A diagram of a diagram of a new year

Description automatically generated with medium confidence**PART - I**

Hello everyone! I’m here this time to explain Apache Spark. And I’ll make this article with two parts. In the first part, I will cover the Apache Spark ecosystem, and in the second part, we will discuss data manipulation with Apache Spark, with a focus on SparkSQL. But before delving into Apache Spark, let’s first understand why people need it. Let’s take a closer look at that:

**Before Apache Spark**

Before Apache Spark, it’s important to understand the context of distributed data processing. Distributed data processing refers to the practice of processing and analyzing large volumes of data by distributing the work across multiple computers or nodes within a network or cluster. This approach is commonly used for handling big data, which is data that is too large or complex to be processed on a single machine within a reasonable amount of time.

For that why many technologies have been developed for big data processing. Hadoop MapReduce was born for this purpose in 1998–2004.

**MapReduce:** Before Apache Spark, the most popular big data processing framework was Hadoop MapReduce.

MapReduce is a batch processing framework that is well-suited for large-scale data processing, but it is not as efficient for iterative processing or interactive data analysis.

MapReduce is a programming model and processing framework designed for distribution and parallel processing of large datasets. It was introduced by Google in a research paper titled [“MapReduce: Simplified Data Processing on Large Clusters”](https://research.google.com/archive/mapreduce-osdi04.pdf) published by Jeffrey Dean and Sanjay Ghemawat in 2004. MapReduce has been influential and served as the foundation for many distributed data processing systems, including Hadoop MapReduce.

[Please check my other article on Hadoop, which discusses the MapReduce ecosystem and distributed data processing.](https://medium.com/@mucagriaktas/apache-hadoop-hdfs-yarn-mapreduce-and-apache-hive-echosystem-64260f58309f)

**Apache Spark**

Apache Spark is an open-source, distributed computing framework designed for fast and efficient big data processing. It was initially developed at the University of California, Berkeley’s AMP Lab, and later donated to the Apache Software Foundation. Spark has gained significant popularity in the world of big data analytics due to its speed, scalability, ease of use and versatility.

**Speed:** Spark is one of the fastest big data processing engines available. It can process large amounts of data very quickly, even when the data is distributed across multiple machines.

**Scalability:** Spark can scale to handle very large datasets. It can be used to process datasets that are too large for traditional databases.

**Ease of use:** Spark is relatively easy to use, even for users who are not familiar with big data processing. It provides a few high-level APIs that make it easy to develop data processing applications.

**Versatility:** Spark can be used for a wide range of data processing tasks, including batch processing, real-time stream processing, machine learning, and graph processing.

**The technical processing capability of Apache Spark:**

The technical working principle of Apache Spark is designed to perform large-scale data processing tasks quickly, scalable, and efficiently. Below, you can find the key steps that explain the technical working principle of Spark:

Let’s look at what RDD is to understand the technical level of Apache Spark. Then, we’ll return to Spark’s technical steps.

**RDD (Resilient Distributed Datasets):** RDD is the fundamental data structure of Apache Spark. RDD, a distributed collection or data structure, represents Spark’s parallel and distributed data processing capabilities. RDDs are used to perform large-scale data processing tasks quickly and reliably.

**Distributed:**RDDs are stored and processed in a distributed manner across multiple computers or nodes. This allows for the parallel processing of large datasets, meaning that the data is divided and processed simultaneously across multiple computers.

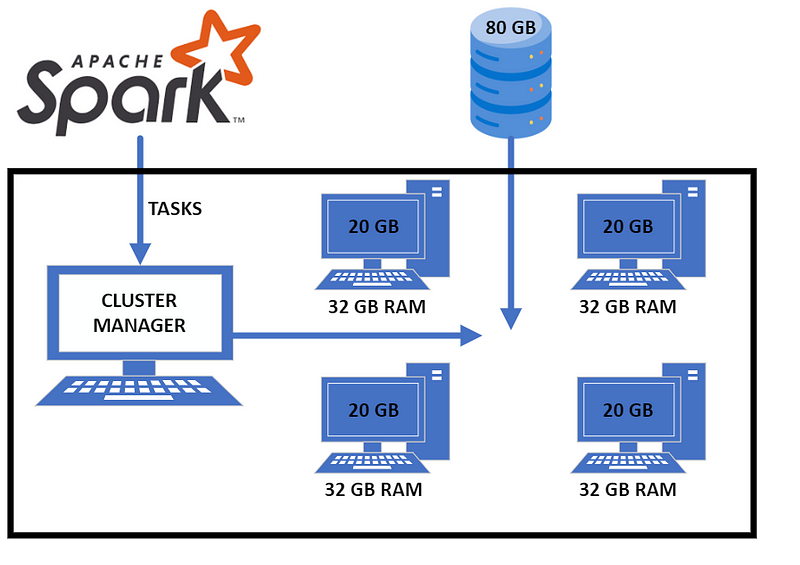
**Resilient:** RDDs are resilient to data loss or error situations. Each RDD is constructed immutably on top of the source data it was created from. Therefore, in the event of data loss, RDDs can be reconstructed.

**Immutable:**RDDs are immutable data structures. Any operation performed on an RDD results in the creation of a new RDD. This ensures data integrity and traceability for each operation step.

**Lazily Evaluated:** RDDs operate based on the principle of lazy evaluation. This means that operations on an RDD can be deferred, and the computation is only performed when the result is explicitly requested. This makes data processing more efficient.

**Catchable:**RDDs can be cached in memory. This increases the processing speed of data because the results of an RDD can be stored in memory for later operations.

I provided a brief overview of what RDDs are. Now, we can look at how Apache Spark works.



**Data Loading and Preparation:** The first step is to load and prepare the data for processing. Spark can fetch data from various data sources (**HDFS, Apache HBase, Cassandra, file systems**, etc.) and represent this data as RDDs (Resilient Distributed Datasets).

**RDD Creation and Transformations:** Data processing operations are performed using RDDs. RDD is a distributed data structure designed for parallel processing.These transformations work in parallel to achieve the desired results.

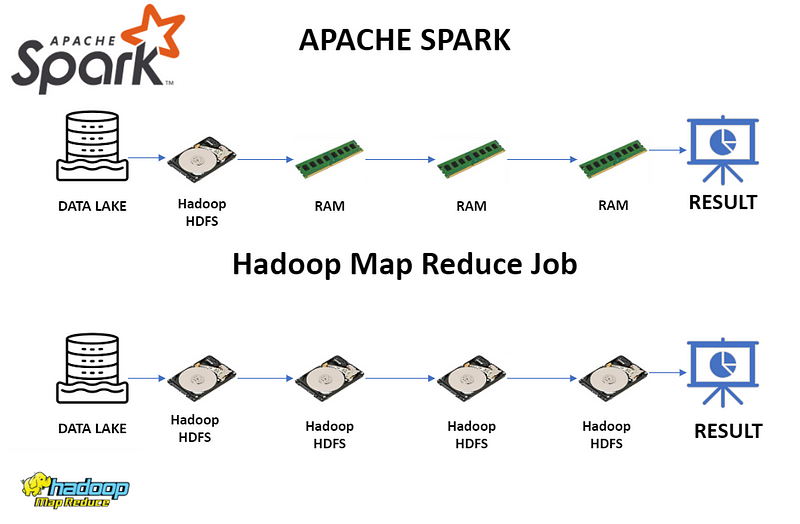
**Cache Usage:** Spark provides fast access to data during processing thanks to its ability to store data in memory **(in-memory caching – This is mean ram!).** This means that when you use Spark for your old processing, it is very fast for this process.

**Actions:** Transformations can be done on RDDs, but you need to make a call to an action to get the results.Actions trigger the execution of Spark and return results to the user.

**Parallel Processing and Distributed Execution:** Spark distributes tasks across multiple executor nodes in parallel. Each executor node can run many tasks simultaneously.Tasks are executed by the Spark core to process data, and the results are collected.

**Error Handling and Data Recovery:** Spark has internal error handling mechanisms to address potential errors and failures during processing.Caching RDDs and having idempotent operations allow for data recovery in error scenarios.

**Collecting and Processing Results:** When actions collect their results or store data, users can retrieve these results and process them as they see fit. For example, results can be used for reporting or further processing as needed.

**Why Apache Spark is so popular than Hadoop MapReduce?**

You can see why the Apache Spark is x100 faster than Hadoop MapReduce because Apache Spark is working on ram, but Hadoop MapReduce is written and read on disk.

Hadoop MapReduce, data processing using a two-step process called **“map”** and **“reduce.”** In the **“map”** phase, data is read from disk and prepared for processing. In the **“reduce”** phase, the processed data is aggregated, and the result is computed. [(Check on my other article about map and reduce)](https://medium.com/@mucagriaktas/apache-hadoop-hdfs-yarn-mapreduce-and-apache-hive-echosystem-64260f58309f)

Spark, storing data in RAM allows it to spend less time reading data from disk during the **“map”** phase and perform in-memory merging during the **“reduce”** phase instead of writing data to disk. This contributes to Spark’s faster performance compared to Hadoop MapReduce.