International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075, Volume-10 Issue-1, November 2020

A Benchmark for Suitability of Alluxio over Spark

Kanchana Rajaram, Kavietha Haridass

***Abstract:** Big data applications play an important role in real time data processing. Apache Spark is a data processing framework with in-memory data engine that quickly processes large data sets. It can also distribute data processing tasks across multiple computers, either on its own or in tandem with other distributed computing tools. Spark's in-memory processing cannot share data between the applications and hence, the RAM memory will be insufficient for storing petabytes of data. Alluxio is a virtual distributed storage system that leverages memory for data storage and provides faster access to data in different storage systems. Alluxio helps to speed up data intensive Spark applications, with various storage systems. In this work, the performance of applications on Spark as well as Spark running over Alluxio have been studied with respect to several storage formats such as Parquet, ORC, CSV, and JSON; and four types of queries from Star Schema Benchmark (SSB). A benchmark is evolved to suggest the suitability of Spark Alluxio combination for big data applications. It is found that Alluxio is suitable for applications that use databases of size more than 2.6 GB storing data in JSON and CSV formats. Spark is found suitable for applications that use storage formats such as parquet and ORC with database sizes less than 2.6GB.*

***Keywords:** Alluxio, Spark, file formats, benchmark, performance, VDFS.*

I. INTRODUCTION

Apache Spark is an in-memory cluster-computing framework for large scale data processing [1]. It is useful for interactively processing large datasets on a cluster. Spark includes libraries such as Spark SQL, Spark streaming for processing real-time data, MLlib that provides machine learning functionalities and GraphX for providing charts and graphs. Spark SQL provides APIs for data frames that are used to read, write and query the data [2]. It extends the MapReduce model with an abstraction called Resilient Distributed Dataset (RDD) which improves the performance of Spark by caching the data in executors and enables data to be accessed directly from memory on frequent access. Spark can process petabytes of data with low latency and is 100 times faster than Hadoop. It can process both batch and streaming data. However, there are two shortcomings: inadequate memory for large datasets since RDDs are memory bound; loss of cached data in the case of job failures.

Alluxio [3] is a virtual distributed file system which lies between data driven applications like Spark, Presto, Tensorflow, etc. and persistent storage systems like Amazon S3 (Amazon Simple Storage Service), HDFS (Hadoop Distributed File System), Google Cloud Storage, etc. It helps applications to access data regardless of their formats and locations. Alluxio is used as a data access layer for Spark applications. Alluxio framework over Spark decouples storage and computation [4]. When used with Spark, Alluxio persists the data even when the Spark shell is closed. It speeds up I/O performance especially when data is stored remotely from Spark. When the data is locally stored, Spark by itself shows a good performance. Though Spark framework is efficient for big data processing, it is required to identify the need for using Alluxio with Spark. It is important to know whether Alluxio on Spark would always provide better performance. Towards this objective, we have proposed a benchmark to recommend the suitability of Alluxio over Spark with respect to different storage formats and types of queries.

Our contributions in this work are listed below:

- Generating a benchmark dataset in various file formats.
- Observing the performance of different types of queries that use scan, join, aggregate functions, etc.
- Observing the performance of Spark applications with and without Alluxio.

The rest of the paper is organized as follows. The next section elaborates the existing literature. Section III introduces the proposed work. Section IV explains the various experiments that were carried out and Section V discusses the results and provides our recommendation. The last Section concludes our work.

II. LITERATURE

Grandhi et al. [5] conducted performance analysis of queries run on MySQL, queries on Apache Spark with CPU system and queries run on a GPU system. Two datasets were considered, with size 3.7 GB and 4.3 GB respectively. Cache time and response time of queries were compared between these three scenarios. The execution of queries on Spark was faster than MySQL, and execution of queries on GPU was faster than Spark and MySQL. Liet al. [6] introduced Tachyon which was later known as Alluxio. They analyzed its efficiency over in-memory HDFS and showed that Tachyon outperforms HDFS 110X for write throughput and 2X for read throughput. A Virtual Distributed File System (VDFS) [7] was proposed between computation and storage layer. An evaluation framework to evaluate the performance of Alluxio, CephFS, GlusterFS, and HDFS was also evolved in this work. The performance of accessing Ceph storage with and without Alluxio using Spark was compared. In addition, experiments were performed to avoid the ETL process with multi-data center infrastructure by using Alluxio as VDFS.

Revised Manuscript Received on November 01, 2020.
***** Correspondence Author

Kanchana Rajaram, Computer Science and Engineering, Sri Sivasubramaniya Nadar College of Engineering, Chennai, India. Email: rkanch@ssn.edu.in

Kavietha Haridass*, Computer Science and Engineering, Sri Sivasubramaniya Nadar College of Engineering, Chennai, India. Email: hkavietha@gmail.com

It was observed that HDFS outperforms CephFS and GlusterFS. Further, it was verified that Alluxio is slower than HDFS since Alluxio uses APIs to read the data stored in HDFS. Lawrie et al. [8] evaluated the suitability of Alluxio for Hadoop processing frameworks. Alluxio was compared with Hadoop caching and Spark caching. Filter jobs were run to compute the execution time. Performances were compared with respect to Alluxio versus Hadoop caching as well as Alluxio versus Spark caching. The authors suggested that Alluxio is more suitable for implementing a Spark cluster that utilizes in-memory storage. Ivanov and Pergolesi evaluated the impact of columnar file formats on Hadoop processing engines [9]. BigBench (TPCx-BB) benchmark was used for experimentation with dataset size of 1000 GB in ORC and Parquet file formats. Queries were executed using HiveQL and SparkSQL and the execution times were compared. Results showed that ORC performs better with Hive and parquet performs better with Spark. Alonso studied the performance of various HDFS file formats [10]. The queries were run using Hive and using Java MapReduce script. Tests were done on two datasets ADS-B (Automatic Dependent Surveillance - Broadcast) messages collected from Opensky with dataset size 364.16 GB and GitHub log of size 195.46 GB. Different file formats such as CSV, SequenceFile, Avro, RCFile, ORC and Parquet were compared. Moreover, the files were compressed using snappy and gzip. It was concluded that it is better to use compression while working with MapReduce; Avro provides poor performance and ORC shows the best results while using with Hive. It was also emphasized that no particular format is perfect in all situations. It is observed that none of the existing works have evaluated the suitability of Alluxio on Spark

III. PROPOSED WORK

The proposed decoupled architecture of Spark with Alluxio is depicted in Fig. 1. It has three layers, namely, compute, data cache and storage. Compute layer uses Apache Spark to process the data. Data cache layer caches data in Alluxio file system which acts as a VDFS that decouples storage and computation. Storage layer uses Amazon S3 as an object storage service to persist the data in-memory on Alluxio file system. The Star Schema Benchmark (SSB) [11] is designed to measure the performance of transactions in data warehouse applications. SSB contains a fact table LINEORDER and four dimension tables DWDATE, CUSTOMER, SUPPLIER, and PART. The schema is shown in Fig. 2. SSB DBGen tool is used to generate data pertaining to SSB schema. The data can be grown to any size by selecting the appropriate Scale Factor (SF). Data with SF 5, 10 and 15 have been generated for the experiments. The generated data is stored as objects in Amazon S3 buckets and can be accessed through their access points.

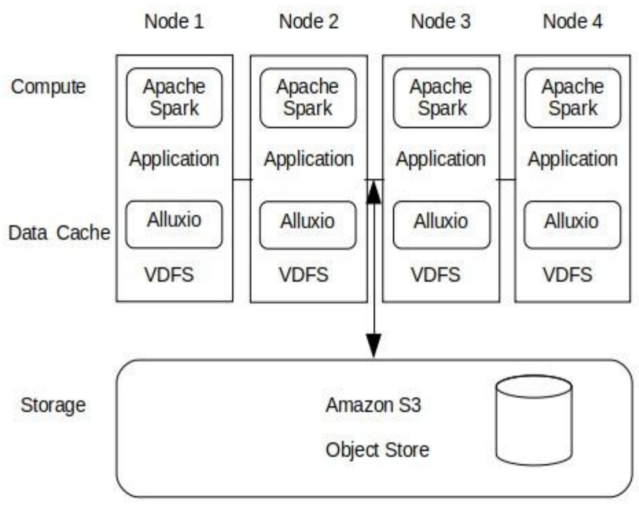

Fig. 1. Decoupled Architecture of Spark and Alluxio

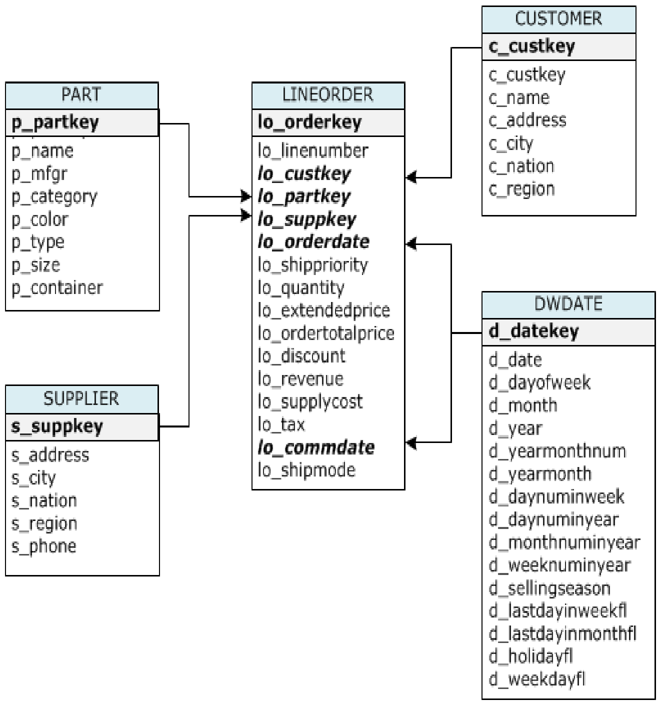

Fig. 2. Star Schema Benchmark schema

The storage efficiency is improved by storing the generated SSB dataset in different file formats such as Parquet, ORC, CSV, and JSON [12]. Parquet and ORC (Optimized Row Columnar) file formats are column oriented and compress the data to minimize the storage space. A file in CSV (Comma-Separated Values) format is simple to parse and supports faster query processing. A file in JSON (JavaScript Object Notation) stores metadata and is the standard for communicating on the web. JSON does not support block compression. Query parses only the metadata when using parquet format whereas the entire content of the file is parsed when using CSV and JSON formats, thus deteriorating the performance. Some of the properties of these file formats are tabulated in Table I.

Table-I: Properties of File Formats

Properties	Parquet	ORC	CSV	JSON
Format	Columnar	Columnar	Row based	Structured
Data Structure	Complex nested data structure	Stripe structure	Comma separated values	Key-value pairs
Compressable	Yes	Yes	Yes	Yes
Splittable	Yes	Yes	Splittable when uncompressed	Splittable when uncompressed
Readable	machine-readable	machine-readable	human-readable	human-readable
Schema Evolution	Yes	Yes	No	No

With SSB dataset, a set of four categories of queries with restrictions upto four dimensions are discussed in [13]. Each category consists of three to four queries with varying selectivity. For our evaluation purpose, the following four queries have been considered, one from each category.

Query 1: *SELECT sum(lo_extendedprice*lo_discount) as revenue*
FROM lineorder, dwdate
WHERE lo_orderdate = d_datekey
AND d_weeknuminyear = 6
AND d_year = 1994
AND lo_discount between 5 and 7
AND lo_quantity between 26 and 35
Query 1 lists the revenues by applying a range of discounts in a given product order, for a quantity interval shipped in a given year. It involves two restrictions on the dimension table DWDATE and three restrictions on the fact table LINEORDER.

Query 2: *SELECT sum(lo_revenue), d_year, p_brand1*
FROM lineorder, dwdate, part, supplier
WHERE lo_orderdate = d_datekey
AND lo_partkey = p_partkey
AND lo_suppkey = s_suppkey
AND p_brand1= 'MFGR#2239'
AND s_region = 'EUROPE'
GROUP BY d_year, p_brand1
ORDER BY d_year, p_brand1;
Query 2 lists revenues for certain product classes and suppliers in a certain region, grouped by restrictive product classes and all years of orders. It involves one restriction on two dimension tables PART and SUPPLIER and three restrictions on fact table LINEORDER with group by restrictions on two dimension tables.

Query 3: *SELECT c_city, s_city, d_year, sum(lo_revenue) AS revenue*
FROM customer, lineorder, supplier, dwdate
WHERE lo_custkey = c_custkey
AND lo_suppkey = s_suppkey
AND lo_orderdate = d_datekey
AND (c_city='UNITED KI1'
OR c_city='UNITED KI5')
AND (s_city='UNITED KI1'
OR s_city='UNITED KI5')
AND d_yearmonth = 'Dec1997'
GROUP BY c_city, s_city, d_year
ORDER BY d_year asc, revenue desc;
Query 3 retrieves total revenue for lineorder transactions within a region restricting to two cities in a certain time period, grouped by customer nation, supplier nation and year. It involves one restriction on three dimension tables CUSTOMER, SUPPLIER and DWDATE and three restrictions on fact table LINEORDER with group by restrictions on three dimensional tables.
Query 4: *SELECT d_year, s_nation, p_category,*
sum(lo_revenue - lo_supplycost) as profit
FROM dwdate, customer, supplier, part, lineorder
WHERE lo_custkey = c_custkey
AND lo_suppkey = s_suppkey
AND lo_partkey = p_partkey
AND lo_orderdate = d_datekey
AND c_region = 'AMERICA'
AND s_region = 'AMERICA'
AND (d_year = 1997 or d_year = 1998)
AND (p_mfgr = 'MFGR#1'
OR p_mfgr = 'MFGR#2')
GROUP BY d_year, s_nation, p_category
ORDER BY d_year, s_nation, p_category
Query 4 retrieves aggregate profit grouped by year and nation. It has restrictions on four dimension tables DWDATE, CUSTOMER, SUPPLIER and PART and four restrictions on fact table LINEORDER with group by restrictions on the four dimension tables. The above queries were run by a Spark application by accessing data remotely from Amazon S3 storage. The performance of Spark with its own in-memory storage as well as Spark on Alluxio, are analyzed by executing these four benchmark queries on a database of different sizes by storing them in four different file formats. Queries were run with Spark, accessing data from Amazon S3 and caching the data in its in-memory. The same queries were run by accessing data from Amazon S3 and storing data in Alluxio in-memory. Initially, the query processing time was higher, as the application needs to mount the Amazon S3 storage to file system. On subsequent queries, the data was accessed directly from memory and performed its computation. The details of the experiments are discussed in the next section.

IV. EXPERIMENTATION

A testbed consisting of a multi node cluster is setup with 4 nodes, 1 master and 3 worker nodes. Master node has intel i9, 12 core processor using 64 GB of RAM. Worker node has intel i7, 4 core processor using 16 GB of RAM. With these nodes Hadoop is installed as master slave architecture and Spark runs on the Hadoop cluster with underlying HDFS.

Experiment is done on Ubuntu platform in which Spark 2.4.3 is deployed and Alluxio 2.1.9 compatible to Spark version is used. The SSB dataset size was scaled up by 5, 10 and 15 and for each scale up, the data was stored in four different file formats such as parquet, ORC, CSV and JSON. Four benchmark queries were run on Spark cluster. The experiments have been carried out with and without having Alluxio as virtual storage, to understand the performance of Spark and Spark on Alluxio. The first two experiments are based on varying file formats and varying the database size. Alluxio UI is useful to measure the data mounted on Alluxio and its distribution across nodes while the performance of the application is measured through Spark UI. The resulting sizes of the database by scaling up the SSB dataset and the corresponding file format and file size are tabulated in Table III. The file size of a database varies with different formats due to its compression nature as mentioned in Table I.

Table-III: Database size and File sizes corresponding to different File Formats

Scale factor	Database size (GB)	File format	File size (GB)
SF5	2.6	Parquet	1.1
		CSV	2.6
		JSON	10.8
		ORC	1.1
SF10	6.8	Parquet	2.1
		CSV	6.8
		JSON	21.5
		ORC	2.2
SF15	9.4	Parquet	3.2
		CSV	9.4
		JSON	32.3
		ORC	3.3

Experiment 1: Performance of four benchmark queries are measured on fixed size database stored in four different file formats such as Parquet, ORC, CSV, and JSON. The experiments are conducted for datasets generated with scale factors of 5, 10 and 15 and results are depicted as graphs in Fig. 3, 4, and 5.

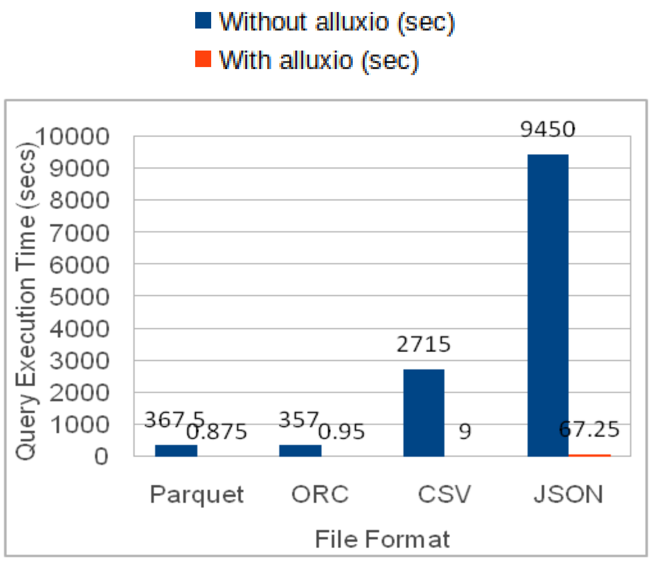

Fig. 3. Performance comparison for database of SF 5

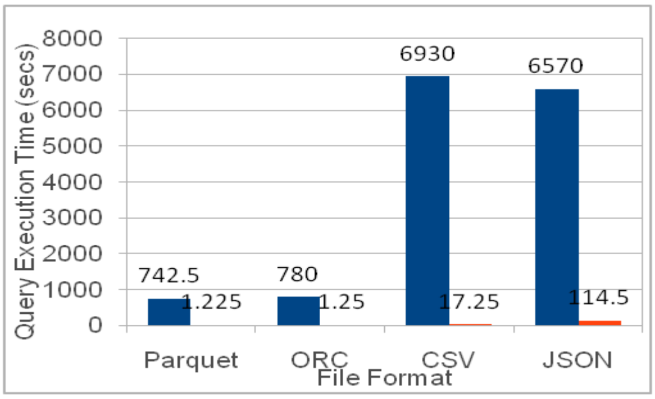

Fig. 4. Performance comparison for database of SF 10

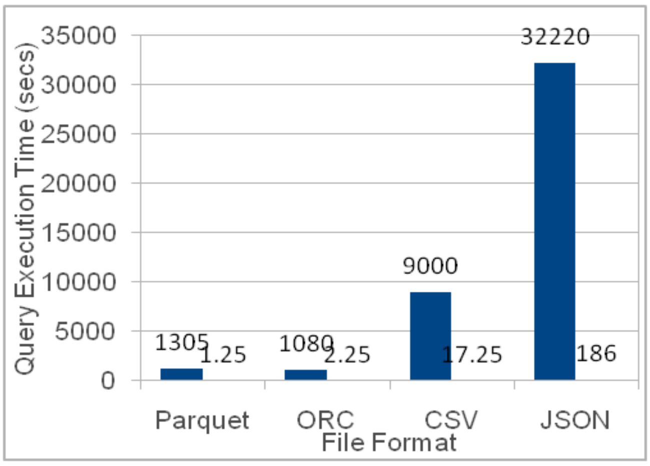

Fig. 5. Performance comparison for database of SF 15

In this experiment, it is observed that the data stored in parquet and ORC formats provides better performance with Spark as well as Spark on Alluxio.

Experiment 2: Performance of four benchmark queries are analyzed for different database loads of SF 5, 10, and 15 with the file stored in parquet format.

Similar tests have been conducted for the other three file formats such as ORC, CSV, and JSON. The execution time taken by the queries are depicted as graphs as shown in Fig. 6, 7, 8, and 9.

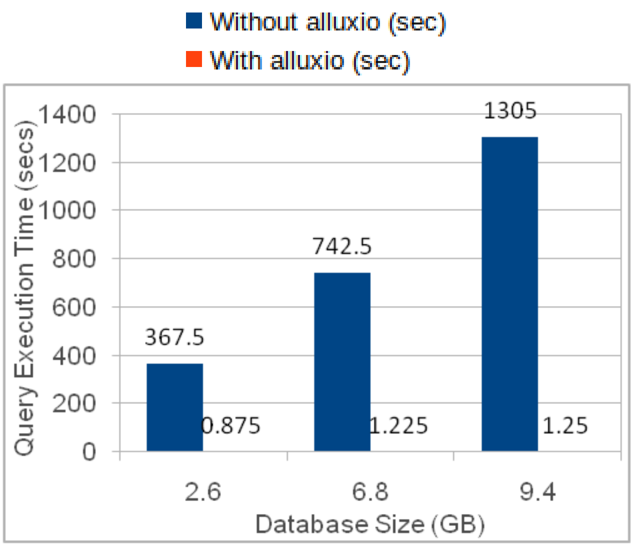

Fig. 6. Performance comparison for Parquet format

It is observed that without using Alluxio the query execution time increases from 375 seconds to 25650 seconds as the database size increases. When Alluxio is used, the increase in query execution time is not more than 75 seconds.

V. RESULTS AND DISCUSSION

As a result of experiments, the average query processing time for database SFs 5, 10 and 15 when the data is stored in different file formats, are tabulated in Table IV.

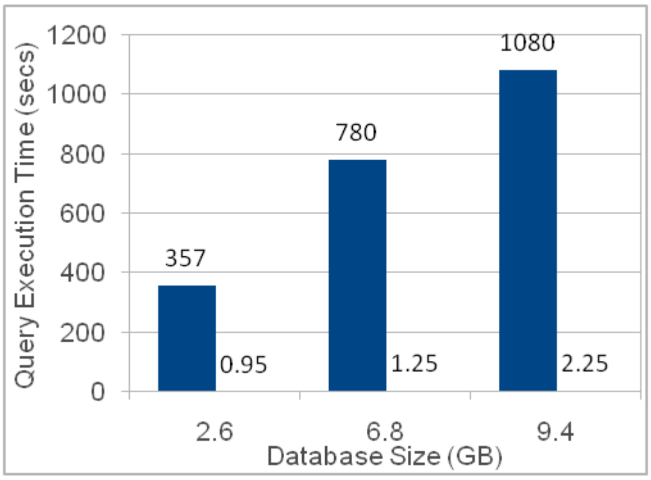

Fig. 7. Performance comparison for ORC format

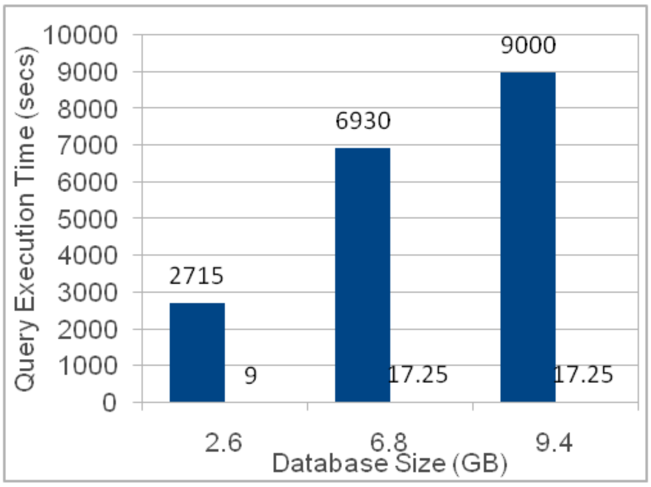

Fig. 8. Performance comparison for CSV format

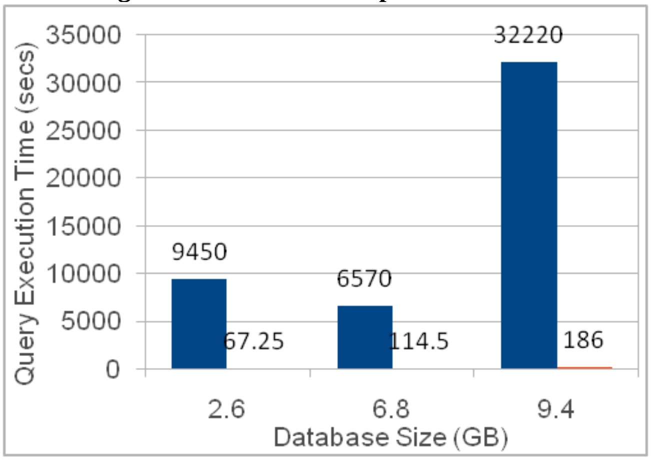

Fig. 9. Performance comparison for JSON format

Table-IV: Spark versus Spark with Alluxio

File format	Spark	Spark on Alluxio	Recommendation
Parquet	13.0 mins	1.0 sec	Spark
ORC	12.0 mins	1.5 secs	Spark
CSV	1.7 hrs	14.5 secs	Alluxio when dataset is >2.6 GB
JSON	4.5 hrs	2.0 mins	Alluxio

When the file is stored in Parquet format, it highly compresses the data to 70 percentage. With Spark cluster, query execution time for parquet reached a maximum of 27 minutes for 9.8 GB data. When Alluxio is used for storage, the query executed in 2 seconds for the same 9.8 GB of data with 99% improvement in performance. ORC format also compresses the data and shows on-par performance to parquet format. Though parquet and ORC provides similar performances, their functional differences need to be considered. Vectorized query execution [14] is a feature that greatly reduces CPU usage for query operations such as scan, filter, aggregates and joins. Spark has a vectorized parquet reader but does not provide one for ORC. Though Alluxio provides faster query execution time as compared to Spark's in-memory storage, Spark is recommended for time insensitive queries, since, CPU usage is found minimal with files stored in parquet format.

The data is not compressed for files stored in CSV format. With CSV file format, query performance decreases as database size increases. With Alluxio, the performance is improved by around 90 percentage. Without Alluxio, query processing takes 1 to 2 hours. For 2.6 GB of data, the query execution time using Alluxio is 300 times lesser. Similarly, with database sizes of 6.8 GB and 9.4 GB, query execution times using Alluxio is 400 times and 500 times lesser respectively. Therefore, Alluxio proves to be a better choice for databases of size more than 2.6 GB. For database size more than 2.6 GB, Spark may not be efficient because of its limited in-memory. JSON format increases the actual file size by 68 percent since the data is stored along with metadata. Since data becomes heavy, the data transfer rate increases as the database size scales up. Without Alluxio, it takes hours for query processing whereas Alluxio improves the performance by 99 percentage. It is observed that when Spark takes 8 hours to process a query for 9.8 GB data, Alluxio fetches the result around 3 minutes. Efficient usage of Alluxio can be observed when processing JSON format files, since even with 2.6 GB data, execution time is higher by 9000 seconds than the data stored in other formats. Alluxio processes the query in 67 seconds as shown in Fig. 3. Hence, Alluxio is preferred to query the files stored in JSON format.

VI. CONCLUSION

High performance and optimized storage are the requirements of any big data application. With many big data technologies available, it is important to know their suitability for an efficient big data framework. Spark is always efficient when the storage is local.

When data is remote, the need for Spark with Alluxio has been analyzed. Spark is efficient for file formats parquet and ORC for all database sizes 2.6, 6.8 and 9.4 GB. Deployment of Alluxio is an overhead since, the faster query execution time is compensated with higher mounting time of Alluxio file system. For data stored in CSV format, execution time is 45 minutes for 2.6 GB. Hence, Alluxio is preferred for database size more than 2.6 GB and Spark itself gives better performance for lesser database sizes. Average query execution time of data stored in JSON format is 4.5 hours on Spark and just 2 minutes with Alluxio. Though Alluxio is useful for data storage making computation more powerful, it is observed to be an overhead in some scenarios. With large dataset where the queries are I/O heavy, Alluxio helps to improve the performance of the query. Alluxio provides better performance in high computational applications by effectively utilizing the computational potential. Hence, it is concluded that Alluxio is suitable for big data applications that use databases of size more than 2.6 GB storing data in JSON and CSV formats. Spark is preferred for applications that use storage formats such as parquet and ORC with database sizes less than 2.6GB.

ACKNOWLEDGEMENT

This work is supported by the Biotechnology Industry Research Assistance Council (BIRAC), Department of Bio Technology, Government of India under the scheme of Grand Challenges India - Immunization Data Innovating for Action as a part of the project titled "Nagarik Rog Pratirakshak: Unified Smart Immunization Coverage Monitoring and Analysis (NRP-UniSICMA)". We thank Prof. S. Selvakumar, Director and Principal Investigator, IIIT Una (HP) for the consistent support of the work.

REFERENCES

1. A. da Silva Veith, M. D. de Assuncao , "*Apache Spark*", Springer International Publishing, Cham, 2018 pp.1-5.
2. Z. Han, Y. Zhang, "*Spark: A big data processing platform based on memory computing*" Seventh International Symposium on Parallel Architectures, Algorithms and Programming, 2015, pp. 172-176.
3. Alluxio, Inc. (2018) "*Alluxio Overview*", White paper, Available: https://www.alluxio.io/resources/whitepapers/alluxio-overview/.
4. P. Xuan, "*Accelerating Big Data Analytics on Traditional High-Performance Computing Systems Using Two-Level Storage*", All Dissertations, 2318, Clemson University, 2016.
5. B. Grandhi, S. Chickerur, M. S. Patil, "*Performance Analysis of MySQL, Apache Spark on CPU and GPU*", 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology, Bangalore, 2018, pp. 1494-1499.
6. H. Li, A. Ghodsi, M. Zaharia, S. Shenker and I. Stoica, "*Tachyon: Reliable, memory speed storage for cluster computing frameworks*", In Proceedings of the ACM Symposium on Cloud Computing, 2014, pp. 1-15.
7. H. Li, "*Alluxio: A virtual distributed file system*", Doctoral dissertation, UC Berkeley, 2018.
8. C. Lawrie, P. Kothuri, "*Evaluation of the Suitability of Alluxio for Hadoop Processing Frameworks*", CERN summer student program, 2016.
9. T. Ivanov, M. Pergolesi, "*The impact of columnar file formats on SQL-on-hadoop engine performance: A study on ORC and Parquet*", Concurrency and Computation: Practice and Experience, volume 32(5), 2020, pp. 1-31.
10. A. Alonso Isla, "*HDFS File Formats: Study and Performance Comparison*", Master's thesis, Universidad de Valladolid, 2018.
11. P.E. O'Neil, E. J. O'Neil, X. Chen, "*The star schema benchmark (SSB)*", 2007
12. A. Trivedi, P. Stuedi, J. Pfefferle, A. Schuepbach, B. Metzler, "*Albis: High-performance file format for big data systems*", USENIX Annual Technical Conference, 2018, pp. 615-630.
13. Sanchez J, "*A Review of Star Schema Benchmark*", arXiv preprint, East Carolina University, 2016, Available: http://arxiv.org/abs/1606.00295.
14. Eric N. Hanson, "*Vectorized Query Execution*", Jul 29, 2017, Available: https://cwiki.apache.org/confluence/display/Hive/Vectorized+Query+Execution.

AUTHORS PROFILE

Kanchana Rajaram is an Associate Professor at the Department of Computer Science and Engineering, Sri Sivasubramaniya Nadar College of Engineering, Chennai. She obtained PhD in Computer Science and Engineering from Anna University, Chennai. She has 30 years of teaching experience and her research interests are Web Services, Big Data Analytics and Distributed Computing. She has to her credit of publishing 31 research papers in journals and conferences. Currently she is the Principal Investigator of the sponsored collaborative project worth Rs. 106 lakh funded by Biotechnology Industry Research Assistance Council (BIRAC), Department of Bio Technology, Government of India. She is a member of IEEE, ACM, and ISTE.

Kavietha Haridass has been working as a Junior Research Fellow under the BIRAC (Biotechnology Industry Research Assistance Council) funded research project titled "Nagarik Rog Pratirakshak: Unified Smart Immunization Coverage Monitoring and Analysis (NRP UniSICMA)". She has a post graduate degree in Computer Science and Engineering from Sri Sivasubramaniya Nadar College of Engineering affiliated to Anna University, Chennai. She obtained her undergraduate degree from Jeppiaar SRR Engineering College affiliated to Anna University, Chennai. Her research interests include Big data analytics, Big data technologies, Data Visualization, Web technologies and Storage management. Earlier, she has a teaching experience of 8 months.