Hadoop vs Spark - Main Big Data Tools Explained

CONTENTS

Reading time: 11 minutes

Hadoop and Spark are the two most popular platforms for Big Data processing. They both enable you to deal with huge collections of data no matter its format – from Excel tables to user feedback on websites to images and video files. But which one of the celebrities should you entrust your information assets to?

To come to the right decision, we must divide this big question into several smaller ones:

• What is Hadoop?

• How does it work?

• What are its limitations and how does the Hadoop ecosystem address them?

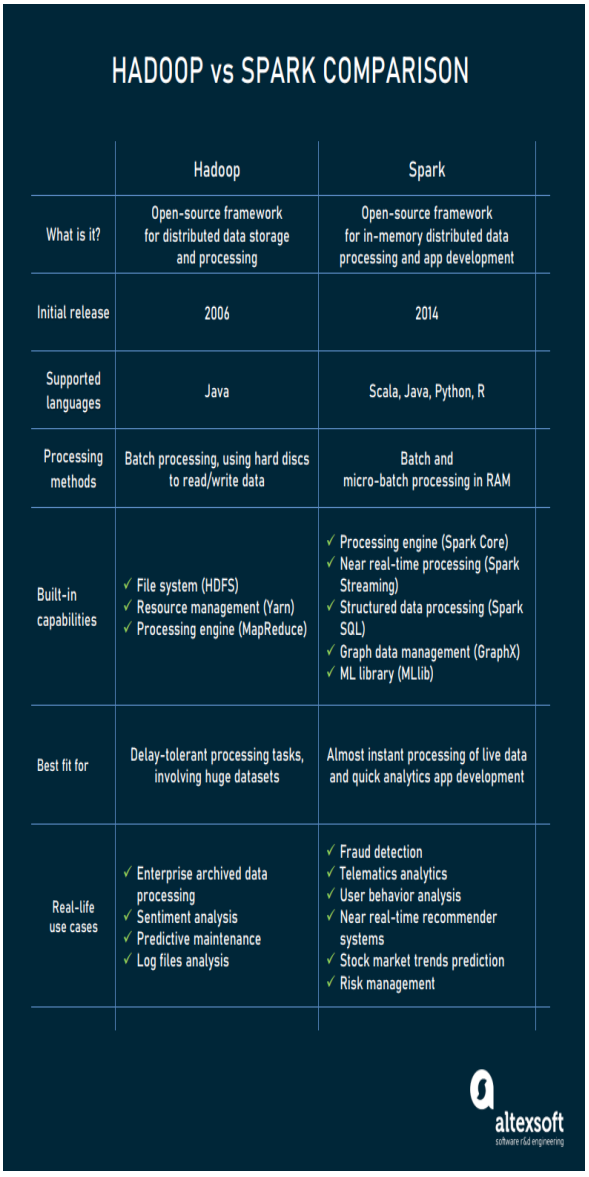
• Why did the need for Spark arise at all?

• Which Big Data tasks does Spark solve most effectively?

• What should you know about Spark cons?

This article gives you all the answers, one by one. If you already know some of them – no big deal! Skip to the next section and add to your personal

knowledge base. The table below summarizes core differences between the two platforms in question.



Hadoop vs Spark differences summarized.

What is Hadoop?

Apache Hadoop is an open-source framework written in Java for distributed storage and processing of huge datasets. The keyword here is distributed since the data quantities in question are too large to be accommodated and analyzed by a single computer.

The framework provides a way to divide a huge data collection into smaller chunks and shove them across interconnected computers or nodes that make up a Hadoop cluster. As a result, a Big Data analytics task is split up, with each machine performing its own little part in parallel. Still, an end-user sees all the fragments as a single unit.

Hadoop hides away the complexities of distributed computing, offering an abstracted API to get direct access to the system's functionality and its benefits:

• **scalability**. You can quickly add new nodes to the cluster, scaling it up from a single computer for proof-of-concept to hundreds of machines. Hadoop puts virtually no limits on the storage capacity.

• **versatility**. Hadoop allows you to leverage data from multiple sources and in different formats, both structured and unstructured. You don't need to archive or clean data before loading.

• **cost-effectiveness**. Hadoop works on low-cost, commodity hardware, making it relatively cheap to maintain.

• **fail-safe design**. The system automatically replicates information to prevent data loss in the case of a node failure.

To understand how the entire mechanism works, we need to get familiar with Hadoop structure and key parts.

Hadoop architecture, or how the framework works

There are two ways to deploy Hadoop – as a single-node cluster or as a multi-node cluster. In the former, the framework is set up on one virtual machine, which is preferable for the evaluation or test phase.

In the latter, more common, scenario, each node runs on a separate virtual machine. Obviously, Big Data processing involves hundreds of computing units. So, further reading refers to a multi-node deployment option.

Hadoop nodes: masters and slaves

Not all the nodes in the Hadoop clusters are the same. Their role determines which of the three

groups they fall into.

Master Nodes control and coordinate two key functions of Hadoop: data storage and parallel processing of data. Physically, they require the best hardware resources available.

Worker or Slave Nodes are the majority of nodes used to store data and run computations according to instructions from a master node.

A **Client Node** also called a **Gateway or Edge Node** acts as an interface between a Hadoop cluster and an outside network. It doesn't belong to the master-slave paradigm, being responsible for loading data into the cluster, describing how the data must be processed, and retrieving the output.

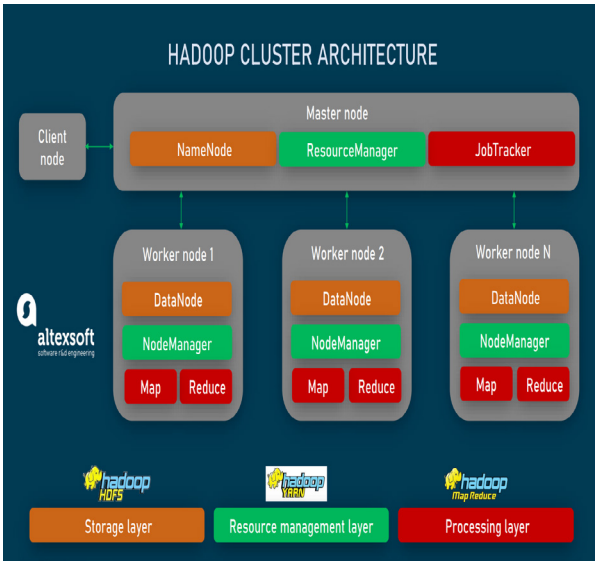
Each Hadoop cluster has three functional layers:

• the storage layer created by Hadoop's native file system – HDFS,

• the resource management layer represented by YARN, and

• the processing layer called MapReduce.

All Hadoop layers are built around master/worker interactions – or, in other words, include master and slave nodes. Let's explore how they work in more detail.



Hadoop cluster layers and nodes.

HDFS: a storage layer

The backbone of the framework, Hadoop Distributed File System (HDFS for short) stores and manages data that is split into blocks across numerous computers. By default, the block size in Hadoop is 128MB, but this can be easily changed in the config file.

HDFS works on the "write once, read many times" principle. A file stored in the system can't be modified but it can be analyzed for different purposes again and again. Following this approach, the tool focuses on fast retrieval of the whole data set rather than on the speed of the storing process or fetching a single record.

Each block of HDFS is automatically replicated in different worker nodes to ensure fault tolerance. If a node with required data fails, you can always make use of a backup. The recommended number of replicas is three, with "no more than one copy on the same node and no more than two copies in the same rack."

HDFS master-slave structure

An **HDFS Master Node**, called a **NameNode**, keeps metadata with critical information about system files (like their names, locations, number of data blocks in the file, etc.) and keeps track of storage capacity, the volume of data being transferred, etc.

Multiple Worker Nodes – **DataNodes** – house blocks of large files. Every three seconds workers send signals to their master to inform it that everything goes well and data is ready to be accessed. DataNodes are organized in racks of 40-50 nodes connected to the same network switch.

YARN: a resource management layer

An acronym for Yet Another Resource Negotiator, YARN is a software layer that monitors the usage of CPU, memory, and disk space, allocates resources to running applications, and schedules jobs based on the application requirements.

YARN master-slave structure

A **ResourceManager** serves as a Master Node and has ultimate authority over resources in the system.

Multiple slaves – **NodeManagers** – monitor resources of each virtual machine, reporting results to the **ResourceManager**.

MapReduce: a processing layer

MapReduce is often recognized as the best solution for batch processing, when files gathered over a period of time are automatically handled as a single group or batch.

The entire job is divided into two phases: map and reduce (hence the name.) Map operations deal with data filtering, sorting, and splitting while the Reduce stage takes care of aggregating and summarizing results.

This approach fits companies that want to extract detailed insights from large data volumes – rather than to get fast analytics results in real-time.

MapReduce master-slave structure

A Master Node known as a **JobTracker** receives client requests via a client node, connects with the NameNode to find out the data location, assigns tasks to slave nodes and keeps the client informed about the job status.

Slave Nodes or **TaskTrackers** perform map and reduce tasks according to the JobTracker instructions. Similar to DataNodes, they are constantly informing their Master Node on the execution progress.

Hadoop limitations

A powerful Big Data tool, Apache Hadoop alone is far from being all-powerful. It has multiple limitations. Below we list the greatest drawbacks of Hadoop.

Small file problem. Hadoop was created to deal with huge datasets rather than with a large number of files extremely smaller than the default size of 128 MB. For every data unit, the NameNode has to store metadata with names, access rights, locations, and so on. Millions of small files will evidently occupy too much memory in the Master Nodes and create lots of tasks that will slow down the processing.

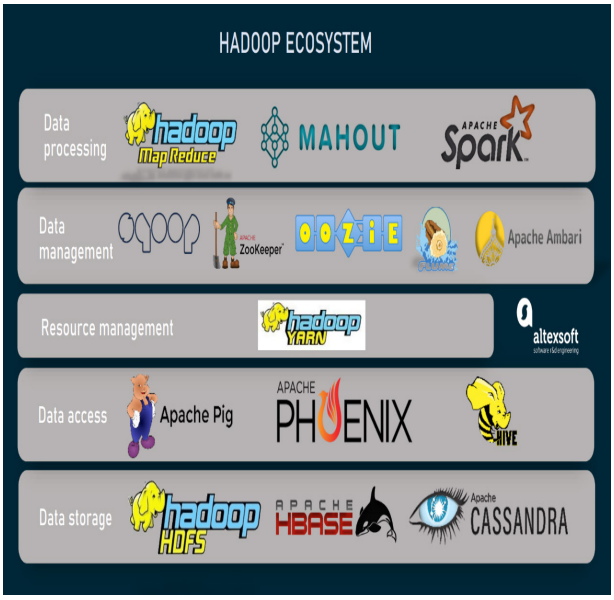
High latency of data access. Hadoop ensures high throughput which means the system's ability to deliver large data batches. But this comes at the expense of latency – or delay between user action and system response. In other words, it will take relatively long to find and retrieve a single record. High latency makes Hadoop unsuitable for tasks that require nearly real-time data access.

No real-time data processing. MapReduce performs batch processing only and doesn't fit time-sensitive data or real-time analytics jobs.

Complex programming environment. Data engineers who previously worked only with relational database management systems and SQL queries need training to take advantage of Hadoop. They have to know Java to go deep in Hadoop coding and effectively use features available via Java APIs. It's also important to understand the core principles behind Hadoop.

Apache Hadoop ecosystem

These pitfalls along with the need to cover an end-to-end Big Data workflow prompted the emergence of various additional services, compatible with each other. Together, they create a Hadoop ecosystem – a large suite of tools supplementing the framework and addressing its limitations.



Some components of the Hadoop ecosystem.

Data storage options

Apache HBase, a noSQL database on top of HDFS, is designed to store huge tables, with millions of columns and billions of rows. Its in-memory processing engine allows for quick, real-time access

to data stored in HDFS.

Alternatively, you can opt for Apache Cassandra – one more noSQL database in the family. Unlike HBase, it's a self-sufficient technology and has its own SQL-like language – Cassandra Query Language. Cassandra excels at streaming data analysis.

Data access options

HBase is often paired with Apache Phoenix, which translates common SQL queries into specific HBase commands (scans) and runs them in parallel.

There are other tools like Apache Pig and Apache Hive that simplify the use of Hadoop and HBase for data experts who typically know SQL.

Pig, developed by Yahoo, provides an SQL-like scripting language to express data flows known as Pig Latin (not to be confused with the childhood word game!) It works for all types of data – unstructured, semi-structured, and structured. But the best part of it is that you can describe a MapReduce operation with only 10 lines of Pig Latin code – instead of 200 lines in Java.

Hive, created at Facebook is a kind of query engine, utilizing the query-processing language Hive QL, that is also very similar to SQL. Main users of Hive are data analysts who work with structured data stored in the HDFS or HBase.

Data management and monitoring options

Among solutions facilitation data management are

• **Apache Sqoop**, which facilitates data transfer between Hadoop and relational databases;

• **Apache ZooKeeper** to coordinate operations and keep track of metadata in HBase;

• **Apache Oozie** for scheduling Hadoop jobs;

• Apache Flume to aggregate massive quantities of log data and move them to HDFS for analysis; and

• **Apache Ambari**, enabling administrators to monitor and control every application running on a Hadoop cluster via a highly interactive dashboard.

Processing options

Hadoop uses Apache Mahout to run machine learning algorithms for clustering, classification, and other tasks on top of MapReduce.

Of course, it's by far not all components of the ecosystem that has grown around Hadoop. Yet, for now, its most highly-sought satellite is data processing engine Apache Spark. This big star of the Big Data world was born from the need to process data much faster than MapReduce.

What is Apache Spark: its key concepts, components, and benefits over Hadoop

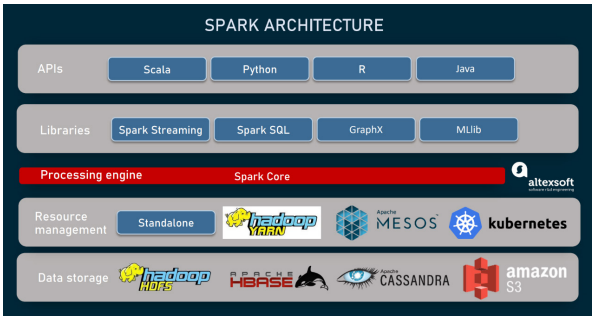
Designed specifically to replace MapReduce, Spark also processes data in batches, with workloads distributed across a cluster of interconnected servers.

Similar to its predecessor, the engine supports single- and multi-node deployment scenarios and master-slave architecture. Each Spark cluster has a single master node or driver to manage tasks and numerous slaves or executors to perform operations. And that's almost where the likeness ends.

The main difference between Hadoop and Spark lies in data processing methods.

MapReduce stores intermediate results on local discs and reads them later for further calculations. In contrast, Spark caches data in the main computer memory or RAM (Random Access Memory.)

Even the best possible disk read time lags far behind RAM speeds. Not a big surprise that Spark runs workloads 100 times faster than MapReduce if all data fits in RAM. When datasets are so large or queries are so complex that they have to be saved to disc, Spark still outperforms the Hadoop engine by ten times.



Key components of Spark and additional tools to run it.

Below, we'll explore key components of Apache Spark and what else makes it different from Hadoop besides in-memory data processing.

Spark Core and distributed data structures

The heart of the framework is its computational engine known as Spark Core. It is responsible for

- distributed data processing,
- memory management,
- task scheduling,
- fault recovery, and
- communications with an external cluster manager and data repository.

The fundamental data structure Spark Core works with is Resilient Distributed Dataset (RDD.) Basically, it's a read-only and fault-tolerant collection of records that can be processed in parallel, hiding partitioning from an end-user. RDD easily handles both structured and unstructured data.

Another available schema – DataFrames – is used to organize information in the named columns, similar to tables in relational databases.

Its extension called Datasets merges benefits of the two previous models. It supports all types of data like RDD and at the same time allows for performing SQL queries – though it happens more slowly than with DataFrames.

No default storage system and resource manager

Unlike Hadoop, which unites storing, processing, and resource management capabilities, Spark is for processing only, having no native storage system. Instead, it can read and write data from/to different sources, including but not limited to HDFS, HBase, and Apache Cassandra. It is compatible with a plethora of other data repositories, outside the Hadoop ecosystem – say, Amazon S3.

Processing data across multiple servers, Spark couldn't control resources – mainly, CPU and memory – by itself. For this task, it needs a resource or cluster manager. Currently, the framework supports four options:

- **Standalone**, a simple pre-built cluster manager;
- **Hadoop YARN**, which is the most common choice for Spark;
- **Apache Mesos**, used to control resources of entire data centers and heavy-duty services; and
- **Kubernetes**, a container orchestration platform.

Running Spark on Kubernetes makes sense if a company plans to move the entire company tech-stack to the cloud-native infrastructure.

Native libraries: Spark Streaming, Spark SQL, MLlib, and GraphX

While using an external cluster manager and data repository, Spark comes with a stack of four libraries that allow for creating various analytics apps on top of a single platform.

Spark Streaming empowers the core engine with near-real-time processing capabilities and facilitates building streaming analytics products. The module can absorb live data streams from Apache Kafka, Apache Flume, Amazon Kinesis, Twitter, and other sources and process them as micro-batches.

Just for reference, Spark Streaming and Kafka are used by

- Uber for telematics analytics;
- Pinterest for analyzing user behavior globally, and
- Netflix for near real-time movie recommendations.

Spark SQL creates a communication layer between RDDs and relational databases. It allows data scientists to conveniently query structured data in Spark programs.

GraphX offers a set of operators and algorithms to run analytics on graph data.

MLlib is a scalable machine learning library, containing algorithms for a range of ML tasks such as

classification, clustering, and regression. It also provides tools for statistics, creating ML pipelines, model evaluation, and more.

Multi-language intuitive APIs

Spark core engine, data structures, and libraries are available via developer-friendly APIs. Written in Scala, the framework also supports Java, Python, and R. This makes it easy to learn for a wide range of experts with experience in the listed languages. As a result, companies can count on a wider pool of talent – compared to Java-centric Hadoop.

Spark limitations

Obviously, Spark has some advantages over Hadoop's MapReduce engine. Yet, it also comes with certain drawbacks you should consider.

Pricey hardware. RAM prices are higher than those of hard discs exploited by MapReduce, making Spark operations more expensive.

Near, but not truly real-time processing. **Spark Streamin**g and in-memory caching allow you to analyze data very quickly. But still it won't be truly real-time, since the module works with micro-batches – or small groups of events collected over a predefined interval. Genuine real-time processing tools process data streams at the moment they are generated.

Owing to this fact, Spark doesn't perfectly suit IoT solutions. You can find better tools for real-time analytics in the Apache portfolio. For example, Apache Flink was designed specifically to process live data. Apache Storm running over HBase can also handle real-time streams better than Spark.

Issues with small files. Like Hadoop, Spark doesn't cope well with a large number of small datasets. More files within a workload mean more metadata to parse and more tasks to schedule, which can slow up the processing dramatically.

How to choose between Hadoop or Spark?

Strictly speaking, the choice is not between Spark and Hadoop, but between two processing engines, since Hadoop is more than that.

A clear advantage of MapReduce is that you can perform large, delay-tolerant processing tasks at a relatively low cost. It works best for archived data that can be analyzed later – say, during night hours. Some real-life use cases are

- online sentiment analysis to understand how people feel about your products;
- predictive maintenance to address issues with equipment before they really happen; and
- log files analysis to prevent security breaches.

Spark, in turn, shines when speed is prioritized over price. It's a natural choice for

- fraud detection and prevention,
- stock market trends prediction,
- near real-time recommendation systems, and
- risk management.

Yet, other factors – like availability of experts – may also tip the scale. Besides that, the efficiency of Spark and Hadoop depends greatly on the right tools they are combined with.

Though Spark can do without Hadoop, it is commonly teamed with HDFS as a data repository and YARN as a resource manager. So, in many cases you will actually use both platforms. Moreover, many companies run two engines – MapReduce and Spark Core – for different Big Data tasks. The former undertakes heavier operations at a bargain price while the latter deals with smaller data batches when quick analytics results are required.