Evaluating the Query Performance of Spark-SQL and Hive

Anish Joshi
School of Computing
Clemson University
anishj@g.clemson.edu

Pallavi Karan
School of Computing
Clemson University
pkaran@g.clemson.edu

***Abstract* – With the ever increasing amount of data and the rise of big data applications, data retrieval in a timely and efficient manner is one of the prime factors of concern. Querying big data using the standard querying technologies is very time consuming, inefficient and thus modern querying frameworks like Spark-SQL and Hive have been designed specially to handle this problem. These frameworks are highly efficient in dealing with big data as they make use of multiple processors/cores to distribute their work when handling such large datasets. In this paper, we evaluate and compare the query performance of two major big data querying platforms namely, Spark-SQL and Hive along with their performance variations with the change in the amount of data and the number of compute nodes respectively.**

Keywords—Apache Spark, Apache Hive, SQL, Parallel Architecture, Query Performance, Amazon Web Services, Elastic Map reduce

I. INTRODUCTION

In the past decade as we have moved towards a more digital world, the amount of data that is being generated and collected has increased drastically and so there is a need for more advanced data extraction and querying platforms which are capable of handling big data in an efficient manner. To this extent, there has been a lot of research in the field of big data extraction and retrieval for querying large datasets as standard data querying platforms are inefficient in handling such a huge scale of data. Modern big data querying platforms like Spark-SQL and Hive take advantage of multiple cores/processors to run queries on large datasets solving the problem of querying big data efficiently.

While Spark-SQL and Hive take advantage of the multiple processor/core phenomena, their actual performance varies drastically depending on the underlying architecture of the multicore system. It can be possible that Spark-SQL might perform better on certain multicore architectures while Hive may perform better on other multicore architectures. One cannot simply determine which of the two querying platforms might perform better just by looking at the data. This might depend on a variety of factors such as number of compute nodes, memory, network latency, number of input-output requests, disk speed etc. Hence it is important to perform an experimental analysis to understand the behavior of these querying platforms with respect to the above mentioned parameters. By conducting such experiments it would be possible to establish a relationship between size of the data, query type, number of compute nodes, and type of compute nodes etc. thus enabling one to predict the behavior of these platforms.

With an understanding of scaling properties for data and compute nodes when using Spark-SQL and Hive, an end user will no longer have to run his/her application on multiple frameworks to find the best performing framework for their requirements. Instead using the derived relation, the user can make an estimate depending on the data size and the availability of compute nodes to solve their big data querying problems.

The rise of cloud platforms like Amazon Web Services provide the user with the flexibility to choose any desired hardware and software as per their requirement along with any data processing framework that best suits the task at hand. This is contrary to traditional cases where a user had to build his own architecture by purchasing hardware and software which are not very scalable, modifiable and generalized for all analytical problems. Cloud platforms like AWS have thus made it easy for users to choose and switch back and forth between different hardware resources and data processing frameworks depending on the user's needs at any particular time.

In this paper, we conduct an experiment to evaluate the query performance (runtime) for two major big data processing engines namely Spark and Map-Reduce, by running queries on Spark and Hive (Map-Reduce framework) platforms respectively. Our experimental study was performed using the AWS-Elastic Map Reduce environment, configured with Hive and Spark frameworks respectively.

The following are the contributions of this paper

- To perform an experiment evaluating the query performance on Spark and Hive platforms.
- To study the effect of number of compute nodes on the query performance
- To evaluate the scalability of the querying platform for different data sizes

II. RELATED WORK

In this section, we overview some of the related works in the field of big data query performance evaluation.

The paper [14] discusses the performance difference between Hadoop and spark for iterative operations. The authors suggest that the variation in performance is because of the difference in which Hadoop and Spark store the data they operate on. While spark stores its data in memory, it is comparatively faster than Hadoop which stores the data on disk.

Xiaopeng and Zhou in their paper [12] focus mainly on comparing the query performance of querying platforms like hive, impala and spark. The performance for all the platforms is measured for different file formats. Along with the query performance the authors also discuss about the CPU and memory performance when using different file formats.

The work by Ilias and Karatza [13] mainly focuses on comparing the querying performance of Apache Hadoop and Apache Spark when working with log files. Due to an ever increasing generation of log files, it is important to make use of distributed systems on the cloud to analyze them. It is seen that for this particular case, Spark seems to perform better than Hadoop in almost all the test cases.

The paper by Ivanov and Beer [15] is related to quantizing the performance of Spark and Hive for SQL queries using BigBench. They take into consideration factors such as CPU and memory utilization and conclude that Spark SQL is in general 6 times faster than Hive SQL. The authors also focus in depth on the performance of each of these platforms for a number of metrics which are particularly useful when trying to decide a platform given certain constraints.

III. EXPERIMENTAL METHODOLOGY

We conduct a real scale experiment to evaluate the query performance (execution time) on Spark-SQL and Hive frameworks for multiple queries with variations in the size of the dataset and the number of compute nodes. Our choice of test bench provides reproducibility through its cluster configuration options and our set of queries provides realism through fixed qualitative and quantitative results based on a set of query conditions. In this section we present the tools/platforms and the experimental setup required to conduct the experimental study. Finally, we will also discuss the experimental procedure/design used to execute the experiment on AWS platform.

i) Tools/Platforms –

Apache Spark – is a fast, in-memory data processing engine developed by AMPLab at UCBerkeley for large scale data analytics. Spark achieves much of its performance advantage by relying heavily on in-memory computations. It achieves 100 times faster performance than map reduce framework in memory and up to 10 times faster performance on disk. Spark is a generic framework which can run on multiple platforms like Hadoop YARN, Standalone, Apache Mesos, along with cloud platforms like AWS-EC2 & AWS-EMR. Spark consists of components such as Spark Core which is a general execution engine used mainly for developing and running spark programs. It also provides API's for programming languages like Python, Java and Scala. SparkSQL is another component which adds support for SQL like queries and relational processing to the Spark data processing engine. Spark-SQL can also run unmodified HiveQL queries directly and can also use the Hive Metastore for data retrieval operations. Spark-SQL is similar to Spark as Hive is to Map-Reduce for executing SQL-like queries on their respective platforms [2]. Apart from the two main components discussed above there are a number of other components like GraphX which is a graph processing framework, Spark Streaming which is used mainly for data analysis and MLlib which is a machine learning framework in Spark [1].

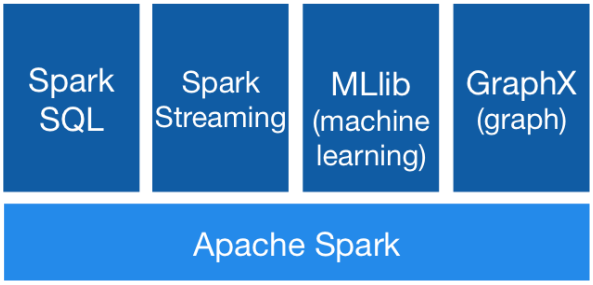

Fig 1. Components of Apache Spark [1]

Apache Hive – is a data warehouse which is built on top of Hadoop and can be used for data manipulation, reading and writing of large datasets stored on HDFS by using a structured query language known as HiveQL [3]. When a HiveQL query is submitted for execution, the Hive compiler translates this query into a Map-Reduce job on its own and thus this frees users from the tedious task of applying the Map-Reduce model themselves [4]. The Hive framework architecture is shown below [Fig 2].

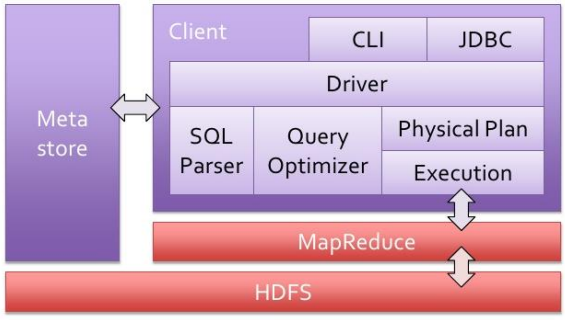

Fig 2. Apache Hive Architecture [10]

Amazon Web services – is a comprehensive, cloud computing platform provided by Amazon.com which provides on-demand delivery of compute power, applications, database storage and other information technology resources through a cloud platform via the internet with pay-as-you-go pricing [5].

AWS-Elastic Map Reduce – provides a completely managed Hadoop framework which makes it very easy and cost-effective to process large amounts of data across dynamically scalable EC2 instances. AWS-EMR also supports many popular distributed frameworks like Flink, Presto, HBase, Sqoop, Oozie and Spark, built-in which makes it very easy and convenient to integrate these frameworks with Hadoop and other AWS storage services like Amazon DynamoDB and Amazon S3 [6]. It also provides a wide selection of instances comprising of different CPU configurations, memory, storage and networking capabilities thus allowing us to scale the resources depending on our needs.

AWS-Simple Storage Service – provides the users with durable, secure and a highly scalable cloud storage service. Amazon S3 has a very simple user-interface which can be used to easily store and retrieve data from anywhere on the web. Amazon S3 can be used as a standalone service or it can also be used in collaboration with other AWS services like Amazon EC2, Amazon EMR and AWS Identity & Access Management services. With Amazon S3 you only pay for the storage you use and there is no specific minimum fee or any other setup cost associated with it [7].

ii) Dataset –

For this experiment we have used the freely available public dataset of Google Books N-Gram Corpora. The dataset is directly available at s3://datasets.elasticmapreduce/ngrams/books/ in an Amazon S3 bucket. The entire dataset is of size 2.2 TB but is not available as a single data file object. There are multiple sub datasets of varying sizes where each dataset is a single n-gram type like, 1-gram, 2-gram etc for a given input corpus such as English or Chinese text. For our experimental study we have used two such sub datasets of English text of size 5GB (1-gram dataset) and 315GB (4-gram dataset) respectively [9].

iii) Experimental Setup –

In this section we go through the initial setup of the experiment and a brief overview of the technology stack used for performing the experimental study. A simple pictorial representation of technology stack used for the experiment and its interaction is shown in the figure below [Fig 3]. As described earlier, we have used a freely available public dataset for our experiment and thus the data was readily available in a publically accessible Amazon S3 bucket and so we were able to access the dataset directly via the S3 bucket link.

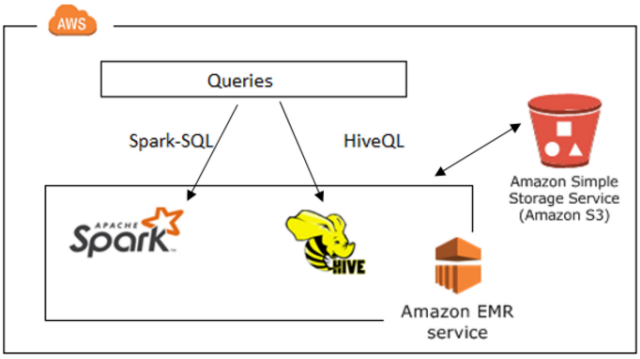

Fig 3. Technology Stack for our experiment

As a first step, we setup the AWS environment on which our big data query platforms were to be evaluated for our experimental study. To do so, we configured the Elastic Map Reduce (EMR) cluster for executing Hive and Spark in the AWS cloud environment. While setting up the EMR cluster, we configured the cluster to include Hive and Spark as preinstalled applications. Also as part of the cluster configuration, we used the general purpose m3.xLarge nodes with two different configurations- i) 1 Master with 24 Child cores (25 cores) and ii) 1 Master with 49 Child cores (50 cores) in our experimental study.

iii) Experimental Procedure/Design –

After the initial setup and configuration, we run the Hive and Spark frameworks on top of Amazon-EMR for querying data stored on the Amazon S3. SQL-like queries are run on Hive and Spark using HiveQL and Spark-SQL respectively for measuring the query execution time on the two frameworks. A total of five queries for both Hive and Spark-SQL are executed for evaluating the query performance on the two platforms with variation in the number of compute nodes and the amount of data the platforms query on. We perform four such variation studies in our experiment for both Spark and Hive querying platforms. The four variation studies are listed below –

- 5GB data with 25 compute nodes
- 5GB data and 25 compute nodes
- 315GB data and 50 compute nodes
- 315GB data and 50 compute nodes

Each of the five queries are executed thrice and then the execution time for the three is averaged to calculate the final execution time for each query so as to avoid bias in any of the query runs.

The queries executed for the experiment are as follows –

```
Q 1 - SELECT COUNT(*) FROM google_books;
Q 2 - INSERT OVERWRITE TABLE
simple_query_storage SELECT * FROM
google_books;
```

Q 3 – INSERT OVERWRITE TABLE normalized_google_books SELECT lower(gram),year,occurrences FROM google_books WHERE year >= 1890;

Q 4 – INSERT OVERWRITE TABLE sort_by_decade_google_books SELECT a.norm_gram, b.norm1_decade,sum(a.norm_occurrences)/b.norm1_total as dog FROM normalized_google_books a JOIN (SELECT substr(norm_year, 0, 3) as norm1_decade, sum(norm_occurrences) as norm1_total FROM normalized_google_books GROUP BY substr(norm_year, 0, 3)) b ON substr(a.norm_year, 0, 3) = b.norm1_decade GROUP BY a.norm_gram, b.norm1_decade, b.norm1_total;

Q 5 – INSERT OVERWRITE TABLE decade_query_google_books SELECT a.sortby_gram as s_gram,a.sortby_decade as s_decade, a.sortby_ratio as s_ratio, a.sortby_ratio / b.sortby_ratio as s_increase FROM sort_by_decade_google_books a JOIN sort_by_decade_google_books b ON a.sortby_gram = b.sortby_gram and a.sortby_decade - 1 = b.sortby_decade WHERE a.sortby_ratio > 0.000001 and a.sortby_decade >= 190 DISTRIBUTE BY s_decade SORT BY s_decade ASC,s_increase DESC; [11]

IV. RESULTS

In this section we will present the results obtained after running the queries on the above described configuration. There are a total of eight tables, four for Hive and four for Spark-SQL respectively with each table representing one type of variation in the size of the data and the number of compute nodes. We also represent the obtained results in the form of line graphs with each graph representing a specific data size and compute node configuration. Each line graph displays two lines, one for Hive and one Spark-SQL respectively, which helps us visually compare the query execution time on the two platforms in different configuration setting.

All query execution times are measured in seconds.

Query No	Runtime 1	Runtime 2	Runtime 3	Average Runtime	Standard Deviation
Q 1	26.327	25.110	25.551	25.662	0.616
Q 2	71.324	71.619	71.186	71.376	0.221
Q 3	63.396	60.613	61.576	61.861	1.413
Q 4	398.612	378.142	396.175	390.976	11.181
Q 5	79.792	78.504	83.689	80.661	2.699

Tab 1. 5 GB & 25 Compute Nodes – Hive

Query No	Runtime 1	Runtime 2	Runtime 3	Average Runtime	Standard Deviation
Q 1	0.189	0.040	0.044	0.091	0.084
Q 2	43.486	43.859	44.400	43.915	0.459
Q 3	28.545	30.878	29.309	29.577	1.189
Q 4	26.713	26.871	29.846	27.810	1.764
Q 5	6.180	9.597	5.786	7.187	2.095

Tab 2. 5 GB & 25 Compute Nodes – Spark-SQL

Query No	Runtime 1	Runtime 2	Runtime 3	Average Runtime	Standard Deviation
Q 1	19.687	17.032	19.765	18.828	1.555
Q 2	54.331	55.106	58.765	56.067	2.368
Q 3	41.85	43.035	47.039	43.974	2.719
Q 4	335.356	350.474	347.083	344.304	7.932
Q 5	69.933	74.026	75.500	73.153	2.884

Tab 3. 5 GB & 50 Compute Nodes – Hive

Query No	Runtime 1	Runtime 2	Runtime 3	Average Runtime	Standard Deviation
Q 1	0.022	0.037	0.028	0.029	0.007
Q 2	14.996	12.729	15.051	14.258	1.325
Q 3	18.562	32.880	16.444	22.628	8.940
Q 4	7.971	7.837	6.760	7.522	0.663
Q 5	5.863	4.868	4.710	5.714	0.625

Tab 4. 5 GB & 50 Compute Nodes – Spark-SQL

Query No	Runtime 1	Runtime 2	Runtime 3	Average Runtime	Standard Deviation
Q 1	498.974	495.054	496.349	496.792	1.997
Q 2	1325.930	1295.256	1298.361	1306.518	16.882
Q 3	617.471	624.561	617.454	619.828	4.098
Q 4	26.348	27.846	27.736	27.310	0.834
Q 5	12.617	15.675	14.953	14.415	1.598

Tab 5. 315 GB & 25 Compute Nodes - Hive

Query No	Runtime 1	Runtime 2	Runtime 3	Average Runtime	Standard Deviation
Q 1	0.133	0.125	0.099	0.119	0.017
Q 2	564.060	560.653	562.862	562.525	1.728
Q 3	310.612	315.235	309.365	311.737	3.092
Q 4	35.636	36.681	36.312	36.209	0.529
Q 5	26.393	24.246	20.843	23.827	2.798

Tab 6. 315 GB & 25 Compute Nodes – Spark-SQL

Query No	Runtime 1	Runtime 2	Runtime 3	Average Runtime	Standard Deviation
Q 1	265.206	256.891	262.354	261.483	4.225
Q 2	1174.535	1163.312	1190.846	1176.231	13.845
Q 3	570.881	589.648	582.432	580.987	9.466
Q 4	20.764	17.752	18.495	19.003	1.569
Q 5	5.695	7.275	6.946	6.638	0.833

Tab 7. 315 GB & 50 Compute Nodes – Hive

Query No	Runtime 1	Runtime 2	Runtime 3	Average Runtime	Standard Deviation
Q 1	0.092	0.086	0.154	0.110	0.037
Q 2	319.336	320.965	317.256	319.185	1.859
Q 3	206.052	200.057	201.795	202.634	3.084
Q 4	31.928	31.680	31.698	31.768	0.318
Q 5	4.401	5.265	4.756	4.807	0.434

Tab 8. 315 GB & 50 Compute Nodes – Spark-SQL

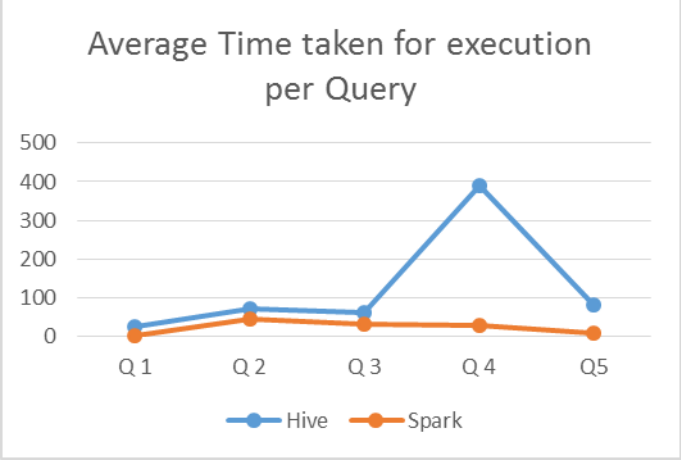

Fig 4. Dataset Size: 5 GB, Compute Nodes: 25

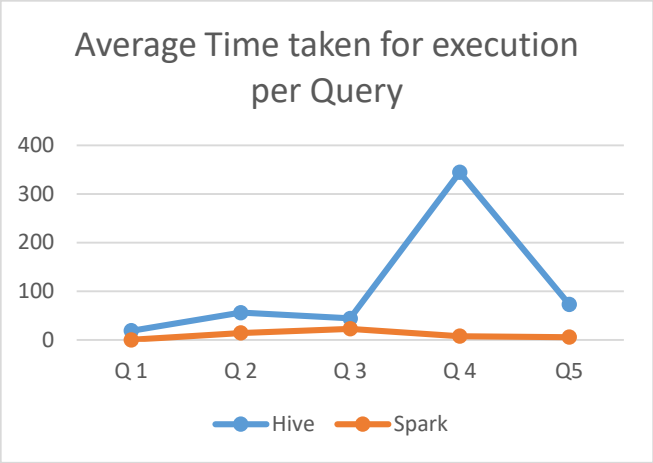

Fig 5. Dataset Size: 5 GB, Compute Nodes: 50

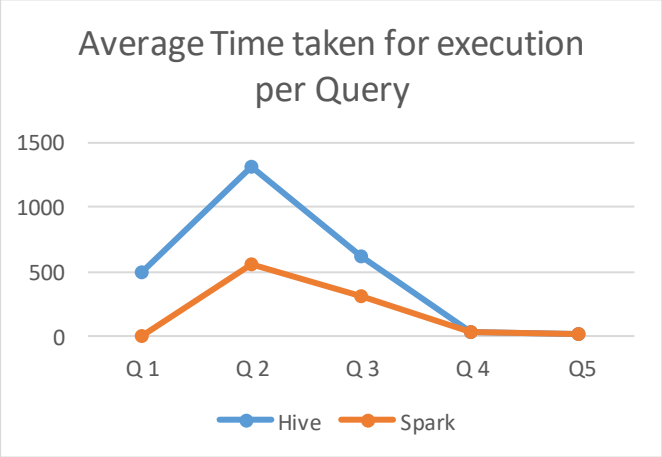

Fig 6. Dataset Size: 315 GB, Compute Nodes: 25

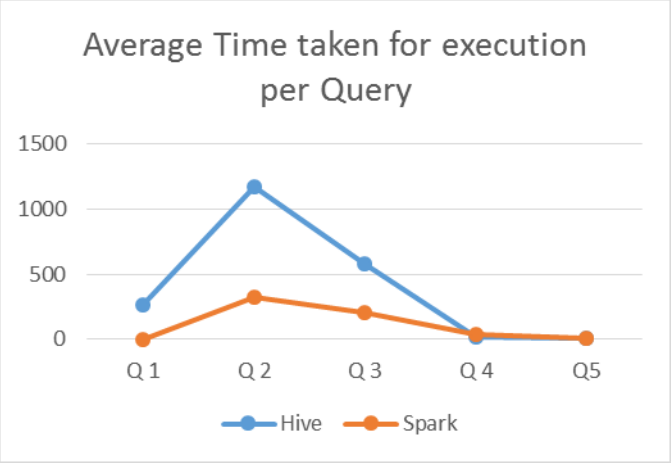

Fig 7. Dataset Size: 315 GB, Computing Nodes used: 50

From the above results we can see that the performance of the queries varied greatly based on different configurations of dataset sizes and computing nodes. The result of the query execution times can be summarized in a tabularized form as below:

Dataset Size	Computing Nodes	Queries faster in Hive than Spark	Queries faster in Spark than Hive
5GB	25	-	Q1, Q2, Q3, Q4, Q5
5GB	50	-	Q1, Q2, Q3, Q4, Q5
315 GB	25	Q4, Q5	Q1, Q2, Q3
315 GB	50	Q4	Q1, Q2, Q3, Q5

From the above results, we can see a general trend that as we increase the number of compute nodes the query execution time reduces except for a few exceptional cases. An increase in the amount data also causes the query execution time to naturally go up for both Spark and Hive platforms. Also we can conclude from the above observations that Spark is approximately 7.8 times faster than Hive when considering a small dataset of 5 GB but for 315 GB of data, Spark only performs a little slower than HIVE for the queries which involve JOIN operations whose reason has been listed in the current Spark SQL documentation.

V. FUTURE SCOPE

As an extension to this experiment in the future we would like to test our experiment with queries run through external scripts, queries that are run for different expressions of null values etc. We would also like to use a standard big data benchmarking tool like BigBench to perform a more standardized benchmarking of the query performance (execution time) for our experiment. Also as an extension to this experiment we can gather resource utilization metrics like CPU utilization, disk input/output, memory utilization and network input/output by using a performance analysis tool like Intel's Performance Analysis Tool (PAT) for both Spark and Hive Platforms.

VI. CONCLUSION

This experiment helps us better understand the Spark and Hive querying platforms and when should we prefer to use one over the other based on their advantages and limitations. We also get a sense of how the two platforms perform for multiple types of queries with different data sizes and compute node configurations. With the above understanding and derived relation, the end user will have a better understanding of the querying framework that best suits his/her needs taking into consideration all the factors described in this paper.

VII. REFERENCES

[1] http://spark.apache.org/

[2] Armbrust, Michael, et al. "Spark sql: Relational data processing in spark." *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015.

[3] https://hive.apache.org/

[4] Thusoo, Ashish, et al. "Hive-a petabyte scale data warehouse using hadoop." *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*. IEEE, 2010.

[5] https://aws.amazon.com/

[6] https://aws.amazon.com/emr/

[7] https://aws.amazon.com/s3/

[8] https://aws.amazon.com/documentation/

[9] https://aws.amazon.com/datasets/google-books-ngrams/?tag=datasets%23keywords%23encyclopedic

[10] http://www.slideshare.net/jetlore/spark-and-shark-lightningfast-analytics-over-hadoop-and-hive-data

[11] https://aws.amazon.com/articles/Elastic-MapReduce/5249664154115844

[12] Li, Xiaopeng, and Wenli Zhou. "Performance Comparison of Hive, Impala and Spark SQL." *Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2015 7th International Conference on*. Vol. 1. IEEE, 2015.

[13] Mavridis, Ilias, and Eleni Karatza. "Log File Analysis in Cloud with Apache Hadoop and Apache Spark." (2015).

[14] Gu, Lei, and Huan Li. "Memory or time: Performance evaluation for iterative operation on hadoop and spark." *High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on*. IEEE, 2013.

[15] http://clds.sdsc.edu/sites/clds.sdsc.edu/files/BigBench_SparkSQL.pdf