Information System Design

Lecture 7: Big Data

Dr. Moustafa Alzantot

Big data

What is it really

A buzz word!

There is no standard definition for what is / what is not big data.

- *Everyone talks about it.*
- *Nobody knows how do to it.*
- *Everyone thinks everyone else is doing it.*
- *So everyone claims they are doing it.*

Big data

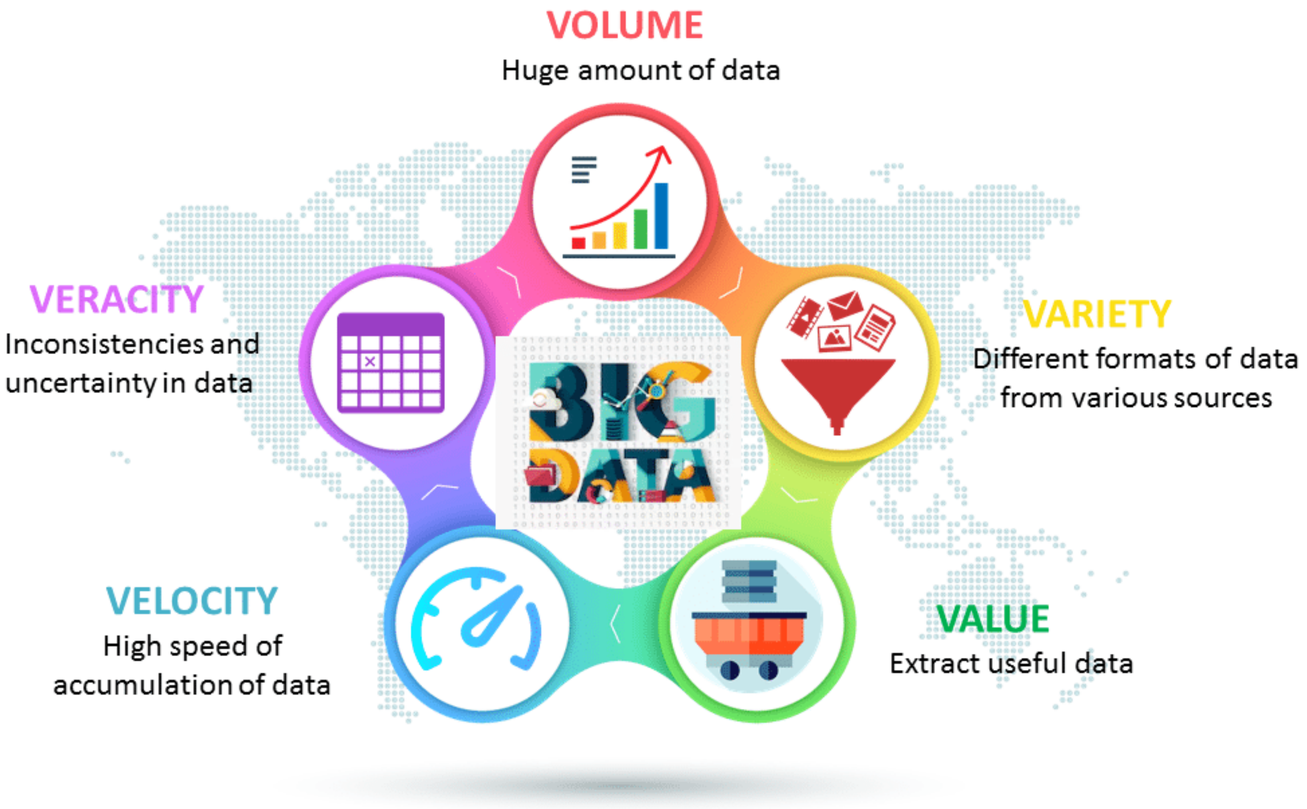
What is it really

Commonly used definition (from Wikipedia)

> ***Big data*** refers to huge volumes of data that can not be stored and processed by any traditional data storage or processing systems.

5 V’s Characteristics of Big Data

Often characterized by what is known as the 5V’s.



5 V’s Characteristics of Big Data

Volume

Big data is generated in unimaginable amounts of information at every second.

Example: social media, online shopping, cell phones, sensors, etc.

For example: Facebook has over 1.8 billion daily active users (in Q4 2020) [1]

[1] https://www.businessofapps.com/data/facebook-statistics/

5 V’s Characteristics of Big Data

Variety

Big data is generated in multiple varieties. It can be either structured, semi-structured or unstructured.

Structured data: Has a well defined structure. Can be stored in well defined columns. E.g. data stored in RDBMS.

Semi-structured data: Data may have a certain structure but not all information collected has identical structure. No rigid schema. E.g. HTML web pages, CSV and JSON files.

Unstructured data: does not have structured format. It changes a lot from one instance to another. E.g. satellite images, videos from security cameras, audio recordings.

5 V’s Characteristics of Big Data

Velocity

The term ‘velocity’ refers to speed of generation.

Data is being generated very fast and needs to be processed fast, as well.

Examples:

- Location-based promotion: based on your current location and your purchase history -> identify what you like and recommend some promotions at nearby stores.

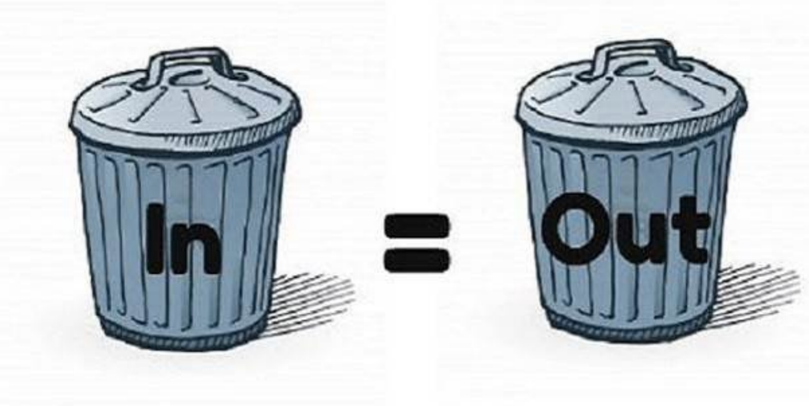
5 V’s Characteristics of Big Data

Veracity

Data veracity = how accurate or trusted is the data.

Big data can be noisy and uncertain. It can also be full of biases.

The evidence from data analysis is only
valuable if the data has a satisfactory quality.



5 V’s Characteristics of Big Data

Veracity

Data quality can be defined in terms of

- Accuracy of data
- Reliability of the data source

- We need data provenance: a record that keeps track of where is the data coming from, what was collected and how it was analyzed / transformed prior to current use.

5 V’s Characteristics of Big Data

Value

Big data is meaningless if it does not provide value toward some meaningful goal.

GFS : Google File system

Large scale distributed file system developed by Google.

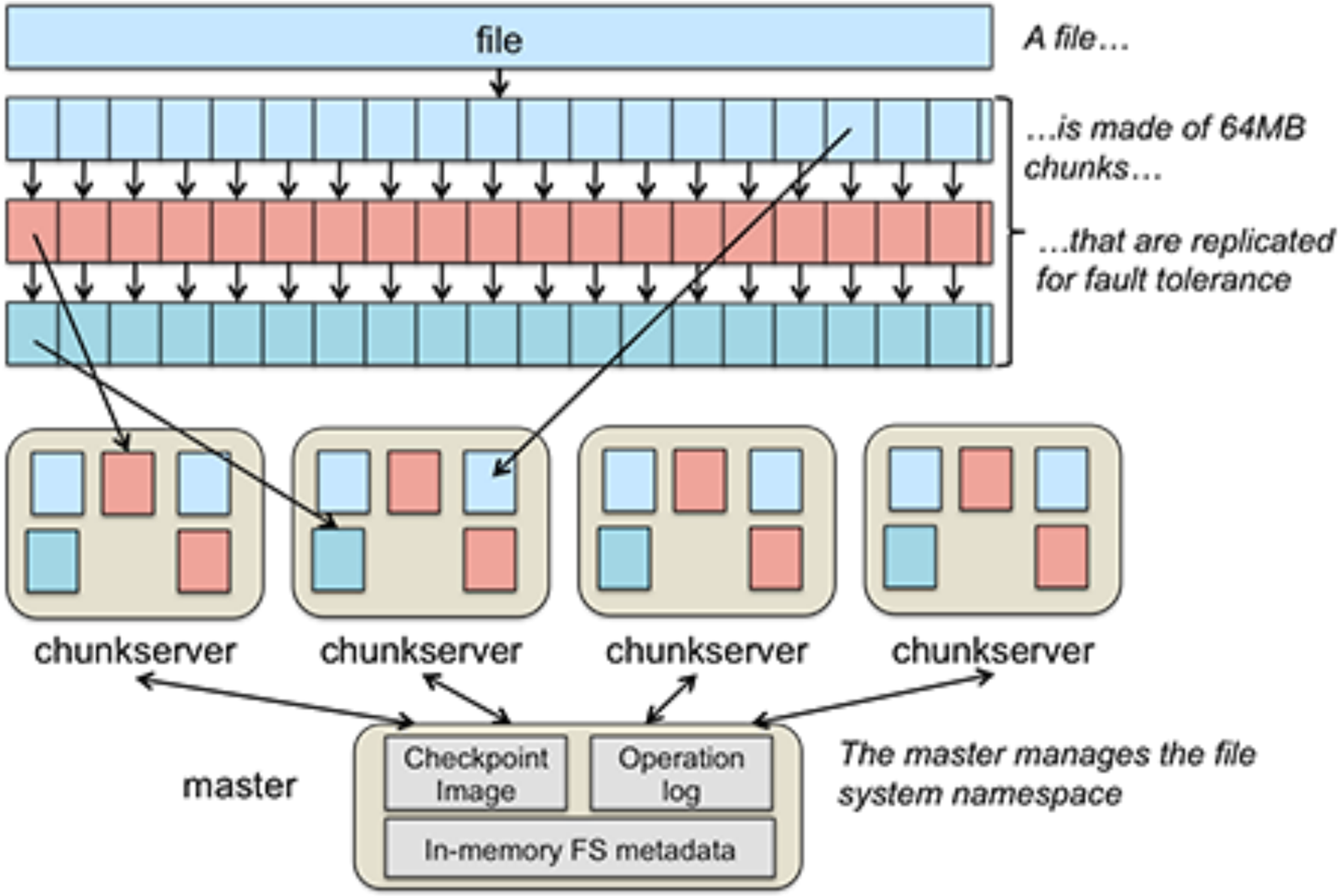
Optimized for use cases at Google:

- Running on inexpensive commodity machines.
- Component failure is considered the norm rather than exception.
- Uses constant monitoring to detect and recover errors.

GFS : Google File system

- GFS cluster consists of a single ***master*** and multiple ***chunkservers*** and is accessed by multiple clients.
- Files are divided into fixed-size chunks (chunk size = 64KB), each chunk has an ID.
- Chunks are stored as Linux files on ***chunkservers***.
- Each chunk is replicated on multiple ***chunkservers*** (3 times by default).
- Master maintains all file system metadata (mapping from files to chunks and current locations chunks). All metadata is stored in memory for fast access.

GFS : Google File system



GFS : Google File system

- GFS client code communicates with the master and chunk servers to read/write data.
- GFS client interacts with the master for metadata operations (asking for Chunk ID and chunk location of a given file), but actual data communications take place between client and chunk server directly.
  - *This prevents the master from being a bottleneck.*

GFS : Google File system

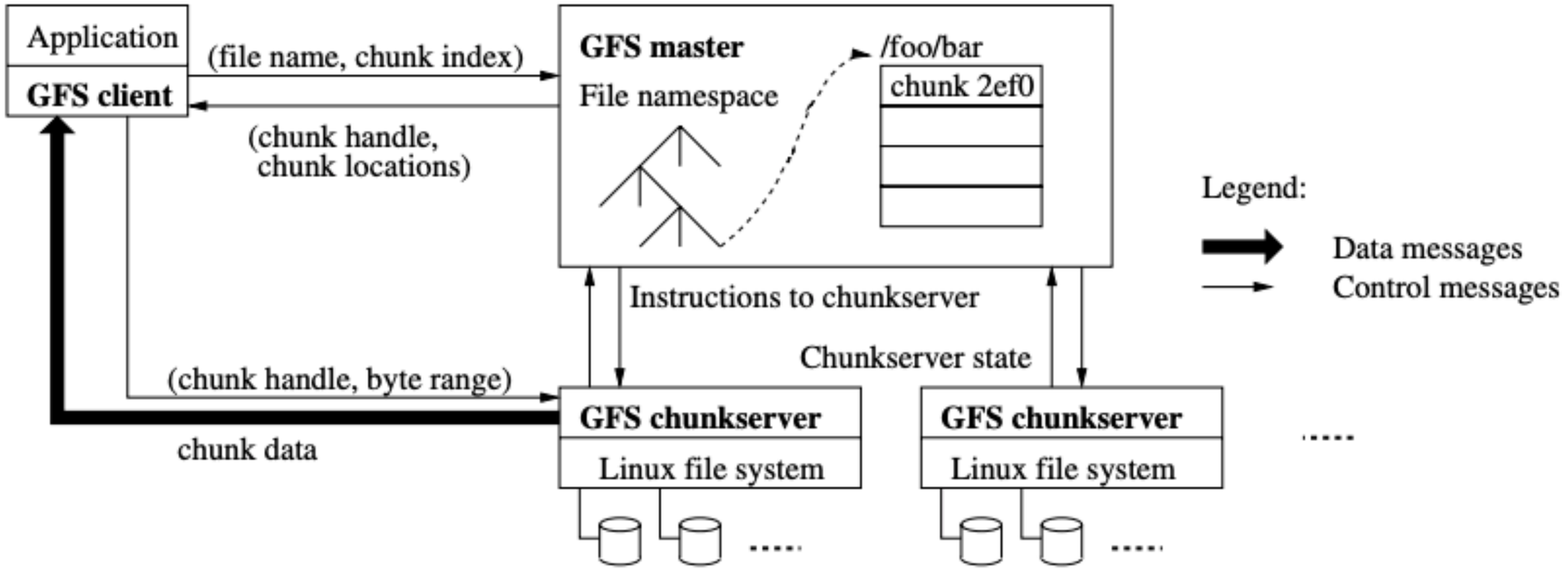


Figure 1: GFS Architecture

Google Map Reduce

Map reduce is a programming model to simplify processing of large datasets.

Motivation for MapReduce:

- There was need for an abstraction that hides many system-level details from the programmer.
- Allows programer to specify parallel computation (***What***) without knowing (***how***) the hardware is organized (i.e. job scheduling, fault tolerance, etc.)

Google Map Reduce

Inspired from map and reduce operations commonly used in functional programming languages like LISP.

Computation takes a set of input key/value pairs and produces a set of output key/value pairs.

User of MapReduce library expresses computation in terms of two functions: Map and Reduce.

Google Map Reduce

Map function:

Takes an input pair and produces a set of intermediate key/value pairs.

map(k,v) -> list(k1,v1)

Reduce function:

Accepts an intermediate key and set of values for that key. It merges together these values to compute the final output for that key.

reduce(k1, list(v1)) -> v2

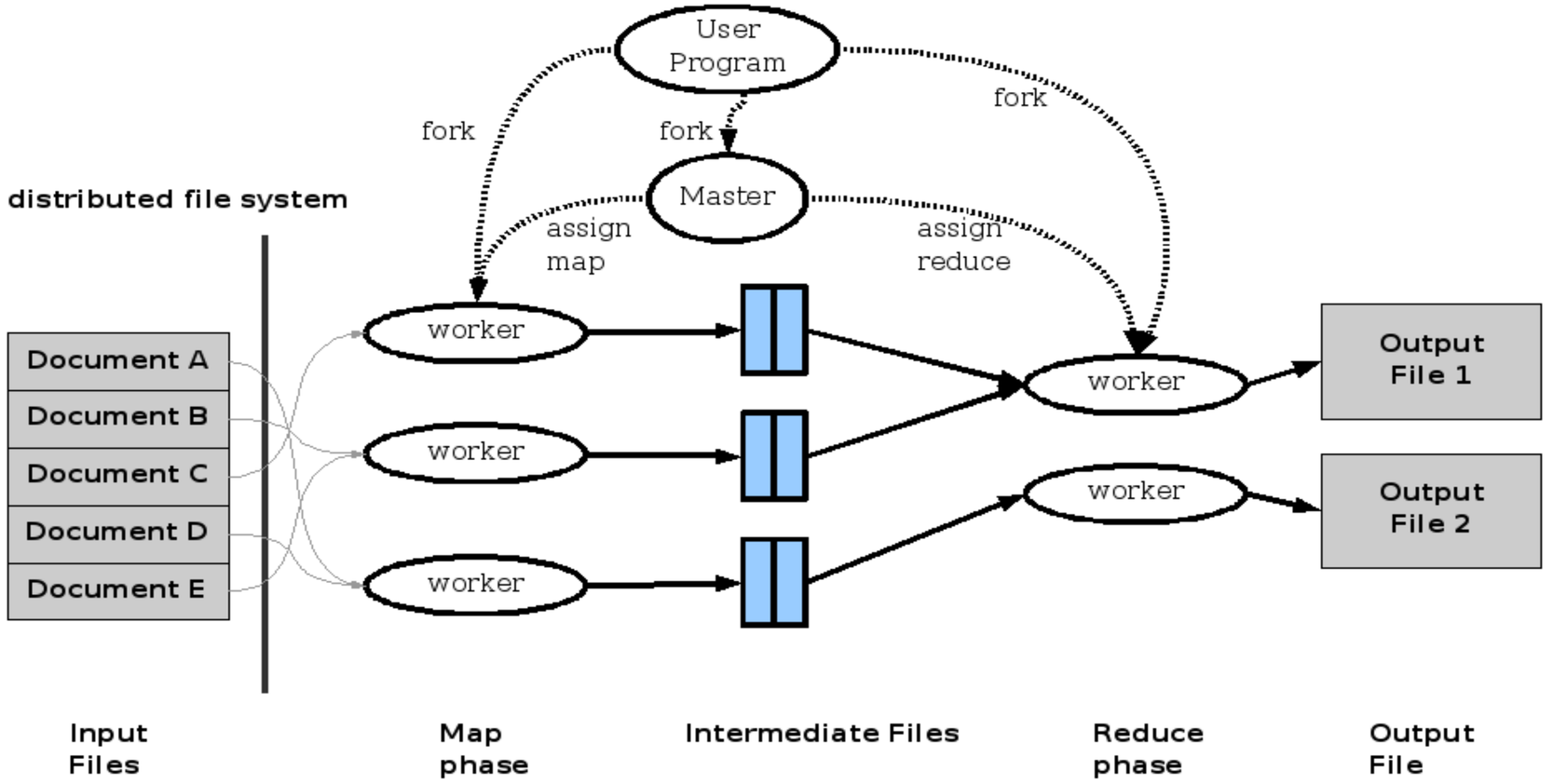
Google Map Reduce

Execution

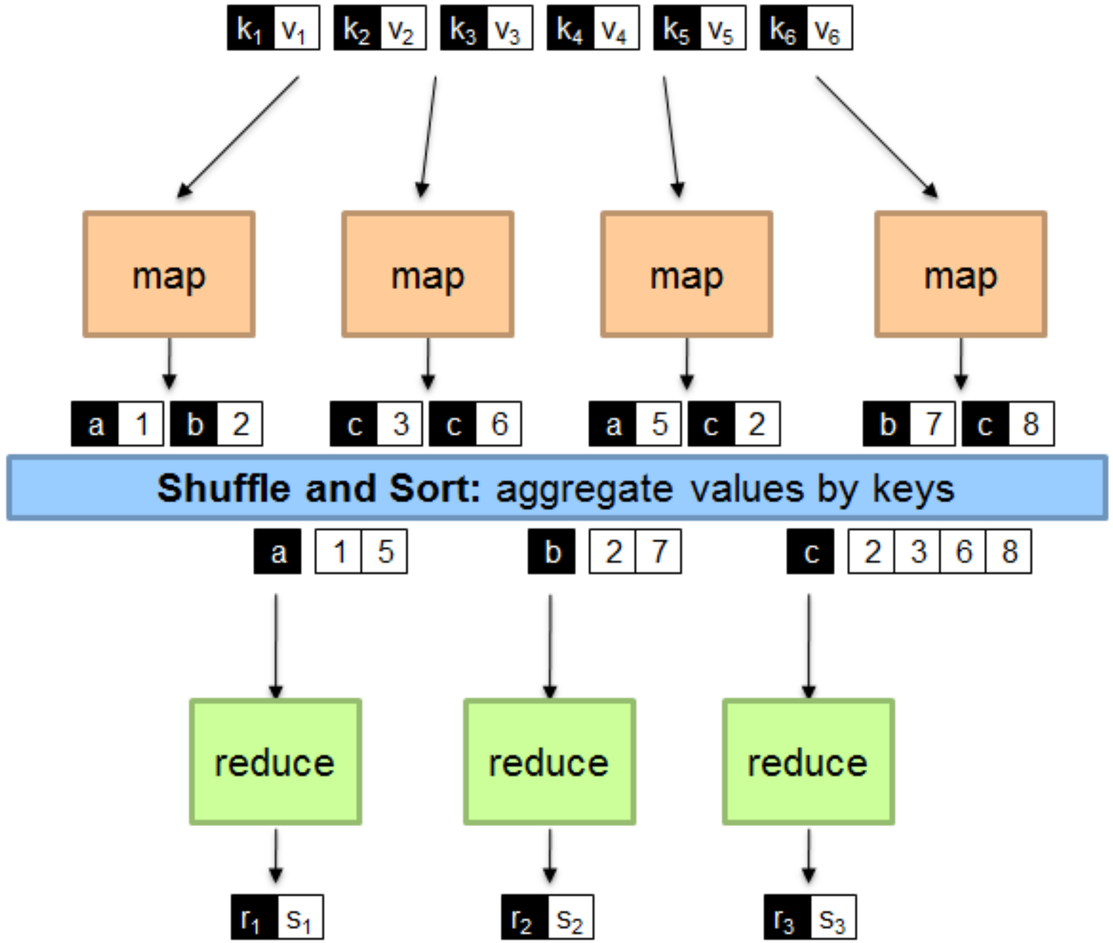
- Input data is split into **M** splits.
- User program is copied into multiple machines. One of them is special called the ***master***.
- **M** machines are assigned the "***mapper***" work by the master.
- A worker who is assigned a map task will read the contents of corresponding input data split and process it using the user-defined Map function.
- Outputs from map workers will be stored in local disk partitioned into R regions by partitioning function (hash(key) mod R).

- R machine are assigned the reduce work. They read all intermediate data in a corresponding partition, sorts the data by intermediate key so that all occurrence of the same intermediate key are grouped together. Then, passes the intermediate key and the group of corresponding values to the user's defined ***Reduce*** function.
- Outputs from all reduce workers are appended together.

Google Map Reduce



Google Map Reduce



Apache Hadoop

Apache Hadoop : a collection of open-source framework for distributed storage processing of big data.

Includes :

- **Hadoop common**: common utilities
- **Hadoop distributed file system (HDFS)**: distributed file system that stores data on commodity machines (open source equivalent of GFS)
- **Hadoop MapReduce**: an implementation of MapReduce programming model.
- **Apache YARN**: manages resources in a computing cluster.

Apache Spark

Apache Spark started as a research project at UC Berkeley in 2009.

It became open source in 2010.

In 2013, it was donated to Apache Software Foundation.

In 2016, Spark released TensorFrame for Deep learning

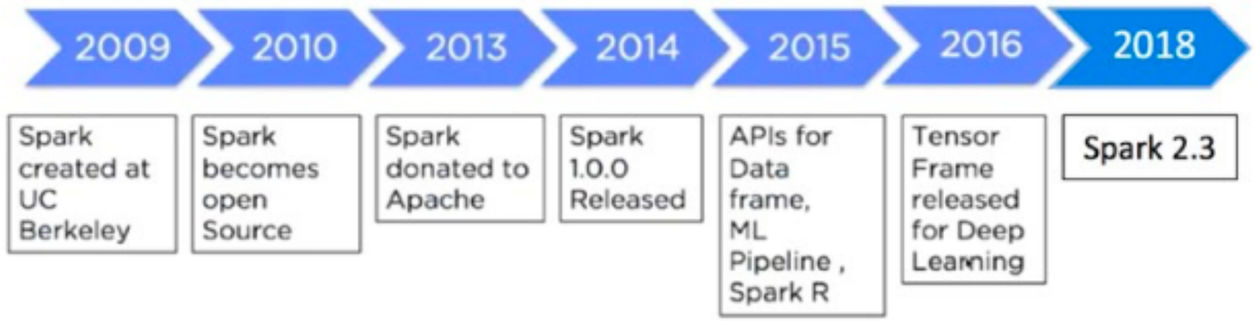


Figure 1-4. *Spark Evolution*

Spark Architecture

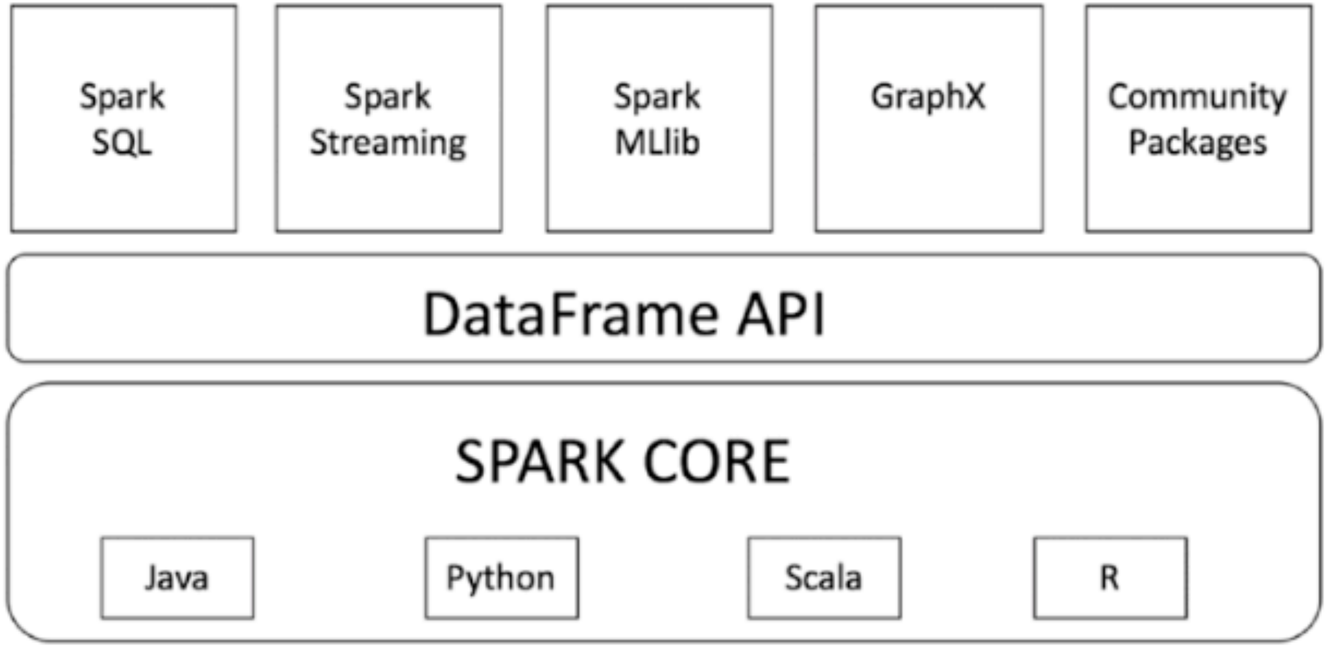


Figure 1-6. *Spark Architecture*

Spark Architecture

SparkCore: fundamental building block of Spark. It is the backbone of Spark's functionality. All other features are built on top it. It enables the parallel and distributed processing of data.

SparkSQL: Deals with structured data. Can be considered a distributed SQL query engine.

SparkStreaming: deals with processing real-time streaming data in scalable and fault-tolerant manner. It uses micro-batching to read and process incoming streams of data.

SparkMLib: library for building machine learning models on Big Data in a distributed manner. MLib has most of algorithms implemented for classification, regression and clustering.

Spark GraphX: performs graph analytics and graph data mining.

Spark RDD

The core element of Spark is an abstraction called **Resilient Distributed Dataset (RDD)**

Resilient : provides fault tolerance by keeping a log of all transformations applied to chunks of data. If things go wrong and a part of data is lost, it can recompute itself.

Distributed: may be stored across different worker machines.

Dataset: it is an immutable list of data.

RDD Transformations

Spark defines a number of transformation that you can apply to RDD to create a new one.

Transformations are executed in a lazy manner. I.e, if you apply the following transformations to input RDD

1) count the occurrence of distinct values in a certain column
2) Select those that start with letter ‘A’
3) Print the results to the screen

Spark execution could only count the occurrence of distinct values that start with letter A.

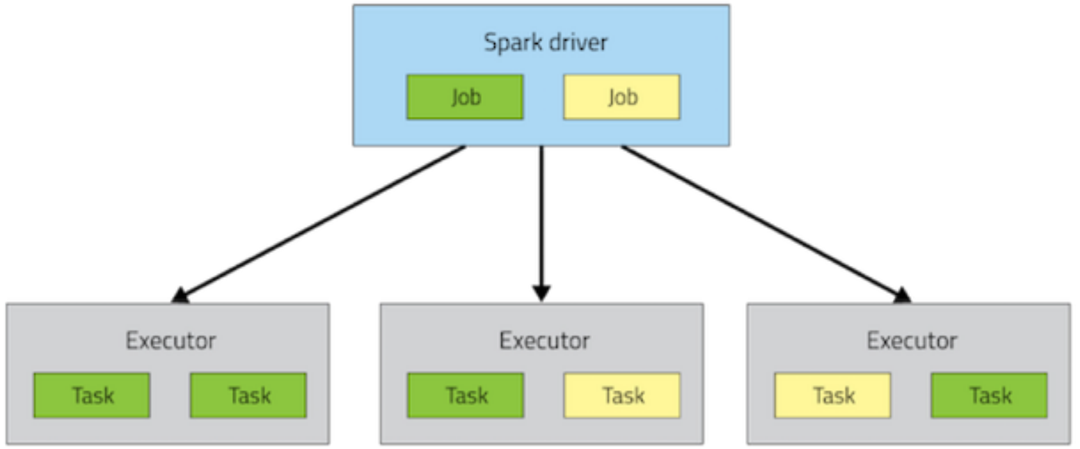
Spark Execution

Spark is distributed framework. It works on master and worker node settings.

At runtime, the code you write is mapped to a single ***driver*** and a set of ***executer*** processes distributed across hosts in a cluster.

The driver manages the job flow and schedules tasks.

Executors perform the work and store any results in its memory.



PySpark

Apache Spark is written in Scala programming languages.

You can write applications using Spark in Scala, Java, R, or Python

PySpark is an interface for Apache spark in Python.

https://spark.apache.org/docs/latest/api/python/

SparkContext

- **SparkContext** is the main entry point to Spark functionality. It represents a connection to Spark Cluster.
- You can use it to create RDDs.
- Only one SparkContext can be active at time per JVM. You must call .stop() before creating another one.

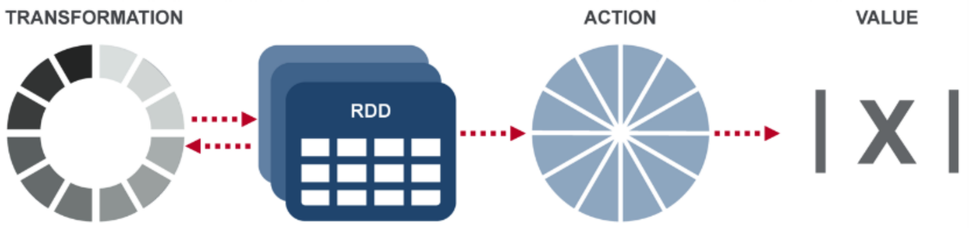
Creating RDD

Two main ways to create RDD

Either by calling `sc.parallelize(…)` given a list of elements

Or by reading data from file
`sc.textFile(….)`

RDD Operations



Transformation: returns a new RDD.

Nothing gets evaluated when you call a Transformation function, it just takes an RDD and return a new RDD.

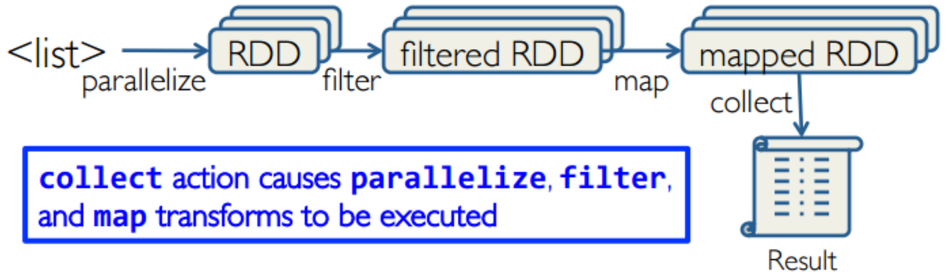
Transformation functions include map, filter, flatMap, groupByKey, reduceByKey, aggregateByKey, filter, join, etc.

Action: evaluates and returns a new value.

When an Action function is called on a RDD object, all the data processing queries are computed at that time and the result value is returned.

Action operations include reduce, collect, count, first, take, countByKey, foreach, saveAsTextFile, etc.

RDD Operations



Create an RDD from a data source

- by parallelizing existing Python collections (lists)
- by transforming an existing RDDs
- from files in HDFS or any other storage system

Apply transformations to an RDD: e.g., map, filter

Apply actions to an RDD: e.g., collect, count

RDD Transformation

Create new datasets from an existing one

Use lazy evaluation: results not computed right away – instead Spark remembers set of transformations applied to base dataset

Spark optimizes the required calculations

Transformation	Description
`map(`*`func`*`)`	return a new distributed dataset formed by passing each element of the source through a function *func*
`filter(`*`func`*`)`	return a new dataset formed by selecting those elements of the source on which *func* returns true
`distinct([`*`numTasks`*`]))`	return a new dataset that contains the distinct elements of the source dataset
`flatMap(`*`func`*`)`	similar to map, but each input item can be mapped to 0 or more output items (so *func* should return a Seq rather than a single item)

RDD Actions

Actions execute scheduled task on the data.

`.collect()` : returns all elements of the RDD to the driver. This may be impractical for huge datasets.

`.take(..)`: returns the top n rows of RDD to the driver.

`.reduce()`: reduces the elements of an RDD using a specific method and returns the results to the driver.

`.count()` : returns the number of elements in RDD.

`.saveAsTextFile(…)`: saves the RDD to text file.