# Big-Data Analysis Review

## Big Data Processing with Apache Spark and Comparison with Hadoop

## Assignment 1

**By:**
**Dhrvukumar Patel**
**Student ID: 887349306**

**CPSC 589-01 (13667)**
**Spring, 2021**
**Professor: Dr. Chun-l Phillip Chen**
**Department of Computer Science**
**California State University, Fullerton**
**February 27, 2021**

# Table of Contents



# List of Figures and Table

## *Abstract*

*Big-Data processing could be a challenging task. Processing a large volume, high velocity, and a wide variety of data is difficult. There are many data processing tools currently available in today's market. Apache Spark and Hadoop MapReduce are two of them. With the help of these tools, one can build a highly available, scalable, and fault-tolerance system. A strong understanding is needed to make a feasible system. This paper describes Spark and Hadoop's work and discusses the advantages of Spark compared to Hadoop. It also discusses the limitations of apache spark and how one can overcome it. In the end, it provides various libraries that developers can use on top of Spark.*


# Introduction

In today's world, we generate comprehensive data ranging from petabytes ($10^{12}$ bytes) to exabytes ($10^{18}$ bytes). Wilder-James (2011) defines big data as "Data that exceeds conventional database systems' processing capacity. The Data is too big, moves too fast, or does not fit your database architectures' strictures. To gain value from this data, one must choose an alternative way to process it". Initially, when the learning trend begins, big data was characterized by three V's: Volume, Velocity, and Variety. Then, later on, Value and Veracity were added to it.

Significant benefits could be gained from processing a large amount of data, and we could enhance business intelligence, predict a better model, better analyze and predict future outcomes. Processing a large volume, high velocity, and a wide variety of data can be difficult, conventional relational database infrastructures were not built for this purpose. High availability, fault tolerance at low cost and scalability are some other factor that favors Big Data technology. Massively parallel processing (MPP) architectures, Data warehouses, or databases like Apache Spark, Hadoop, Flink are needed to manage Big Data. MPP uses nothing-shared architecture; it has a cluster of the node where data is stored and processed, and each node in the cluster has its memory, CPU, and disks. Data is partition over different nodes, and they communicate via a

network. Since no data is shared among nodes, they can process data in parallel. Some examples of it are Teradata, Vertica, Netezza, and Greenplum. Since MMP products' cost is high, it can be a limitation for some organizations (Guller, 2015). The Data Warehouse approach can be used if data is growing regularly and consistently. On the other hand, Apache Spark and Hadoop have no constrained over the data structure, it can process.

MapReduce was the first framework design to process and generate large-scale datasets in a distributed and automated ways. With the help of a map and reduce, users do not require the knowledge of data partitioning, failure recovery, or job communication. Later on, Apache Hadoop has emerged as an open-source platform for MapReduce. Although it was prevalent, it was not designed to scale well when dealing with an online and iterative process such as Stream analytics and machine learning (García, S., Ramírez-Gallego, S., Luengo, J. *et al., 2016*). An alternative to Hadoop, Apache Spark was designed, which uses in-memory primitives to perform faster-distributed computing.

Moreover, Spark is a general-purpose framework, which can be used with other distributed programming models like Hadoop or pregel. Spark was built on RDDs (Resilient Distributed Datasets), which maintains persistence in managing data, among other features. (García, S., Ramírez-Gallego, S., Luengo, J. *et al., 2016*). Later on, other competitors of Apache Spark were emerged, especially from the data-streaming side. Apache Flink (Flink, 2021) is an Apache open-source project designed to fill the gap left by Spark, which processes a mini-batch stream instead of pure streaming. Another open-source project, Apache storm (Strom, 2020), is distributed real-time processing platform and capable of processing millions of tuples per second.

García *et al. (2016)* said, to discover knowledge from data, data preprocessing is crucial because we do not want redundant, inconsistent, imperfect, corrupt data. Analyzing bid data is a more sophisticated mechanism. Albeit data preprocessing is a powerful data tool to process complex data, although it takes longer processing time. It has a wide range of functionality like data transformation, reduction, integration, cleaning, preparation, and normalization. ETL (Extract Transform Load) is another term for data processing. Extracting data from the source layer then transforms data (Cleaning, verification, sorting) and finally loading processed data back to the data warehouse.

# Literature Review

## Apache Spark

Spark is widely used and popular in industry because of its speed, flexibility, and ease of use. The Apache Spark website claims that it processes data jobs 100 times faster when compared to Hadoop MapReduce. Luu H. (2018) mentions that Spark won the Daytona GraySort contest in 2014 (sorting 100TB data). The data-bricked submission claimed that it used 10-time fewer resources and performed 100 times faster than Hadoop MapReduce. It offers more than 80 highly used data processing operators. These operators can be used with multiple languages like Scala, Java, Python, and R. In terms of flexibility, it offers a single unified data processing stack, which can reduce the operational cost and resources. Spark can be integrated with various big-data technology such as HDFS (Hadoop Distributed File System), Cluster Management and efficiently storing different data formats in binary and columnar format.

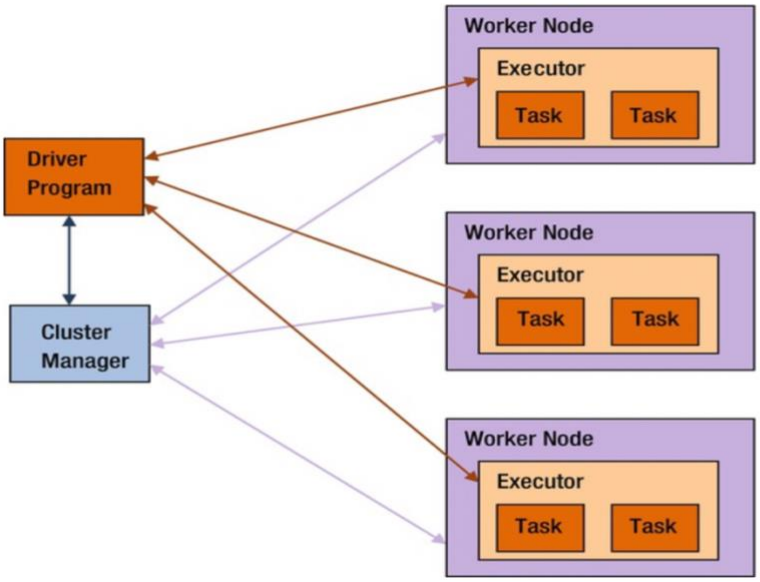



Figure-1: High-level Spark Architecture (Guller, 2015).

Figure 1 outline five key entities of the Spark application: a cluster manager, a driver program, workers, executors, and tasks. Cluster Manager offers low-level scheduling of cluster resources across applications. Each Worker Node provides memory, Storage resources, and CPU to the cluster, and based on that, resources are allocated. Its typical work is to monitor health or launch a particular process. The workers run a Spark application as distributed processes on a cluster of nodes. Currently spark support three cluster manager: Mesos, standalone and YARN (Guller, 2015). The Spark application starts a class called "*SparkSession*," which sets up configuration and APIs for expressing data processing logic. Application data processing logic is expressed using Spark APIs and the Spark driver. The Application logic can be simple as few lines of code to perform operations or to train a large machine model. The driver program is a central coordinator who interacts with the cluster manager to figure out how to distribute the code across the nodes. If a user wants to display the result, the driver collects the result from each worker node and merges them to produce the final results—the driver program launch's new jobs on a cluster when required. A driver program uses Spark as a library. An Executor is JVM (Java Virtual Machine), which executes the code concurrently using multiple threads. Specific spark

applications allocate them and are not shared among multiple spark applications. If one application misbehaves, the other application would not be affected. The lifetime of the Spark

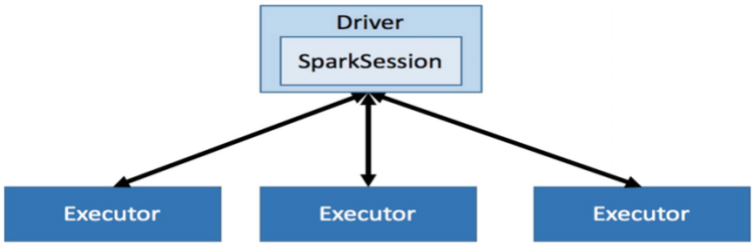



Figure 2 A small Spark cluster with three executors (Luu, 2015)

application can range from few seconds to few days. It can also cache data in disk or memory. Spark follows master-slave architecture (Figure 2). A Spark application consists of one driver and one or many executors. The executor follows data programming logic in the form of tasks. Each task has its own separate CPU core and performs some computations to either partition its output for shuffle or return a result to the driver program. The executor can run more than one task concurrently. Decisions like how many executors, the number of CPU cores each executor needs, and how much memory to allocate are made before launching the application.

**Spark Core**

Spark Core is the backbone of the Spark Application. It consists of the distributed computing infrastructure and *resilient distributed datasets* (RDDs), a generic and powerful programming abstraction for data processing. The distributed computing infrastructure is responsible for coordinating, distributing, and scheduling different tasks over different cluster nodes. It is also responsible for task failure, and efficiently data shuffling between machines. Luu (2015) defines RDD as “An immutable and fault-tolerant collection of objects partitioned across a cluster that can be manipulated in parallel.” It provides a simple set of APIs that spark developers can use to develop extensive applications quickly and efficiently and require no knowledge of where the data resides or data failure. For example, if we have 3 T.B. of data in HDFS and we need to

count the occurrence of the word *great*. One can create an instance of RDD to present all the log statements in the log file. Spark application partition the data across RDDs and counting of the word is done parallelly in the cluster, which increases the speed of computation. The RDD APIs are exposed to Scala, Java, and Python, allowing developers to pass local functions to run on the cluster, which is unique and powerful. RDDs provide various data processing operations; this includes performing data transformation, aggregation, counting, sorting, joining, filtering, and grouping. These operations are coarse-grained, i.e., the same operation is applied to multiple rows, not any specific row. The RDD offers two types of operations: transformation and action.

***Table 1: Transformations and Actions***

| Type | Evaluation | Returned Value |
|---|---|---|
| Transformation | Lazy | Another RDD |
| Action | Eager | Some result or write to the disk |

Transformations operators are lazily evaluated as, in the big data world, the immediate result is not needed. Usually, results are stored in some external storage system. To optimize, the lazy evaluation collapse or combine similar transformations into a single operation during execution. They are not executed until an action is performed. Each transformation operates on the dataset in the RDD instance and returns a new RDD with completed. Some standard transformation operators are map, filter, mappartition, flatmap, etc.

On the other hand, action will evaluate all the transformation operators that preceded it. It will either write data to the storage system or return some result to the driver. The execution of the transformation logic is only triggered by action; Hence, the absence of actions means that the

application does absolutely nothing. To optimize the application, apply as many transformations as one can before actions. Some standard action operators are count, collect, first, reduce, etc.

Currently, Spark provides various collections like Spark Streaming for processing real-time stream data, Spark SQL for batch and interactive data processing, Spark MLlib for machine learning, Spark GraphX for graph processing, and Spark R for running machine learning tasks in R. Research work on several third-party projects are going on, like REST Job Server for Apache Spark, MLbase for Machine learning library, Alluxio for memory speed virtual distributed storage system, EcairJS for node.js integration, FiloDB for in-memory column database, and delta lake (Apache website,2021).

Zaharia et al. (2016, p. 59) explain that Apache Spark provides in-memory storage, which can be very costly and does not have its own file management system. To resolve this issue, Apache Spark can be integrated with Hadoop.

**Hadoop**

Hadoop was one of the first popular open-source big data technologies. It is a fault-tolerant and scalable system for processing large datasets across a cluster of commodity servers (Hadoop,2021). Hadoop gains popularity due to its low cost, high availability, scalability, fault tolerance system, and quick & easy implementation of data processing logic.

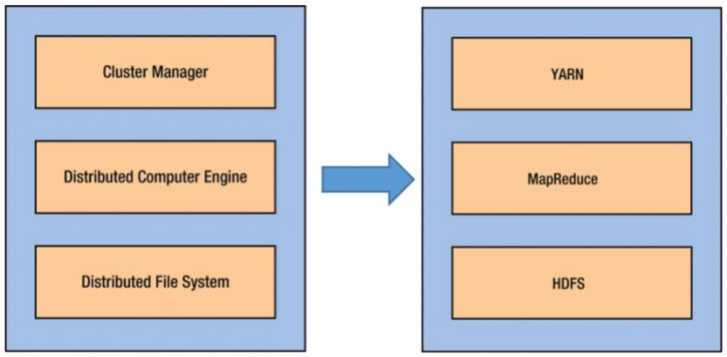



Figure 3 Key Hadoop components (Guller,2015)

Hadoop consist of three main components: cluster manager, distributed computer engine, and distributed file system. Starting with Hadoop 2.0, integration with non-Hadoop technology was

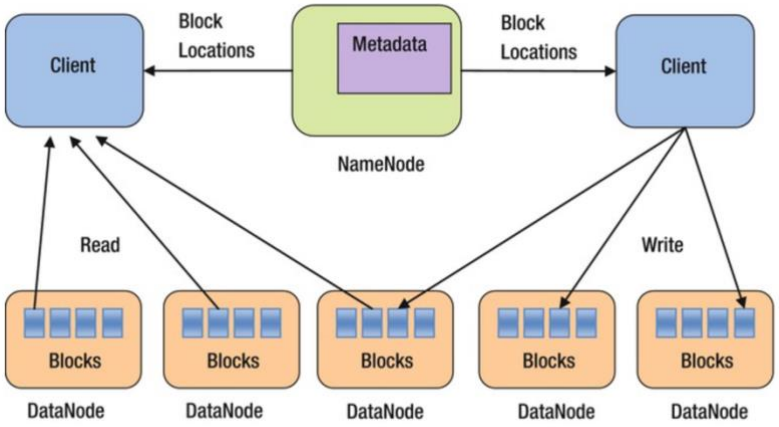



Figure 4: HDFS Architecture(Gullen, 2015)

possible; the previous version of Hadoop did not allow that. HDFS (Hadoop Distributed File System) is a block-structured file system, design to store and provide faster access to a large dataset. HDFS split files into fix-block size, called a partition. HDFS is not designed for smaller file storage. The default size of the block is 128MB. These blocks are distributed across the different machines; hence the read and write operation are handled parallelly. To improve availability, HDFS provides a default replication factor of 3.

HDFS cluster consists of two nodes, Name-Node and Data-Node. NameNode stores meta-data and manages the file system, whereas Data-Node stores actual file content. Whenever the client wants to read the data, it requests Name-Node, and NameNode sent the block locations. Then the client can directly get data from Data-Node. Similarly, when the client wants to write the data, it requests Name-Node, which verifies the existence of the file and whether the client has permission to write or not. Next, the client asks Name-Node to choose Data-Nodes for the first block of the file. NameNode creates a pipeline between all the replica nodes and the client. Moreover, the client writes the data and informs the Name-node about the completion.

MapReduce is distributed computing engine provided by Hadoop. It provides a framework for processing large datasets in parallel across a cluster of computers. It handles internode communication, load balancing, node recovery, and node failure and allows developers to focus on data programming logic. The Map function takes a key-value and returns intermediate key-value pair. It is called once per dataset. Next, it sorts all intermediate values associated with the same intermediate key and passes it to Reduce as an input. The Reduce function will aggregate those values and output aggregated values along with intermediate key. YARN (Yet Another Resource Navigator) is a general resource manager extracted from the Job-Tracker out of the MapReduce version1. YARN provides job scheduling and monitoring capabilities along with cluster management. In research from Elmasri and Navathe (2015, pp. 937–939), Hadoop v1 faced issues in scalability, multitenancy, high cluster utilization, high availability, and backward compatibility. To resolve this, Hadoop version 2 implemented YARN as a cluster manager. YARN has three key components: Resource Manager, Application Manager, and Node Manager.

The Resource Manager distributes resources among clusters based on scheduling policy (fairness or optimization). It is also responsible for tracking nodes' life cycle, whether it is down, unreachable, or newly joined. The Node Manager manages the resources available on a single node. It reports these resources to the Resource Manager. It manages Containers and provides pluggable services for Containers. Node Manager launches processes on its node with the environment and local directories set up. Application Master is responsible for managing work (task data flows, task lifecycles, task failover, etc.). MapReduce is available as a service provided by the MapReduce Application Master. An Application can be a set of processes like a MapReduce job or a long-running service like a Hadoop on-demand (HOD) cluster serving multiple MapReduce jobs. Application Manager will also periodically notify the Resource

Manager of its current Resource Requirements through a heartbeat mechanism. The functionality provided by YARN makes Hadoop a fault-tolerant, highly available, and scalable system.

# Discussion

**Advantages of Apache Spark and Comparison with Hadoop (Guller, 2015)**

**Easy to Use**: It offers a rich set of APIs that comes with Eighty plus operator compared to Hadoop, which provides just two operators: Map and reduces. Fifty lines of code in Hadoop can be implemented in 10 lines in Spark, Making developers five to ten times more productive.

**Fast and In-memory:** It is 100 faster than MapReduce if data fits in memory; reading data from memory is much faster than the disk. Even if data does not seem fit, then also ten times faster (Guller, 2015). Spark support caching data in memory for processing, which minimizes disk I/O latency. In MapReduce, it read from the disk, processes it, and then writes back to the disk, resulting in significant I/O. The application makes decisions like what to cached and when to cache. Spark and Hadoop both convert a job into a directed acyclic graph(DAG). Hadoop split complex data processing algorithms into multiple jobs, which are executed in sequence. Due to such a design, optimization is not possible. While in Spark, DAG can contain many stages and knows the stages. This allows to optimize them by minimizing data shuffling and I/O across the network.

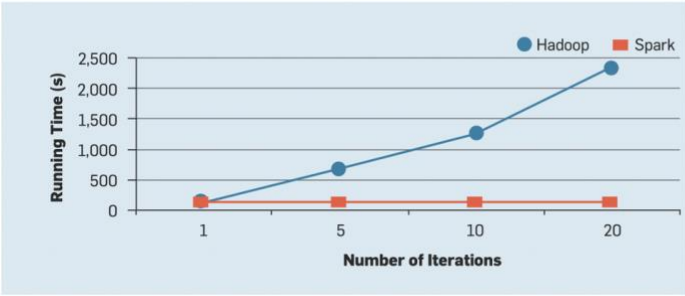



Figure 5 shows, Performance of logistic regression in Hadoop MapReduce vs. Spark for 100GB of data on 50 m2.4xlarge EC2 nodes. MapReduce takes 110 seconds per iteration because each iteration loads the data from disk, while Spark takes only one second per iteration after the first load. (Zaharia et al. ,2016, p. 59)

**General Purpose:** It is used for stream processing, batch processing, interactive analysis, graph computing, and machine learning. In comparison, Hadoop was designed for batch processing. However, stream processing and graph computing are possible with the help of frameworks. However, it creates a challenge for developers, as each has a different interface and becomes more challenging to manage. A single framework could be used for building a data processing pipeline that has multiple different tasks. Deploying separate clusters for various jobs or learning multiple frameworks or is not required. This could prevent data and code duplication; it can reduce operational complexity

**Scalable:** Processing power can be increase just by adding more nodes to the cluster. No code change is required when one simply adds a node to the cluster.

**Fault-Tolerant:** As a cluster runs hundreds of nodes, the probability of node failure is high. Spark handles the failure of a node automatically; application intervention is not required. Failure of a node can reduce the performance, but the application will not crash. Spark uses RDD to achieve fault-tolerance, which rebuilt the lost information with known information. In Hadoop, Fault-tolerant is achieved by replication using Task-Tracker and Job-Tracker.

**Ad-hoc queries and dataset joining:** It can be connected with SQL and NoSQL databases, makes joining efficient between different data formats, for example, extracting records from MySQL and MongoDB. It can also map Common-Separated-Values (CSV) files to in-memory Data-Frames, on which query can be run, without importing them into a database.

**Recovery:** It provides Lineage-based recovery, which is significantly more efficient than replication in data-intensive workloads. Recovery is typically much faster than merely rerunning the program, because a failed node usually contains multiple RDD partitions, can be rebuilt in parallel.

**Iterative Algorithms:** Iterative algorithms, iterate data processing algorithms over the same data a couple of times. Many graph processing and machine learning applications use Iterative Algorithms. Spark is ideal for such applications because it provides in-memory computing capabilities (Cache data in memory).

**Interactive Analysis:** Interactive data analysis involves making a dataset user-friendly. For instance, when the user what to summarize a vast dataset before a long batch processing job. Similarly, a business analyst analyzes data using a B.I visualization tool interactively.

**Limitation of Apache Spark and different ways to overcome it.**

Despite many advantages, it has few limitations.

**Absent of file storage system:** Spark does not have its own file management system and relies on other platforms like Hadoop HDFS, google file management system, Kafka, or any other cloud platform. The working of Hadoop and Spark is explained in the latter part of this paper.

**Expensive:** As in-memory consumption is high, it consumes much RAM for analytics and processing, which is very expensive. Although in today's world memory has become cheap, so this not a big concern.

**Near real-time processing:** Spark Streaming offers ***near*** real-time processing, which is not the same as real-time processing. It divides arriving data into batches of a pre-defined interval, and each RDD then processes each batch. Apache Flink provides much better real-time processing.

**Handling small file:** As discussed earlier, the smallest possible block size can be 128MB. Spark is not efficient in handling the small file. Bende and Shedge (2016, p. 1011) explain various technique like Hadoop Archive, merging files, File mapping, and file extraction for dealing with small files

**Handling back pressure:** Backpressure occurs when the data buffer gets full. When this happens, no data is received or transferred until the buffer is emptied. Manual application programming could be done to handle it.

**Hadoop and Spark together**

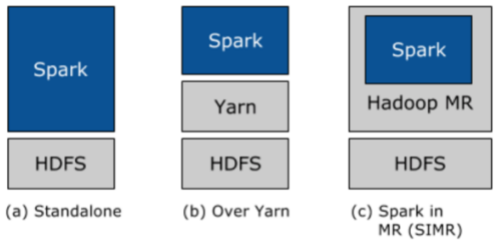



Figure 6 : Spark and Hadoop integration

There are three ways to deploy spark application in the Hadoop cluster: standalone, YARN, and SIMR (Stoica, 2020). In Standalone deployment, statically distribute resources to Hadoop cluster and run Spark side by side. The user can then run arbitrary Spark jobs to get data from HDFS. Over YARN, can be used when the user has already, or planning deployed to Hadoop YARN. It is just like replacing MapReduce with Spark. As Spark provides faster read/write than MapReduce, this is helpful for Machine learning and A.I. applications. SIMR (Spark in MapReduce), the developer can run Spark directly on top of Hadoop MapReduce v1 and does not need any administrative rights or Spark installation on any node. It can work by installing the SIMR package. It is useful when the user wants to output data from a Stream job, and the system is already working on Hadoop.

**Applications' Libraries**

**Spark-SQL:** SQL has been around since the beginning of data processing, making it easy for the user to write a query and optimize it. Spark SQL can process a petabyte of data. Spark developers can write SQL queries and use data-frame API to achieve a high-level of abstraction.

Data-frame is efficient in storing column-wise data Users can also read and write data from various storage systems and formats structured formats such as CSV, JSON, relational databases, Hive, Cassandra, and many more.

**Spark Streaming:** The Spark Streaming module enables processing real-time stream data from various data sources in a fault-tolerant and high throughput manner. Input data can be from Twitter, Kafka, TCP socket, HDFS, and many more.it works by dividing the input data into a bunch of small batches where RDDs take as an input and replicate it into a cluster

**Spark MLlib:** It provides more than fifty machine learning algorithms. It provides an abstraction for simplifying and managing different machine learning models and building tasks like evaluating, tuning, and constructing the model. It also provides persistency in moving models from development to production. A developer can implement algorithms and scale quickly.

**Spark Graphx**: It provides abstraction when working with the graph data structure. It provides a variety of graph processing algorithms like shortest path, Page-rank, and connected-components. It offers a set of fundamental operators like join-Vertices, subgraph, and aggregate-Messages. It also offers an optimized version of the Pregel API.

**SparkR**: Initially R was not designed to process large dataset which cannot be fit into a single machine. Spark provides a light-weight framework, which can be used standard R shell. It provides operations like aggregation, selection, filtering, and many more. SparkR can also be used with MLlib, for distributed machine learning. It uses spark data-frames which are similar to a data frame in R or tables in relational databases.

# Conclusion and Implication

Scalable data processing will be essential for the next generation of computer applications, but typically involves complex processing steps with different computing systems. Processing a Large dataset can be a challenging task. Spark is a versatile, fast, in-memory, scalable, and fault-tolerant data processing engine. It was designed to overcome challenges faced by Hadoop MapReduce. Although Spark had few limitations, which different ways can overcome, as mentions in this paper. Spark can be used in real-life use cases like Recommendation engines, Customer intelligence applications, fraud detection, log-processing, and data warehouse solutions. Published examples of Apache spark (batch-processing) include page personalization and recommendation at Yahoo!; managing a data lake at Goldman Sachs; graph mining at Alibaba; financial Value at Risk calculation; and text mining customer feedback at Toyota. Applications using stream processing are log mining at Netflix, network security monitoring at Cisco, and prescriptive analytics at Samsung SDS. Spark-SQL applications are Pantera for large-scale visualization and Tresata for anti-money laundering, Trifacta for data cleaning ( Zaharia et al., 2016). Apache Spark is currently leading various third-party projects on top of spark core, details for which can be found on the apache website.

**Recommended reading list or sources**


- Apache website, https://spark.apache.org/
- In Big data analytics with Spark: A practitioner's guide to using Spark for large-scale data processing, machine learning, graph analytics, and high-velocity data stream processing.
- Hadoop website http://hadoop.apache.org/

## References

Wilder-James, E. (2012, January 11). What is big data? Retrieved February 22, 2021, from https://www.oreilly.com/radar/what-is-big-data/

Apache Hadoop Project, Apache Hadoop. (2021). Retrieved February 22, 2021, from http://hadoop.apache.org/

Guller, M. (2015). Chapter 1 Big Data Technology Landscape. In *Big data analytics with Spark: A practitioner's guide to using Spark for large-scale data processing, machine learning, graph analytics, and high-velocity data stream processing*. New York, NY: Apress.

Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M. J., Ghodsi, A., Gonzalez, J., Shenker, S., & Stoica, I. (2016). Apache Spark. Communications of the ACM, 59(11), 56–65. https://doi.org/10.1145/2934664

Spark. Apache Spark: Lightning-fast cluster computing. (2021). Retrieved February 22, 2021, from https://spark.apache.org/

García, S., Ramírez-Gallego, S., Luengo, J. *et al.* Big data preprocessing: methods and prospects. *Big Data Anal* **1,** 9 (2016). https://doi.org/10.1186/s41044-016-0014-0

Flink. Apache Flink. (2021). Retrieved February 22, 2021, from https://flink.apache.org/flink-architecture.html

Strom, Apache storm. (2020). Retrieved February 22, 2021, from https://storm.apache.org/

Luu H. (2018) Introduction to Apache Spark. In: Beginning Apache Spark 2. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-3579-9_1(pp 1-13)

Elmasri, R., & Navathe, S. (2015). *Fundamentals of Database Systems* (7th ed.). Pearson.

Stoica, I. (2020, August 18). Apache Spark and Hadoop: Working Together. Databricks.https://databricks.com/blog/2014/01/21/spark-and-hadoop.html

Bende, S., & Shedge, R. (2016). Dealing with Small Files Problem in Hadoop Distributed File System. *Procedia Computer Science*, *79*, 1001–1012. https://doi.org/10.1016/j.procs.2016.03.127

**Turnit.com**

Sources 6,7 was already covered in [2], Source [1] was a book, inside it there were multiple references, so it counted twice.

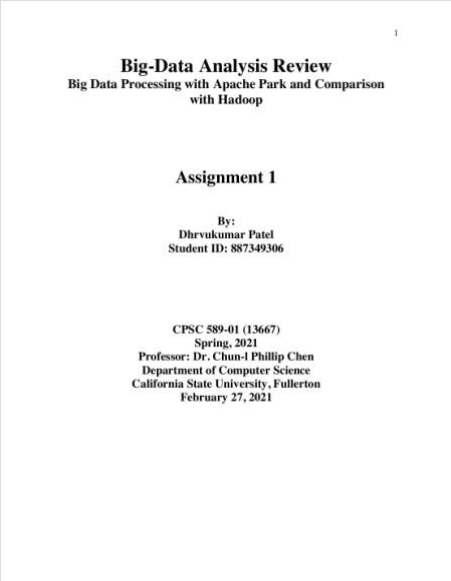
turnitin

# Digital Receipt

This receipt acknowledges that Turnitin received your paper. Below you will find the receipt information regarding your submission.

The first page of your submissions is displayed below.

| | |
|---|---|
| Submission author: | Dhruvkumar PATEL |
| Assignment title: | CPSC589-01 Assignment # 1 simila… |
| Submission title: | CPSC589-01 Dhruvkumar.docx |
| File name: | CPSC589-01_Dhruvkumar.docx |
| File size: | 3.84M |
| Page count: | 17 |
| Word count: | 3,726 |
| Character count: | 21,960 |
| Submission date: | 27-Feb-2021 11:33PM (UTC-0800) |
| Submission ID: | 1519951601 |



**Big-Data Analysis Review**
**Big Data Processing with Apache Park and Comparison with Hadoop**

**Assignment 1**

By:
Dhrvukumar Patel
Student ID: 887349306

CPSC 589-01 (13667)
Spring, 2021
Professor: Dr. Chun-I Phillip Chen
Department of Computer Science
California State University, Fullerton
February 27, 2021