# Big Data Analytics

By Metin Senturk

# Outline

- Introduction to Hadoop
- Introduction to Spark
  - What is Spark
    - Abstractions and Concepts (RDD, DAG, Spark Shell, Transformation, Actions)
  - Spark Components
  - How Spark is Used?
  - Apache Spark Architecture (Standalone, Hadoop Yarn, Spark in MapReduce, etc)

# How these tools are developed?

- Forward thinking software engineers realized that storing large datasets are impossible to store in one big physical storage (ex. Hard drive.)
- Second point is that, reading information from one large physical storage vs many small storage is **less efficient.**
- These systems should be **cost effective** and **scalable**, meaning systems should be able to build from commodity hardware.

# Introduction to Hadoop

- Apache Hadoop is an open source framework designed for distributed storage and processing of very large data sets across clusters of computers.
- The **core** consists of 4 different modules: These modules are designed specifically for big data analytics
  - Distributed-File-System (HDFS)
  - MapReduce
  - Hadoop Common
  - YARN
- Can be configured to run in 3 different modes (Standalone, semi, full distributed)

# HDFS

- HDFS is a distributed **file system**.
  - A "file system" is the method used by a computer to store data, so it can be found and used.
  - Normally, a file system is determined by the computer's operating system.
  - Hadoop’s file systems is on top of host computer, which makes it independent of OS and computer.
- HDFS breaks up files into chunks and distributes them across the nodes of the cluster.

# MapReduce

- The **map** and the **reduce**.
- Map:
  - Reading data from the database, putting it into a format suitable for analysis.
- Reduce:
  - Performing mathematical operations (i.e. Counting the number of males aged 30+ in a customer database)
- What MapReduce algorithm does;
  - It works on your query into all nodes individually where data is present, and then aggregate the final result and return to you.

# Hadoop Common

- Common libraries needed by the other Hadoop subsystems.
  - The tools needed for computer system (Windows, Unix, etc.) to read the data from HDFS.

# Yarn

- A **job scheduling** and cluster resource management software.
- YARN stands for “Yet Another Resource Negotiator“.
- After the storage aspect is mapped out, we need to figure out how to process it and what and how much resources like RAM are to be assigned to each block of data. This is taken care by YARN.
- It is developed as an alternative to MapReduce in Hadoop 1.0. It handles the resources and jobs better than MapReduce.
- Yarn has three components:
  - Resource Manager:
    - There is exactly one resource manager in one Hadoop Cluster. It does what its name says, managing the resources.
  - Node Manager:
    - Each data node has one Node Manager. Each Node Manager reports to Resource Manager.
  - Application Manager:
    - It receives the tasks(known as Applications) that are submitted to the cluster and assigns it to the data nodes

# Hadoop Configurations

**Standalone**

- All Hadoop services run in a single JVM, that is, Java Virtual Machine on a single machine.

**Pseudo-distributed**

- Each Hadoop service runs in its JVM but on a single machine.

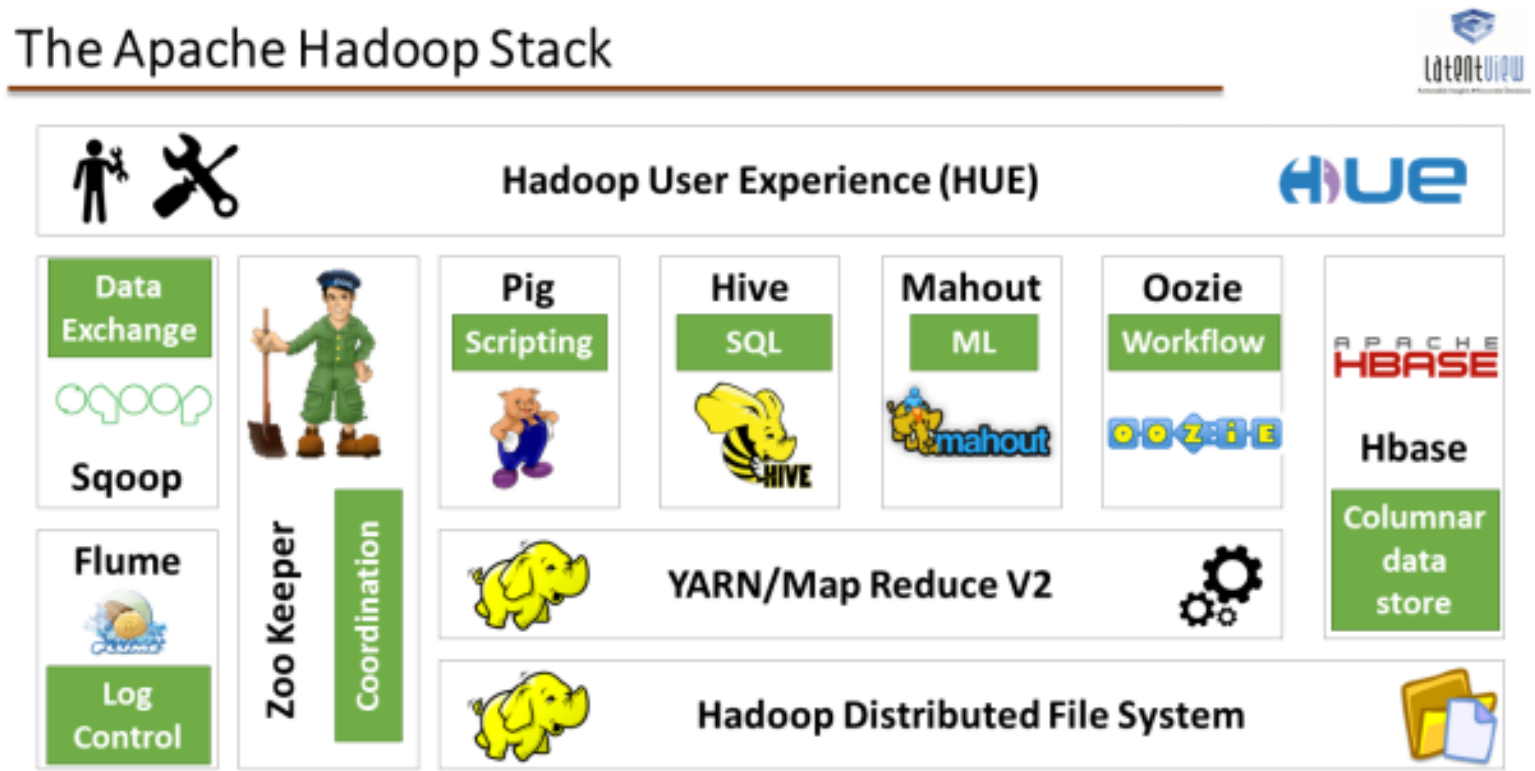
**Full-distributed**

- The Hadoop services run in individual JVMs, but these JVMs reside on different commodity hardware in a single cluster.

# Other Hadoop Components

- HBase
  - It is a NoSQL database or non-relational database. HBase mainly used when you need random, real-time, read/write access to your big data.
- Sqoop
  - It is managed to import data from relational databases such as Oracle and MySQL to HDFS and export data from HDFS to relational database.
- Flume
  - Flume distributed service for ingesting streaming data. So, Distributed data that collect event data & transfer it to HDFS. It is ideally suitable for event data from multiple systems.
- Pig
  - Open source data flow system. It convert pig script to Map-Reduce code and saving producer from writing Map-Reduce code.
- Impala
  - A high-performance SQL engine which runs on a Hadoop cluster. Ideal for interactive analysis. It has a very low latency which can be measured in milliseconds.
- Hive
  - Similar to Impala, for structured data. It preferred for data processing and ETL ( extract, transform and load) operations
- Oozie
  - Oozie is a workflow or coordination method that you can employ to manage Hadoop jobs.

# Other Hadoop Frameworks



# Introduction to Spark

- Spark is written in Scala Programming Language and runs in JVM.
- Apache Spark can be integrated with various data sources like SQL, NoSQL, S3, HDFS, local file system etc.
- Good fit for iterative tasks like Machine Learning (ML) algorithms.
- Spark runs multi-threaded tasks inside of JVM processes, whereas MapReduce runs as heavier weight JVM processes. This gives Spark faster start up, better parallelism, and better CPU utilization.
  - Can anyone explain why this is more faster?

# Introduction to Spark

- Apache Spark is an exceptionally a cluster computing technology, intended for quick computation.
- It depends on Hadoop MapReduce and it stretches out the MapReduce model to effectively utilize it for more kinds of computations, which incorporates intuitive questions and stream handling.
- The fundamental element of Spark is its in-memory clustering that expands the preparing pace of an application.
- Spark is just another YARN job
- Spark's goal was to generalize MapReduce to support new apps within same engine
- Two main additions:
  - Fast data sharing
  - DAG
- Handles batch, interactive, and real-time within a single framework
- Native integration with Java, Python, Scala
- Programming at a higher level of abstraction
- More general: map/reduce is just one set of supported constructs

# Hadoop vs Spark

- Hadoop MapReduce writes all of the data back to the physical storage medium after each operation. This is done to make the system fault tolerant.
- Spark is **fast.** Spark handles most of its operations “in memory” – copying them from the distributed physical storage into far faster logical RAM memory. Spark ensures fault tolerance by Resilient Distributed Datasets (RDD).
- Spark does not have its own distributed storage system. Needs a third party storage system (i.e. HDFS, etc.)

# Hadoop vs Spark

- Spark provides machine learning libraries such as MLib, Hadoop needs a third party software, such as Apache Mahout.
- Real time stream processing in Spark is much more advanced than Hadoop.
  - Real-time processing means that data can be fed into an analytical application the moment it is captured, and insights immediately fed back to the user through a dashboard, to allow action to be taken.
- Security and related infrastructure is advanced in Hadoop, but Spark does not have any.
- Spark doesn’t execute the tasks immediately but maintains a chain of operations as meta-data of the job called DAG.

# Directed Acyclic Graph (DAG)

- Spark allows programmers to employ complex, multi-step data pipelines using directed acyclic graph (DAG) pattern.
- It allows in-memory data sharing across DAGs, so that different jobs can work with the same data without going to disk.
- Direct - Transformation is an action which transitions data partition state from A to B.
- Acyclic -Transformation cannot return to the older partition
- DAG is a designated graph with no inscribed sequences. It reads data from HDFS and Map & Reduce operations are applied. It comprises a series of vertices such that every edge is directed from initial to succeeding in the progression.

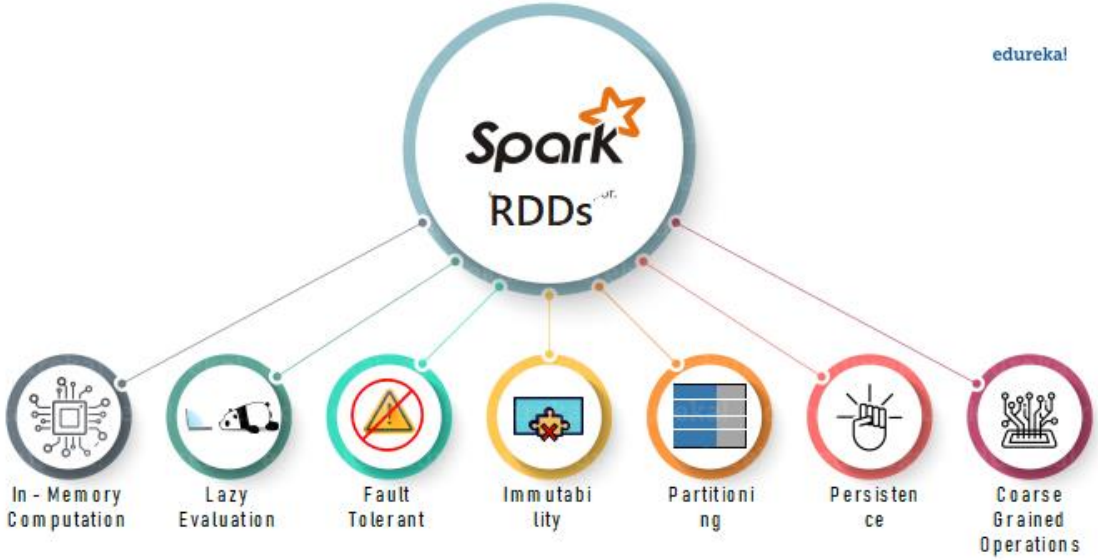
# Spark Shell

- An interactive Shell that can execute a command line of the application effective because of the interactive testing and capability to read a large amount of data sources in various types.

# Resilient Distributed Datasets (RDD)

- **Immutable** collections of objects, spread across cluster
- Statically typed: RDD[T] has objects of type T
- Collections of objects across a cluster with user controlled partitioning & storage (memory, disk, ...)
- Can be built via parallel transformations (map, filter, …)
- A fault-tolerant collection of elements that can be operated on in parallel
- There are currently two types:
  - parallelized collections – take an existing Scala collection and run functions on it in parallel
  - Hadoop datasets – run functions on each record of a file in Hadoop distributed file system or any other storage system supported by Hadoop

# RDDs are Fault Tolerance




**Dataset**
Data storage created from HDFS, S3, HBase, JSON, text, etc.
Once spark reads the data, it can be referenced with an RDD
Or from transforming another RDD: **RDD are immutable**

**Distributed**
Distributes across cluster of machines
Data is divided into ***partitions*** and partitions divided across machines

**Resilient**
Spark tracks history of each partition
Error recovery like node failures, slow processes

# RDDS knows their Partitioning

- Two types of operations on RDDs:
  - transformations and actions
- Transformations are lazy (not computed immediately)
- The transformed RDD gets recomputed when an action is run on it (default)
- RDD can be persisted into storage in memory or disk

# Transformations

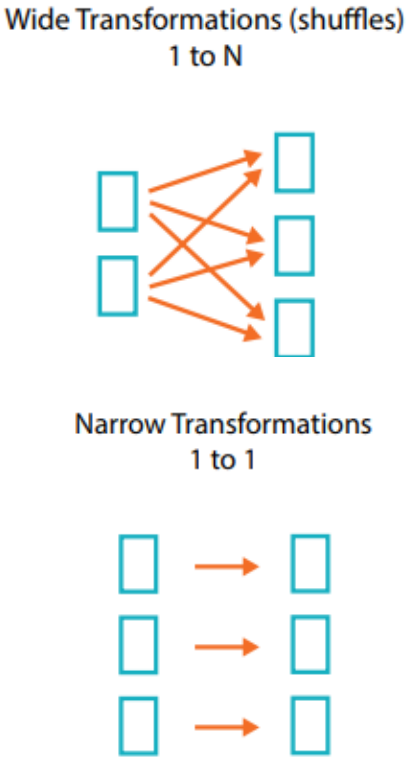
- RDD are immutable but we can transform one RDD to another RDD. Spark does ***lazy transformations*** (execution will not start **until an action is triggered**).
- Transformations create a new dataset from an existing one

## Narrow transformations

- Like **map and filter**
- Do not imply transferring data through the network
- Depends on **memory and CPU**

## Wide transformations

- For example, *groupByKey* transfers data with same key to same partition
- **Shuffle operation** across network
- Also depended on **interconnection speed between nodes**

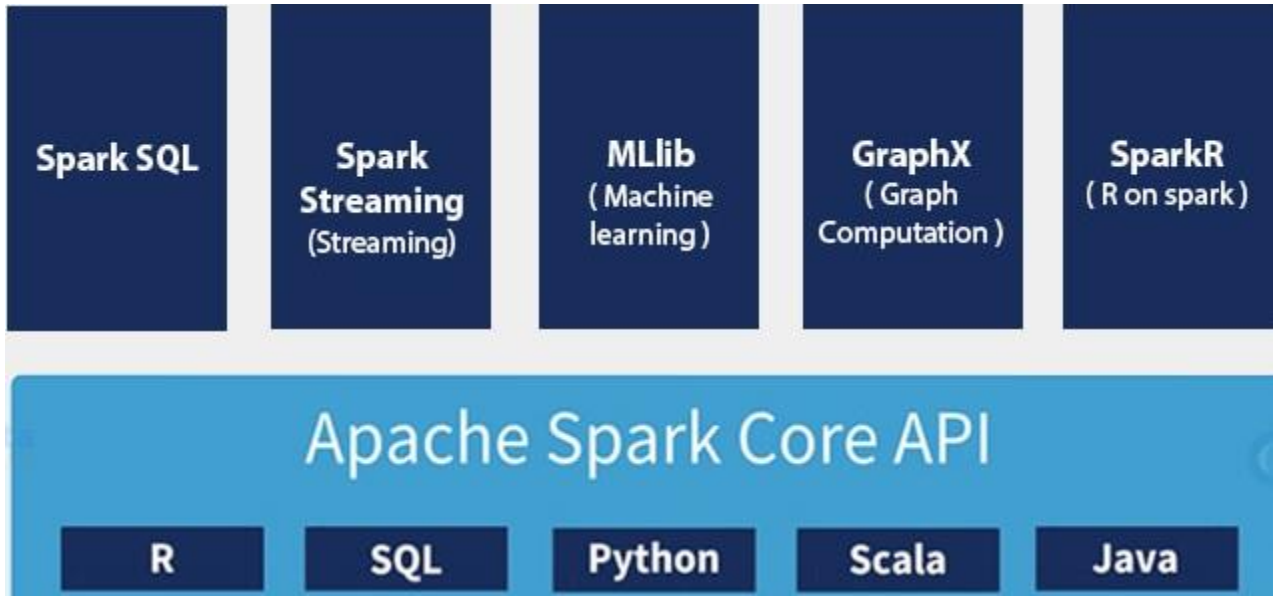


# Actions

- It turns final result to driver program or corresponds it to the external data store.

# Spark Components

- Spark SQL, Spark Streaming, Spark MLlib, Spark GraphX and SparkR.



# Apache Spark Architecture

Apache Spark follows a master/slave architecture with two main daemons and a cluster manager

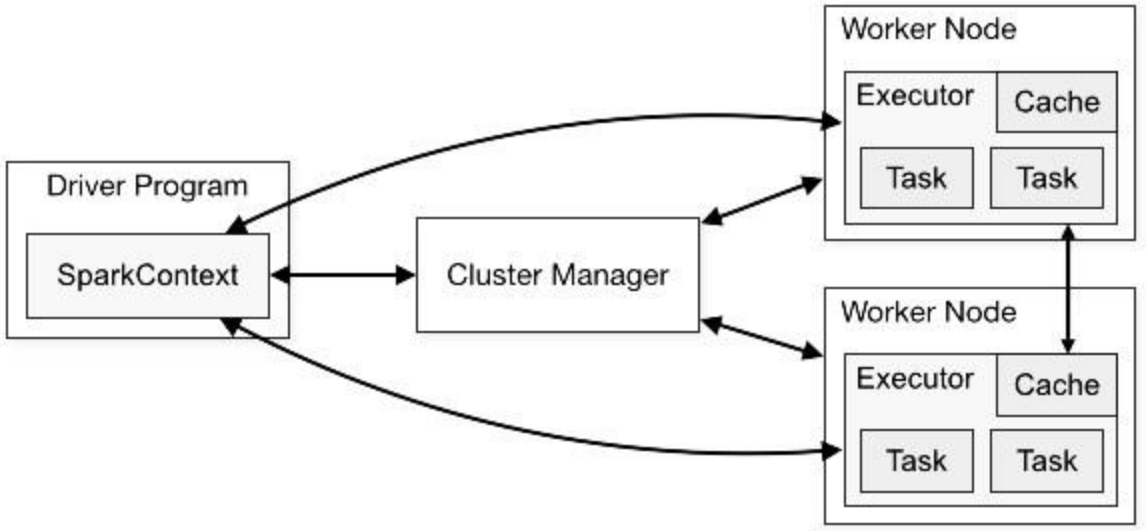
1. Master Daemon – (Master/Driver Process)
2. Worker Daemon –(Slave Process)

**Important Things:**

- Master is the instance that hosts the Driver Program and the Worker is the instance that hosts executors.
- Spark also has a local mode, where the driver and executors run as threads on your computer instead of a cluster, which is useful for developing your applications from a personal computer.

**Steps**

1. A Spark application runs as independent processes, coordinated by the SparkSession object in the driver program.
2. The resource or cluster manager assigns tasks to workers, one task per partition.
3. A task applies its unit of work to the dataset in its partition and outputs a new partition dataset.
4. Results are sent back to the driver application or can be saved to disk.



# Apache Spark Architecture – Worker Node

**Worker Node**

Workers are the instances where executors live to execute the user-written application code in the cluster.

Executor a JVM process launched for an application on a worker node, which runs tasks and keeps data in memory or disk storage across them. Each application has its own executors.

- General JVM executor to execute spark workflows
- Core on which all computation is done
- Interface to rest of the Hadoop ecosystem, e.g., HDFS

# Apache Spark Architecture – Cluster Manager

**Cluster Manager**

Responsible for allocating resources across the spark application. The Spark context is capable of connecting to several types of cluster managers like **Mesos, Yarn or Kubernetes** apart from the Spark's standalone cluster manager.

- System to manage provisioning and starting of worker nodes
- Hadoop YARN (Hadoop cluster manager):
  - Same cluster can be used with both Hadoop MR and spark.
- Standalone:
  - Special spark process that takes care of starting nodes at beginning of computation and restarts them on failures.
- Spark in MapReduce:
  - Spark in MapReduce is used to launch spark job in addition to standalone deployment. With SIMR, user can start Spark and uses its shell without any administrative access.

# Apache Spark Architecture – Driver Program

**Driver Program**

The Spark context objects the main program is the driver program. It is a (Java Virtual Machine) JVM process that is responsible for the execution of the tasks.

It is the central point and the entry point of the Spark Shell (Scala, Python, and R).

Spark Driver contains various components – DAGScheduler, TaskScheduler, BackendScheduler and BlockManager responsible for the translation of spark user code into actual spark jobs executed on the cluster.

# how Spark executes processing tasks on a distributed system?

- There is a “driver” process that is in charge of taking the user’s code and converting it into a set of multiple tasks.
- There are also “executor” processes, each operating on a separate node in the cluster, that are in charge of running the tasks, as delegated by the driver.
- A Spark **application** is a user built program that consists of a driver and that driver’s associated executors.
- A Spark **job** is task or set of tasks to be executed with executor processes, as directed by the driver.
- Finally, a job is triggered by the calling of an RDD Action.

# What does Spark Do?

- Spark is capable of handling several petabytes of data at a time, distributed across a cluster of thousands of cooperating physical or virtual servers.
- Developer libraries and APIs and supports languages such as Java, Python, R, and Scala
- Used with distributed data stores such as Hadoop's HDFS, and Amazon's S3, with popular NoSQL databases such as Apache HBase, Apache Cassandra, and MongoDB
- Databricks lists following use cases for Spark
  - Data integration and ETL
  - Interactive analytics or business intelligence
  - High performance batch computation
  - Machine learning and advanced analytics
  - Real-time stream processing

# What does Spark Do?

**Use Cases**

- **Stream processing:**
  - Streams of data related to financial transactions, for example, can be processed in real time to identify– and refuse– potentially fraudulent transactions.
- **Machine learning:**
  - Spark's ability to store data in memory and rapidly run repeated queries makes it a good choice for training machine learning algorithms. Running broadly similar queries again and again, at scale, significantly reduces the time required to go through a set of possible solutions in order to find the most efficient algorithms.
- **Interactive analytics:**
  - Business is rarely clean or consistent enough to simply and easily be combined for reporting or analysis. Extract, transform, and load (ETL) processes are often used to pull data from different systems, clean and standardize it, and then load it into a separate system for analysis.
- **Data integration:**
  - In a business is rarely clean or consistent enough to simply and easily be combined for reporting or analysis. Extract, transform, and load (ETL) processes are often used to pull data from different systems, clean and standardize it, and then load it into a separate system for analysis.

# Who uses Spark?

Most active community in big data, with 50+companies contributing

# Where to go?

- https://stackoverflow.com/questions/32621990/what-are-workers-executors-cores-in-spark-standalone-cluster
- https://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html
- https://www.coursera.org/lecture/hadoop/introduction-to-apache-spark-9cq0R
- https://www.dezyre.com/article/apache-spark-architecture-explained-in-detail/338

# SparkContext Object

- First thing that a Spark program does is create a SparkContext object, which tells Spark howto access a cluster
- http://spark.apache.org/docs/latest/cluster-overview.html

# Questions

Thanks