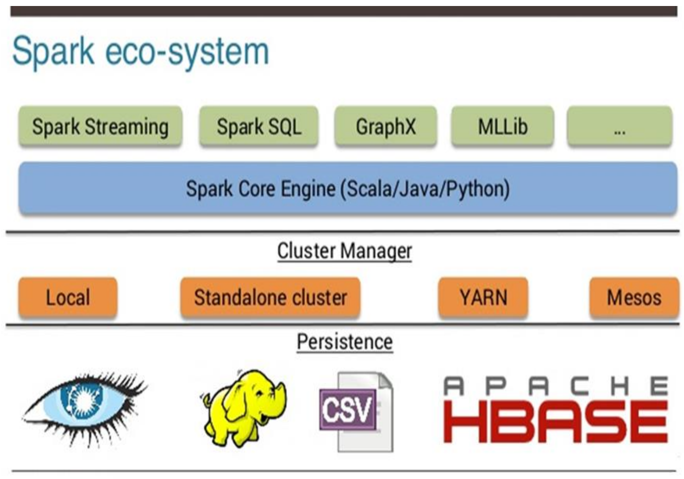
1. INTRODUCTION

Spark was introduced on top of HDFS as it does in-memory computation making it faster compared to Hadoop for some iterative application like graphs and machine learning. Spark like Hadoop and Dryad allows us to write the parallel computations using the set of high-level API without to worry about the work distribution and fault tolerance. Since the data is increasing with the exponential rate it is equally important to store that data and for storing the large scale of data RDBMS are not efficient and to overcome that we use distributed No-SQL database such as Cassandra and ZHT to store the large amount of data on various node on the cluster. For effective data analytics we can use spark on top of Cassandra which provides the computation power of spark and fully distributed nature of Cassandra using the connector to hook them. Since both HDFS and Cassandra are used by a large customer based and both have certain pros and cons for handling big data so in this paper we tried to compare both the file system with spark. In this paper we have first described about the architecture of Spark, HDFS and Cassandra in background information. In the later section the paper describes the proposed solution followed by evaluation section where all the results have been described in form of graphs to support the conclusion. At the end we have described about the similar work that have already been implemented in the related work section.

2. BACKGROUND INFORMATION

2.1. Spark

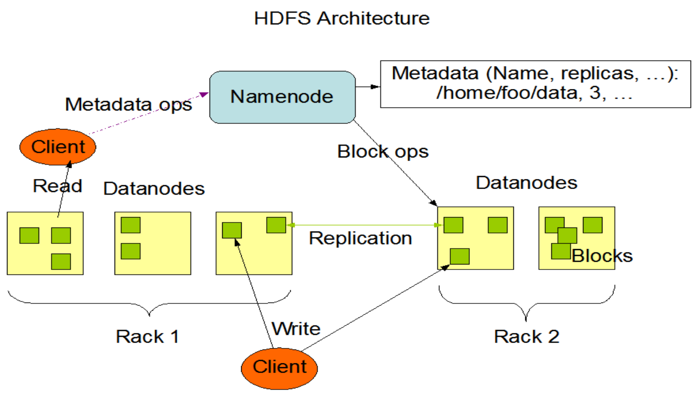
Spark is framework built for distributed systems for large-scale data processing developed by AMPLAB students using the map-reduce paradigm and was open-sourced by Apache in 2010. The main abstraction in Spark is that of a resilient distributed dataset (RDD), which represents a read-only collection of objects portioned across a set of machines that can be rebuilt if a partition is lost making it fault tolerant [1]. Spark offers over 80 high-level operators that make it easy to build parallel apps and also it provides interactive shell for Scala, Python and R shells [2]. Spark outperforms Hadoop by up to 20x in interactive applications and can be used interactively to query hundreds of gigabytes of data [3]. Spark can be integrated with most of the distributed file systems like HDFS and also with database like Cassandra and HBase since it doesn't have its own storage layer. The below figure shows the Spark ecosystem.



2.2 Hadoop File System

The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. HDFS was built on the concepts of Google File System (GFS) basically as the storage layer of Hadoop for processing larger data. HDFS can be used in large-scale distributed storage to build a scalable cloud storage platform with open-source project, high fault tolerance and high performance [4]. HDFS cluster consists of master-slave architecture which consists of a single NameNode, a master server that manages the file system metadata and regulate access to files by clients and many DataNodes, where the actual data is stored in the local file system. Internally the big file is sharded into small blocks and are stored in a set of DataNodes. All the DataNodes will send the heartbeat message to the master node and if master node doesn't receive the message then master node considered it as dead and will not send any further I/O request also the NameNode.

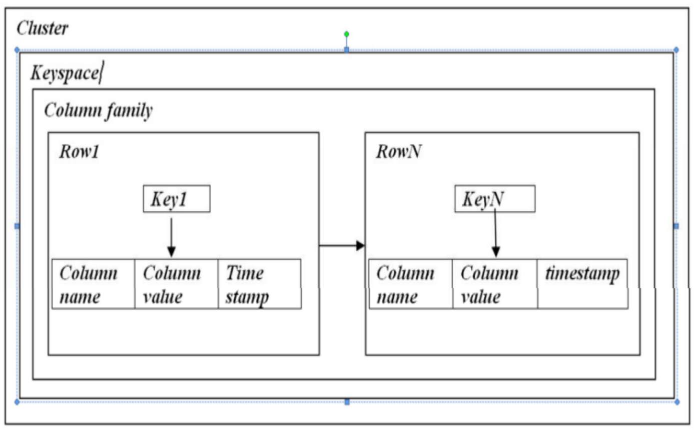
Below figure shows the master-slave architecture and working of HDFS.



There are certain limitations Since HDFS is a master-slave architecture if the name node fails then all the metadata is lost and the system will be down making it as the single point of failure. Since it is designed to store big data so for the smaller size files throughput is decreased. Also it is comparatively hard to configure the HDFS across multiple nodes.

2.3 Cassandra

Cassandra was initially developed at the Facebook to power up the Facebook inbox search feature. Facebook released Cassandra as an open-source project in 2008 and after which it became the an Apache Incubator project and became the top priority project in 2010.[1] Cassandra is a distributed storage system for managing very large amounts of structured data spread out across many commodity servers, while providing highly available service with no single point of failure[2]. Cassandra has Peer-2-Peer architecture making any node to join or leave the cluster easily without need of restarting the whole cluster again. Cassandra is not fully relational data model as a contrast it provides dynamic control over the layout and format with tunable consistency from eventual to strongly consistent data-model



Hierarchy Of Cassandra Model

Cassandra implements the schema from DynamoDB and Google Table. It implements the data replication at the key-space which is similar to schema in MySQL. The way all the nodes communicate with each other on basis of Gossip protocol. Based on the row key in the column family data is broken across the nodes. We can interact with the Cassandra cluster with the help of Cassandra Query Language (CQL). Cassandra provides high throughput and low latency as the data is first written in the memory structure called mem-table and only when it is full it is to the SStable and then to the disk. Cassandra implements two protocol for partitioning the data equally across all the nodes namely RandomPartitioner which internally implements consistent Hashing to distribute the data and also the default implementation for Cassandra and ByteOrderedPartioner. Cassandra is mostly useful for time-series and real-time applications.

3. PROBLEM STATEMENT AND PROPOSED SOLUTION

Spark in-memory computation makes it an ideal choice for certain iterative applications but the storage layer of the Spark can be optimized by using No-SQL distributed database like Cassandra instead of using the default storage system HDFS. With the increasing data volume and also its use for different types of applications in the present era it is critical to manage both structured and unstructured data, Since the way data to be stored impacts the analytics and advantages by extracting the useful information from the raw data it become equally important to know what type of data storage layer would be suitable for particular group of applications. In this paper we try to run different type of application like sort, word count and statistical operation like finding the maximum, minimum, average and standard deviation for measuring the performance. Comparing the various storage options, we try to analyze efficiency of storage in terms of operating with spark jobs based on latency and throughput.

Spark-Configuration and Dataset Used -:

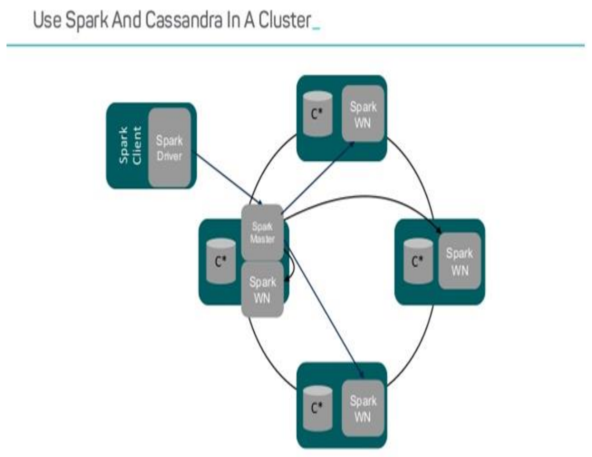
Since we are using Amazon EC2 instances for setting up our cluster so we can use the inbuilt script that is written in Boto for starting the spark cluster using the inbilt commands in spark. For Cassandra we used datastax-3.7.0 version developed by datastax community .We used spark version 1.6.0 compatible with HDFS 2.4 version and above.

For Sort application the data file is generated using gensort which creates lines of 100 bytes each of file size specified. The first 10 bytes of a line were considered as key and next 90 bytes value. Sort was performed based on key first 10 bytes. For the word count application, we used node-lipsum code to generate the data required for performing the experiment. For the statistical analysis we generated a random number generator code in Java for generating the dataset.

Cassandra cluster configuration -:

Cassandra use peer to peer architecture and communication with peer takes place using a contact point named as seed which uses gossip protocol for connecting between the nodes in the cluster. In cassandra.yaml file change seed value gossip contact point to the ip of nodes which we want to be as the communication point for all peer nodes to connect and pass data information. We can have one seed or as many seeds as we want but it is recommended to use a single seed point. For each peer node we have to change the listening address to private ip address of that node and broadcast address to the public ip for that node

All the code was developed using Java API. We installed Oracle java version 1.8 for Cassandra and Spark. Each code was around 100 lines. We used datastax Cassandra and Spark Connector API to gather data from Cassandra and process the input data with application running on the Spark which distributes the application to the nodes of the cluster.



Spark-Cassandra connector is developed and open-sourced by Datastax. Datastax connector provides us high level API in various high level languages like Java, Python, C etc. which can be used for accessing the Spark jobs over Cassandra for retrieving and processing the data to extract the meaningful information that can be used for various analysis purpose.

Since Spark is a distributed analytical engine Spark job is distributed across each worker node which as per the applications makes connection to Cassandra using the inbuilt API that reads the data and perform the computation.

The Spark-Cassandra connector contains API such as saveToCassandra () which is used to store the RDD back to the table in form of rows and column back to Cassandra, cassandraTable () to get the data from Cassandra to Spark in form of RDD etc. . Also the connector provides various set of API which can be used to perform filters, joins and various data analytics on specific rows and columns of a table. After getting the data from Cassandra we can also use various Spark streaming API for performing various statistical operations

4. EVALUATION

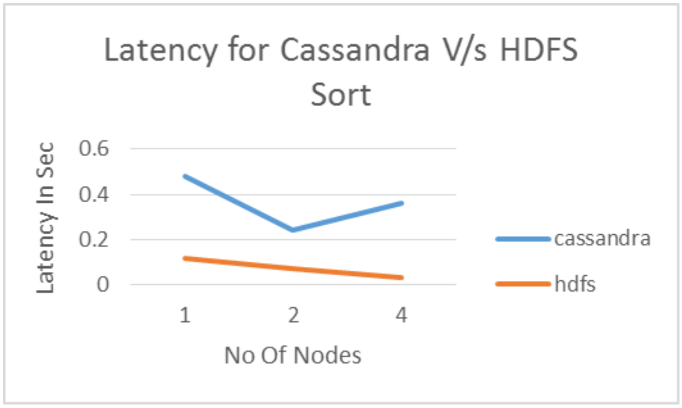
All the applications performance were evaluated on Amazon EC2 m3.large instances by scaling the number of nodes in the cluster. We executed various set of applications on both HDFS and Cassandra as storage layer with varying data sizes to determine and compare the performance of the application on either of the storage systems. The experiment were carried out on the spark cluster and based on the comparison of its empirical data we measured what type of applications gives better performance for the corresponding storage system. We compared the results for applications by scaling from smaller dataset with 2 workers to large dataset using 8 workers that process the input dataset from Cassandra and HDFS. We evaluated sort and word count application for 1, 2 4 and 8 nodes with data size of 5 GB each node.

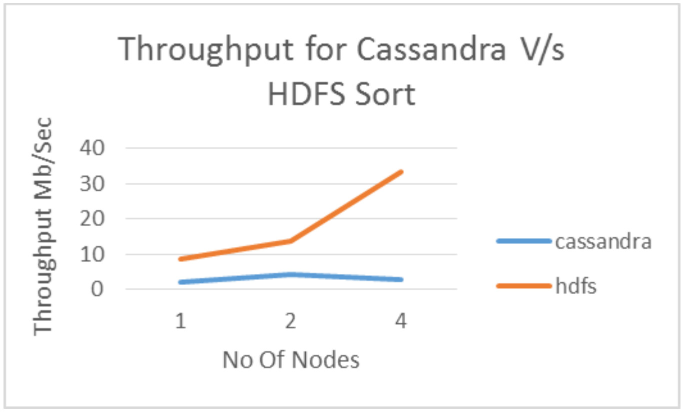
Certain applications e.g. sorting the data is better with HDFS as the storage layer since it processes the data in big chunks of 64MB compared to Cassandra where the data is sorted in row form and based on the primary column that we set during writing the data into table row by row across various nodes increasing the latency and degrading the performance where for the application like word-count where we have less number of rows as input and output Cassandra performs much better when compared to HDFS.

We also performed common statistical operation like finding the minimum, maximum, average and standard deviation using spark on specific set of data where columns contain the integer values to verify performance of Cassandra and HDFS underneath the Spark. Since Cassandra is a NoSQL database storing the data in structured and normalized manner gives the leverage for better performance for the statistical operations like finding minimum, maximum, average e.t.c. We can perform similar statistical operations with the data stored in HDFS but since the data is stored in unstructured manner it poses certain limitations in which format the data should be. It should be having certain type of delimiter like comma, blank-space or colon.

4.1 Sort

The following graph shows the sort latency and throughput for Spark using HDFS and Cassandra as the storage layer. Here we have taken 5GB per node which is equivalent to 50 million rows in Cassandra per node.



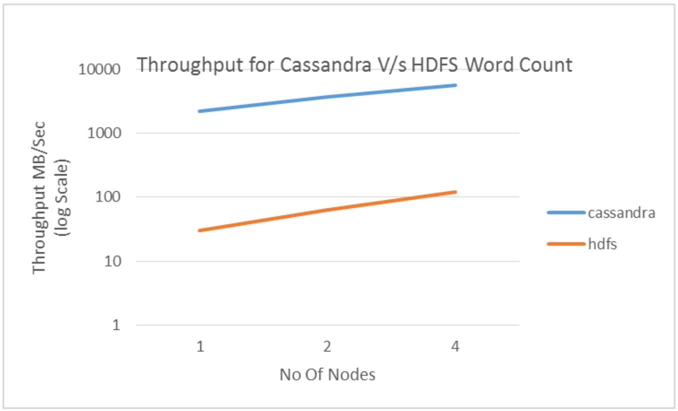


.

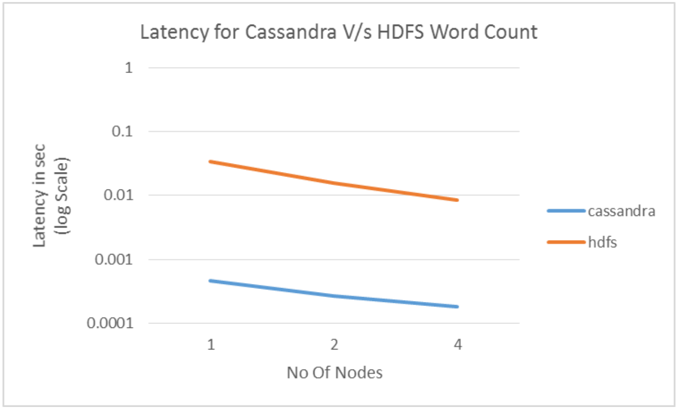
Here as the data size increases with the increase in node throughput increases for HDFS linearly but it remains almost same for Cassandra as there is disk IO bottleneck for writing data to Cassandra as there are around 100 million records for 2 nodes and 200 million for 4 nodes decreasing the throughput and increasing the latency since it imposes the I/O bottleneck on the disk as the data is processed in form of rows compared to block in HDFS.

4.2 Word Count -:

The following graph shows the throughput and latency for the word-count application and here the data generated is with node-lipsum code for each node 5GB and around 400 rows per node in Cassandra.



Throughput

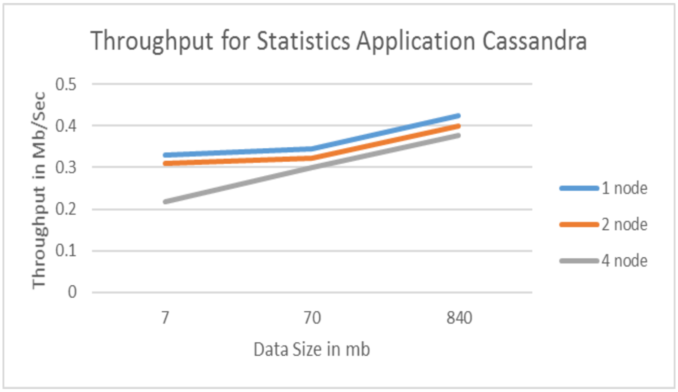


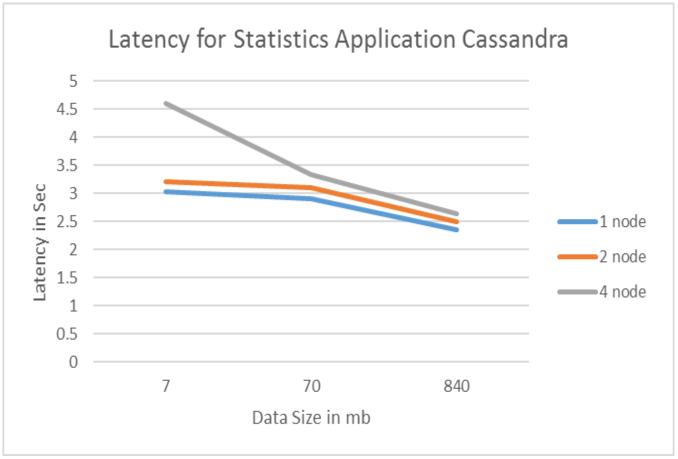
Latency

Here the data ranges from 5GB per node and we scale our experiment from 1 to 4 nodes. There is a linear increase in the throughput and decrease in the latency for both Cassandra and HDFS. As seen from the graph Cassandra has better performance compared to HDFS as there are less number of rows which decreases the I/O bottleneck of reading and writing from the database as compared to Sort data. Also after the data is processed the output that we get is of around 4000 records which is stored back in Cassandra. Since Cassandra stores the data in-memory so it is much faster as compared to HDFS as it stores the data in disk. Here we can observe that the data is in same size but because it occupies in less number of rows in Cassandra it is much faster as data is processed in form of rows compared to blocks in HDFS. So if we want to process big data with Cassandra then to optimize we should accommodate more data in less records.

4.3 Statistical Problems -:

The following graph shows the latency of processing the data stored in Cassandra and doing simple statistical operations finding minimum, maximum, average and standard deviation. Here we have varied the data from 7 MB to 840MB i.e. 1 million to 100 million records in form of Cassandra rows. The data was generated using a random number generator code written in Java.





The above graph describes the throughput and latency when we increase the data on the cluster of 1, 2 and 4 nodes Here we have taken the average of the time of all the four opertions i.e. minimum, maximum, average and standard deviation for better representation. Analyzing the graph we can see that as the data is increased i.e. increasing the rows in Cassandra table the throughput increases and the latency decreases for the cluster consisting of 1, 2 and 4 nodes. Thus we can see that Cassandra performs better i.e. improved throughput and reduced latency with increase in data. For performing this kind of statistical analysis we require the data in structured manner so for the application requiring this kind of information e.g. in stock market appliation if we want to know the highest and lowest price of stock, average or standard devation for a day, month or year then Cassandra will be much better compared to HDFS.

5. RELATED WORK

Exponential increase in data had many challenges to store and manage the data. Many studies in the area of file system and database community to distribute the data across multiple node has been done for enhancing the performance, availability and durability. Comparative analysis of HDFS and Cassandra have been done heavily in the community to put which would be best data storage option for the specific set of applications.

The ideal applications for HDFS are the one that has large data sets, very large files, fault tolerant which ensures that failures of hardware or software doesn't cause data to be lost mostly for the batch processing whereas the Cassandra is ideal for the applications having requirement like flexible sparse wide column requirements, very high velocity random reads and writes mostly in which the latency is critical compared to throughput.

Many studies have been made to compare Hadoop on top of HDFS and Cassandra to see that which is better at storing the Big-Data for what specific types of best suited applications. For evaluation various experiment have been done using a large data sets one of the experiment includes from altering satellite image data by removing undesired areas to create high value images ranging from around 4 million of input records to 64 million records. Empirical data gathered from the evaluation suggest that Hadoop Cassandra-Cassandra is 1.1 times slower compared to Hadoop HDFS-HDFS processing 4 million of records and it gets only 1.8 times slower for 64 million records for smaller output. For same input and output size Hadoop HDFS-HDFS i.e. data is taken from HDFS and written back in HDFS is around 1.9 to 2.9 faster compared to Hadoop Cassandra-Cassandra i.e. data is taken from Cassandra and written back to Cassandra.

There are many applications that are built using Spark with Cassandra one such reference application named killer-weather which uses Spark, Cassandra and Kafka focuses mainly on the use case of time-series data. Basically the app collects all the data from different weather stations which are having unique ID for identification purpose which act as a primary key in sorted and sequentially manner as Cassandra allows businesses to identify meaningful characteristics in their time series data as far as possible to make clear decision about expected future outcomes. Spark streaming which we use as our analytical engine allows us to window the data by specific year, month, day and hour.

6. CONCLUSION

Choosing efficient storage is necessary step in big data analysis. In this paper we compared Spark on top of HDFS and Cassandra by running various Spark jobs in distributed environment.

Comparing the performance of Spark on varying storage layer by running different set of applications we analyzed various factors that affect the performance of application and the type of applications that are suited for the appropriate storage layer.

Our experiments on Spark using No-SQL database and Hadoop Distributed File System(HDFS) indicate that Hadoop Distributed File system which is based on master –slave architecture best suited for applications which work with unformatted and large dataset processed mostly for batch jobs by Spark and which requires fault tolerant and scalable architecture. HDFS is basically not recommended for the smaller files.

Cassandra No-SQL database that implements peer to peer architecture works best with application that requires to process formatted input and where latency is more critical compared to throughput like online transactions. Also it is best suited for jobs that require to compute mathematical statistical operations like finding range of data on basis of particular object, finding minimum, maximum, average etc. Since in Cassandra all the nodes are treated equally there is no single point of failure making it suitable for the applications that require constant uptime like real time applications eg Weather Forecast.

REFERENCES

[1] M. Zaharia, M. Chowdhury, M. Franklin, S. Shenker and I. Stoica. Spark: Cluster Computing with Working Sets. In UC Berkley.

[2] Spark http://spark.apache.org/

[3] M. Zaharia, M. Chowdhury, M. Franklin, T. Das, A. Dave, J. Ma, M. McCauley, S. Shenker and I. Stoica. Resilient Distributed Dataset: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In UC Berkley

[4] http://www-bcf.usc.edu/~minlanyu/teach/csci599-fall12/papers/nsdi_spark.pdf

[5] http://www.slideshare.net/rhatr/tachyon-and-apache-spark/11

[6] https://databricks.com/blog/2015/06/16/zen-and-the-art-of-spark-maintenance-with-cassandra.html

[7] https://people.csail.mit.edu/matei/papers/2015/sigmod_spark_sql.pdf

[8] https://people.csail.mit.edu/matei/papers/2010/hotcloud_spark.pdf

[9] K. Dwivedi and S. Dubey 2015: A Taxonomy and Comparison Of Hadoop Distributed File System With Cassandra File System.

[10] http://edlib.net/2015/icidret/icidret2015037.pdf

[11] http://www.arpnjournals.com/jeas/research_papers/rp_2015/jeas_0915_2492.pdf

[12] http://www.datastax.com/wp-content/uploads/2012/09/WP-DataStax-HDFSvsCFS.pdf