Available online www.ejaet.com

European Journal of Advances in Engineering and Technology, 2018, 5(12):1137-1143

Research Article ISSN: 2394 - 658X

Distributed Data Processing with Spark vs. Hadoop MapReduce

Ravi Shankar Koppula

*Ravikoppula100@gmail.com

ABSTRACT

Apache Spark is a distributed data processing system that offers superiority over the conventional Hadoop MapReduce approach in terms of speed, feasibility, and scalability. Spark operates on distributed data and is specifically designed to perform computations in-memory, leading to significantly faster data processing and analysis. Moreover, Spark not only supports batch processing but also excels in near real-time processing, making it a versatile and powerful tool for a wide range of data processing needs. One of the key advantages of Spark is its ability to address the limitations and challenges posed by the MapReduce model. Spark has emerged as a market leader by filling the gaps left by MapReduce and addressing the latency issues caused by its programming model. Compared to MapReduce, Spark offers faster execution, easier usage of shared variables with iterations, and shorter codes. This enables developers and data scientists to write more concise and efficient code, resulting in increased productivity and reduced development time. Additionally, Spark is particularly advantageous for iterative code and complex computational tasks. Its ability to cache data in memory allows for the efficient reuse of intermediate results, further boosting performance. With Spark, organizations can process and analyze large volumes of data with ease, enabling them to extract valuable insights and make data-driven decisions in a timely manner. In conclusion, Apache Spark is a cutting-edge distributed data processing system that outperforms MapReduce in various aspects of distributed data processing. Its speed, scalability, and support for both batch and near real-time processing make it an indispensable tool for organizations dealing with large-scale data processing and analysis. By leveraging the power of Spark, businesses can unlock the full potential of their data and gain a competitive edge in today's data-driven world.

Key words: Spark, Hadoop, MapReduce, Spark use cases, Hadoop use cases

---

INTRODUCTION

Due to its remarkable qualities, the industry has shown increasing interest in transitioning from the previous MapReduce approach to Apache Spark for both batch processing and stream processing. This analysis aims to explore the distinctions between Apache Spark and the conventional big data storage model facilitated by Hadoop. It will delve into the underlying concepts of Spark, showcase its superiority over Hadoop, discuss the components utilized in Spark, and illustrate why replacing the Hadoop framework with Apache Spark leads to enhanced speed and feasibility in the data processing model. Apache Spark has emerged as a prominent hub for storing and processing data in recent years. The reason for its widespread adoption lies in its in-memory storage, speed of processing, and scalability compared to previous solutions. Spark's ability to quickly process data is attributed to its approach of performing computations in-memory and its efficiency in executing tasks. Furthermore, Apache Spark not only supports the batch processing model, but also excels in near real-time processing, surpassing its predecessors. The foundation of Hadoop MapReduce has served as an inspiration for Spark to revolutionize the big data landscape and address the latency issues caused by the programming model of the MapReduce processing engine. Additionally, the industry has recognized that MapReduce lacks the comprehensive set of libraries required for efficient distributed data processing. This realization led to the emergence of Apache Spark, which fills the gaps left by MapReduce. Spark originated as an academic project at

UC Berkeley in 2009 and later became a part of the Apache Incubator in 2013. It achieved the status of a top-level project in 2014, thereby transforming the big data industry from relying on Hadoop MapReduce to embracing Spark. The widespread acceptance and adoption of the Hadoop framework have established MapReduce as a dominant engine for big data processing. Spark, the next big thing in big data, is specifically designed to operate on distributed data and is currently emerging as a market leader in the industry. The fundamental question arises: why Spark and how does it outperform Hadoop for distributed data processing?

OVERVIEW OF DISTRIBUTED DATA PROCESSING

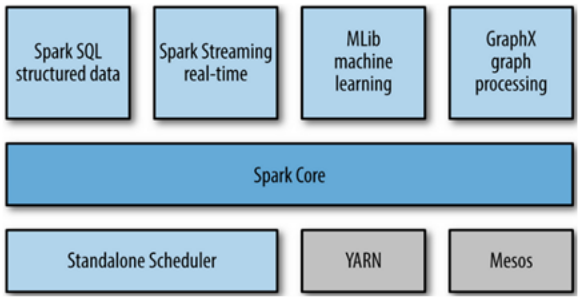
Most of the distributed data processing systems out there rely on a technique called execution framework. An execution framework is a system with a number of well-structured components collaborating for a common goal. In data analysis, it orchestrates a number of computing nodes that are running identical tasks on different data subsets. A system based on an execution framework needs to focus on a number of seemingly checked primes that are assessment and data storage. Assessment is the action of knowing a computing node what it has to do, while data storage is the set of activities employed by that place to persistently store data. If the assessing and data conversing stages take a lot of time than the conclusion implemented by the analytics process of knowing, a large amount of data is lost.

Overview of Distributed Data Processing. Modern data analysis is performed by software that is distributed on a number of interconnected small computers, running in parallel on various data subsets and aggregating results towards the completion of the main task. In the big data context (fragmentary data that is too large to fit a single server's memory), the execution time or, in broader terms, efficiency has a decisive role. The key aspect in distributed data analytics is represented by the system used for executing the analysis. The performance, comprehending factors like execution time, data handling, scalability and usage simplicity, strongly depends on the execution system's architecture.[1]

SPARK: FEATURES AND BENEFITS

Spark is already widely used because of its features. Providing an in-memory execution model that is an order of magnitude faster than Hadoop MapReduce, Spark has made it possible to operate batch and stream processing pipelines in the same stack. Spark makes it easier to provide different types of data analysis on the same data using simple transformations in a core interface called RDDs or DataFrames. In addition, it moves some data processing pipelines that are complex to maintain from frameworks other than Spark to keep data in memory and process it faster. Thus, MapReduce and other representative processing frameworks that have become less expressive over time are beginning to be replaced by Spark.

The Spark stack has two basic components: the Spark Core, which covers the basic functions of the system and is associated with other Spark components, such as SQLSpark, Spark Streaming, GraphX, and Spark MLlib. In addition to the in-memory execution, Spark provides fault-tolerance features through intermediate results and provides derivatives for the classic Hadoop MapReduce operators in a customized RDD interface.[2]



HADOOP MAPREDUCE: FEATURES AND BENEFITS

Hadoop MapReduce is a distributed computing model featuring a central master and several slaves or workers. The programming model is based on a theoretical principle with its functions, map and reduce, which takes physical records and assembles key values up. The mapper (or map job) processes these key values individually, and each record is determined out as a key value matched set. MapReduce computes, shifts keys around and practically accepts these outputs as input to reducer (or reduce job) functions. Hadoop MapReduce is a model and programming interface for effective batch processing of big data, and it has drastically changed the worth

perceived from several Hadoop tasks. The model implements common summarization of statistical parameters facilitating efficient queries for ad-hoc search operations. Frameworks such as Apache Mahout, which implement algorithms based on mathematical functions, can be described as Hadoop MapReduce computing structures. Furthermore, the power of MapReduce is clear and it's also known to be particularly slow for non-disk bound operations involving clusters that don't have hitherto high percentages of idle resources.

Traditional batch data processing involves the execution of a sequence of phases. Each phase processes the output of the preceding phase and generally reads/writes data to the Hadoop Distributed File System (HDFS). MapReduce is designed to dependably process big data using these phases. In fact, because of the requirements from Google Search (or Nutch Crawler), it features optimal disk read and write operations and has been broadly adopted at big data organizations worldwide. Even though some big data processing jobs might simply be executed by other schemes like, for example, Hadoop HDFS (by writing a C++ application), Google Search workloads can be implemented most effectively using MapReduce.[3]

COMPARISON OF SPARK AND HADOOP MAPREDUCE

The recent work on the Spark system has been focused on providing a scalable and efficient distributed scheduler that provides key performance benefits for applications that are important to users of more sophisticated and complex data processing tools. These applications include iterative data processing (such as machine learning algorithms), interactive queries, complex data pipelines, and real-time stream processing. Spark is designed to support these applications by providing a general-purpose programming model that is implemented on a DAG-based distributed scheduler. Spark's API includes the usual map and reduce functions that come with a distributed computing system. Many libraries on top of Spark have benefited from both its expressivity as well as its higher-level features. Like many non-Hadoop data processing systems, Spark is designed to support additional storage systems (such as key-value stores, relational databases, web services, and distributed machines or libraries) and to interwork with them, which is difficult for MapReduce. An increasing number of data structures, such as geometric and graph data, are important to many computing applications, and although classical MapReduce can be used to manipulate these datasets, their performance is not competitive.[4]

Hadoop MapReduce's ability to handle petabyte-scale datasets has enabled a number of new, large-scale computing tools and frameworks, several of which have MapReduce components. However, MapReduce was designed to handle batch processing jobs in a reliable and scalable way and was not designed to address interactive and iterative applications. Thus, MapReduce has limited throughput for these iterative workloads. It reads and writes only to and from HDFS, making it challenging to build real-time streaming systems and interact with other systems and data stores directly.

MapReduce	SPARK
Mainly Restricted For Java Developers	Java , Scala, python, R, SQL closure
Boiler Plate Coding	Conciseness
No Interactive Shell	REPL(Read evaluate print loop)
Disk Based, Performance is Slow	Memory based only
Only For Batch Processing	Batch as well as interactive processing
Not Optimized For Iterative Algorithm	Best for iterative algorithms, No Graph
No Graph Processing	Graph processing is supported

PERFORMANCE COMPARISON

The data locality issue was focused on improved scheduling algorithms for the cluster resource management, especially Tachyon which is a quick SSA Datacenter (SQL Server Analysis Services in Data mode) of the data produced during data processing. Among others, from the performance comparison of Stratos and Tachyon, repeatable and practical performances can be obtained with graceful degradation thresholds and same performance scores, maximum number of faults, simulations of stored data volume, and stored data size of Tachyon. Comparing the performance scores and obtained results of using mathematical equations such as

Poisson and Exponential, the greedy approximation, performance issues in a distributed file system, differences in the maintenance and goal of the work, the techniques, and the parameters can be summarized. There are significant performance gains from fault tolerance mechanisms especially under high computational loads, while read has a large performance impact. In the view of application utilization, some issues to be discussed include industry's map services, plasma physics, large message transport, query system, graph algorithms, email spam classification, and High Energy Physics (HEP) application and the application scalability.[5]

From the benchmark, the impacts of memory usage and serialized data format of Spark can be observed. Comparing and contrasting with the Hadoop MapReduce framework, it can be concluded why and how Spark overcomes the weaknesses of Hadoop MapReduce in big data processing. In the benchmark, it was used a cluster of 240 quad-core beefy machines (each machine includes 96 GB RAM, eight 7200 RPM disks, 1 Gb/s switches) with 8-12 times more RAM per a machine than the machines that were equipped in a typical Hadoop cluster. With financial resources, this cluster can be built using general purpose reliable, available, and scalable commodity hardware, and is more stable as well as practical and cost effective than running on Amazon EC2.

	Hadoop MR Record	Spark Record	Spark 1 PB
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400 physical	6592 virtualized	6080 virtualized
Cluster disk throughput	3150 GB/S (est.)	618 GB/S	570 GB/S
Sort Benchmark Daytona Rules	Yes	Yes	No
Network	dedicated data center, 10Gbps	virtualized (EC2), 10Gbps network	virtualized (EC2), 10Gbps network
Sort rate	**1.42 TB/min**	**4.27 TB/min**	**4.27 TB/min**
Sort rate/node	**0.67 GB/min**	**20.7 GB/min**	**22.5 GB/min**

SCALABILITY COMPARISON

First, a well-defined question is necessary. Will performance of both systems monotonically increase with data set size? Will one or the other one outgrow either the competition or the supplied infrastructural resources? The most important question is how applicable are measurements conducted with small-scale workloads to systems designed for for processing petabytes of data. In essence, this is a good question how realistic are benchmarks in the current ecosystem of big data processing. There are of course more detailed aspects of behavior such comparison would shed light on. Such aspects are whether the time of stage initialization will change with data set size, and e.g. how the peak performance of each is determined. After setting a question, and selecting workloads for experiments, the next step is to ensure you're testing the right system.[6]

The fact that users can use extra hardware to get around bigger workloads by additional machines doesn't mean that it's easy to scale a system by an order of magnitude. Typically, many bottlenecks need to be identified and optimized, and a clunky part of the system (or its first iteration) can be a main inhibitor. This is also true about frameworks used for big data processing, like Hadoop MapReduce or Apache Spark. Both have managed to process data sets of staggering volume in reasonable times. Still, because Spark is touted as faster than Hadoop MapReduce, also scalability of Spark as compared to MapReduce. While there are a few comparisons of the scale of single MapReduce over Spark application, distinguishing the processing framework from the end-to-end system and allowing evaluating several points on a trade-off spectrum, there's no comprehensive research about how each system under identical conditions behaves and affect the processing, even though it is such an essential parameter.

FAULT TOLERANCE COMPARISON

Thus, every so often, every mapper, which is processing a part of the partition of the input data, emits interim results. Every output of the "map" op has its pair (I will explain what it is in a bit) and hence get processed by the reducer that receives it, so it does not accumulate anywhere. The records are committed to a file, and at the end of the commit, an index (called an index) is created so the records could be grouped and processed in parallel. Spark, on the other hand, does not restart from the lost partitions. And, if you never played with Spark, the immediate question should be "how is it possible?" Well, as with many other things in Spark, it is the result of an uncontested fact: Spark's in memory processing is faster. With its almost real-time processing speed, and

its extremely granular control over the cached data, Spark has effectively made the necessity of frequent check points obsolete.

The basic unit of fault tolerance in the data processing world is the data. That is, the work must be redone only for the lost data, and not for the data that had been previously processed. Both the projects operate on the same principle. How do they achieve this? Hadoop MapReduce and Spark both adhere to the standard checkpointing of data processing frameworks. They save the results of the transformations at the predetermined intervals, and restart the operation from the last "check point". For MapReduce, this is typically once for each mapper. The reason for it is the same as why it is normally <key, value> for the mapper and the reducer: Hadoop uses the Java serialization mechanism, which has notoriously bad performance record for anything other than primitives and its close relatives, such as String and long.[7]

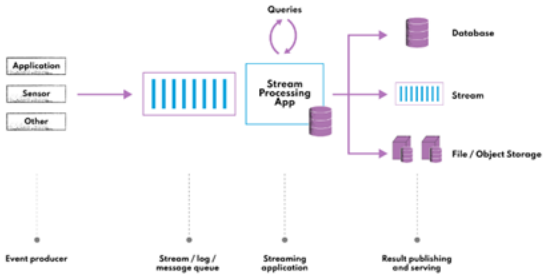
EASE OF USE COMPARISON

In a Big Data environment, the amount of processing resources available to a single application is typically not limited. For that reason, well-established high-level computing tools, such as MATLAB, R, and Excel, have seen limited adoption in distributed settings. With the goal of balancing development productivity and processing efficiency, we find that Spark is a compelling tool for implementing distributed data processing tasks. Although it is common to discuss Spark with respect to Hadoop MapReduce, Spark and Hadoop have distinct designs that cater to distinct use cases. Our analysis has shown that Spark favors implementation efficiency, reduced running production overhead, ease of use, and operational speed over scaling capabilities. Whether the trade-offs of Spark are appropriate for particular Big Data applications are a function of many other factors, such as the scale of input data and the amount of data we wish to output.

Ease of use is a qualitative metric that is typically difficult to address quantitatively. For this analysis, we present a simple qualitative measure of ease of use - the number of steps required to implement a common analysis task: word count. The argument can be made that because the implementation of many common applications in Spark requires fewer steps, the framework is easier to use. However, frequently the implementation of one task requires the implementation of many other auxiliary tasks, and this is the case in Spark. The imperative programming style that Spark follows requires more explicit coordination than the declarative MapReduce style. Finally, when additional resources, such as unit tests, are taken into consideration, implementations in Spark, in general, required more steps. The common paradigm of using SQL for the implementation of common analytics tasks did reduce the number of steps in many Spark fields, but using explicitly required more than a SQL only implementation in MapReduce.[8]

USE CASES FOR SPARK

Here are some examples of typical use cases for Apache Spark:



1. Machine learning: Use Spark to implement MLlib, an effective way of filling the gap with stand-alone model training. Or offload online learning from Storm to Spark as online learning would require faster iterations supported by in-memory modules.
2. Query optimization via Hive-on-Spark or Shark, which converts Hive queries into a comprehensive directed acyclic graph of stages in Spark — similar to Spark HiveContext, mostly excluding physical constructs and can be problematic where the resources are unavailable in your system.

3. Event-driven programming: Consider utilizing Spark for data pattern detection owing to its capability as Spark Streaming builds upon Spark's low-latency micro-batch processing and generalized execution framework (to derive a number of machine learning algorithms from live streaming data).
4. Extract, Transform, Load (ETL) using Spark SQL: Another of the great advantages of Spark is its fantastic performance for data analytics. With Spark's memory computational model, it realizes trade-offs over disk storage and computation. Since it is disk I/O-bound when executing SQL queries over distributed data using PostgreSQL or MySQL, it wouldn't seem practical to analyze data at a large scale.[9]

USE CASES FOR HADOOP MAPREDUCE

Evaluating Spark's performance is more complex than the factors presented in this article. In general, when using Apache Spark, you should aim to fit your dataset in memory. Utilize Spark's lazy evaluation and tight integration with Apache Hadoop (through Hadoop FileSystem APIs) to evaluate the amount of data that moves through the network. In addition, prior to choosing Spark, evaluate its performance with your organization's operational applications. Spark's faster in-memory processing speed by itself is not reason enough to justify lowering the performance of another application that is operational within your enterprise. For enterprise-scale operational applications, you may even want to implement and compare performance between a combined Apache Hadoop and Apache Spark system and an Apache Hadoop-only system.

There are several use-cases that should steer you towards using Hadoop MapReduce over a distributed in-memory processing system such as Apache Spark. The two obvious ones are if your dataset is too large to fit within the memory of your cluster and if you are running any Hadoop-based application on your cluster. Even then, you will have to compare the performance of Apache Spark over HDFS with MapReduce when implementing other Hadoop applications such as HBase and Cassandra (or Hadoop external tables in Apache Hive) on your cluster.

SPARK VS. HADOOP MAPREDUCE: WHICH TO CHOOSE?

In this feature comparison, we will address the task of distributed data processing using both Apache Spark and Hadoop MapReduce. Among the open-source community, Spark has received attention as being a fast and general engine for large-scale data processing. To enable faster computation, unlike MapReduce which writes intermediate results to distributed file systems such as HDFS, Spark allows users to persist intermediate results in-memory as RDD (Resilient Distributed Datasets) with flexibility about their persistence strategies. In fact, our study found that the nested loop algorithm in Spark outperformed Hadoop MapReduce's merge-join algorithm for small join relations in-memory join cases with respect to elapsed time. Then, we compare the performance of two different join types between the two platforms. Runtime evaluation results show that nested loop join in Spark outperforms MapReduce's merge-join algorithms when the input data size is small.

With the increasing popularity of big data technology, many enterprises have invested in the Hadoop ecosystem, a large distributed system that runs on a cluster of commodity hardware to store and process a large amount of data. One of its popular processing engines, MapReduce, serves the needs of the community for years, running

all kinds of big data processing tasks from scheduling, resource management to guaranteeing fault tolerance. However, Hadoop MapReduce has significant limitations when it comes to a newer breed of big data workloads. Most visibly, MapReduce forces programs to materialize intermediate results with writes to distributed storage, which is a big problem for ad-hoc analytical queries and real-time computations like interactive queries, data exploration, and dynamic reports.

CONCLUSION

I think that the level of the learning period could be lower, at least for the user tools of Spark compared to Hadoop MapReduce. MapReduce performed repeatedly like Spark, and the technology gap in learning is much too high. For example, if I have to solve a machine learning problem, I would calculate that I will get faster answers with Spark, so I don't need a week to refine the Hadoop MapReduce code to worry about the global search time. Suppose the implementation time is made up of learning time and coding time, as Spark is faster, so the total optimized-duration is shorter for Spark processes that do complex computational tasks once. Multiple iterative computations exemplify such cases.

Which of Spark and MapReduce wins in the area of distributed data processing? We have discussed different computational speeds-up achievable with different factors depending on the type of computation of interest and costs with accommodating overheads. Apart from the actual speeds, both offer functionalities and constraints, relative development, and assurance differences. Looking at the global picture for all attributes of this study, it is a partial win on Spark's side when one deals with iterative code, for which it was devised. Almost all meaningful iterative codes run faster with Spark than with plain MapReduce or Hadoop MapReduce. With Spark, it is also somewhat easier to use shared variables with iterations not requiring a reshape of each intermediate result back into a sequence of mappers, such as with MapReduce, Hadoop MapReduce (without an intermediate reducer inclusively) and iterable addition to Hadoop MapReduce. One makes shorter codes, runs faster, and maybe implicates less codes as part of the speed of arguments advantage 7 does. Worse, such wise performance related point-v Indices: It is numerous with Spark compared to Hadoop MapReduce.

REFERENCES

[1]. J. T. Lawson and M. P. Mariani, "Distributed data processing system design - A look at the partitioning problem," Aug. 2005, doi: https://doi.org/10.1109/cmpsac.1978.810414

[2]. "What is Spark? - Introduction to Apache Spark and Analytics - AWS," *Amazon Web Services, Inc.* https://aws.amazon.com/what-is/apache-spark/

[3]. S. Sharma, "Spark & MapReduce: Introduction, Differences & Use Case," *Cloud Training Program*, Oct. 24, 2018. https://k21academy.com/big-data-hadoop-dev/spark-vs-mapreduce/

[4]. "Spark | data-analytics," *Gitbook.io*, Apr. 06, 2018. https://kks32-courses.gitbook.io/data-analytics/spark

[5]. "Apache Spark vs Hadoop MapReduce - Feature Wise Comparison [Infographic]," *DataFlair*, Sep. 19, 2016. https://data-flair.training/blogs/spark-vs-hadoop-mapreduce/

[6]. "Hadoop vs Spark vs Flink – Big Data Frameworks Comparison," *DataFlair*, Dec. 14, 2016. https://data-flair.training/blogs/hadoop-vs-spark-vs-flink/

[7]. "NIST Big Data Interoperability Framework: volume 1, definitions, version 2," Jun. 2018, doi: https://doi.org/10.6028/nist.sp.1500-1r1.

[8]. Azharuddin, "What's Better to Learn First: Spark Vs Hadoop? - DataScienceCentral.com," *Data Science Central*, Feb. 14, 2018. https://www.datasciencecentral.com/what-s-better-to-learn-first-spark-vs-hadoop/ (accessed May 05, 2024).

[9]. A. Kalron, "Hadoop vs. Spark: A Head-To-Head Comparison," *Logz.io*, Feb. 12, 2018. https://logz.io/blog/hadoop-vs-spark/