DASCI

Instituto Andaluz Interuniversitario
en Ciencia de Datos e
Inteligencia Computacional

Ciencia de Datos a través del Big Data

Diego García (djgarcia@ugr.es)
Isaac Triguero (isaaktriguero@ugr.es)

Financiado por
la Unión Europea
NextGenerationEU

GOBIERNO
DE ESPAÑA

MINISTERIO
PARA LA TRANSFORMACIÓN DIGITAL
Y DE LA FUNCIÓN PÚBLICA

SECRETARÍA DE ESTADO
DE DIGITALIZACIÓN
E INTELIGENCIA ARTIFICIAL

Plan de
Recuperación,
Transformación
y Resiliencia

UNIVERSIDAD
DE GRANADA

UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

REPSOL

CAMBRIDGE UNIVERSITY PRESS

Chapter 3

Hadoop II

Learning Outcomes

- How a big data application is distributed in Hadoop [KU]
- How a distributed file systems enables data locality for big data processing [KU]
- How transparency and fault tolerance are implemented in Hadoop [KU]
- Limitations of Hadoop for different data processing approaches [KU, IS]

Recap

- Hadoop is an open-source **framework** that implements the MapReduce paradigm
- It is composed of different layers:
  - The MapReduce implementation
  - Resource manager - YARN
  - HDFS – data locality
- Executing a MapReduce program involved multiple processes coordinating everything to ensure transparency and tolerance to faults

Objective for today

- Hadoop Distributed File System
- Limitations
- The Hadoop Ecosystem

Recap: What is a filesystem?

- A system that controls how data is stored and retrieved from a hard or solid-state drive
- Layout of the disk was important to reduce I/O on hard drives
- A disk is split into a number of blocks
- Many different implementations as to how disk blocks are allocated to files exist (e.g., Linked List, FAT32, i-nodes, NTFS)
- The aim of these implementations is to provide efficient and resilient read and write from a single drive

The Hadoop File System – What is it?

- **Hadoop Distributed File System (HDFS)** is a scalable and flexible distributed file system, written in Java for Hadoop.
- **Shared-nothing cluster** of thousand nodes, built from inexpensive hardware → node failure is the norm!
- Network-based filesystem → all the complications of network programming kick in, thus making distributed filesystems more complex than regular disk filesystems
- Preventing data loss is more challenging!! Achieved via replication, and more recently with **erasure coding**
- **This implements the principle of data locality needed in big data!**

The Hadoop File System – Key Design principles

- Very large files, of typically several GB, containing many objects
- A particular **streaming data access** pattern:
  - Write-once, read-many-times pattern.
  - Mostly read and append operations (random updates/access are rare)
  - High throughput (for bulk data) more important than low latency
- *For commodity (low-cost) hardware*
- Not OS integrated, but built on top of your File System

The Hadoop File System – Not For

- Data access with **low latency** (HBase might be a better choice for that)
- Access **many small files**
  - The metadata of all files is stored in the memory of the `NameNode` (in a single node!)
  - Max number of files determined by the memory of the `NameNode` (*HDFS Federation helps with that*)
- Multiple **concurrent writes** or 'arbitrary' modifications
  - It only allows a single access to write
  - It only allows to add data at the end of a file

The Hadoop File System – Fault tolerance mechanisms

- **Replication** (default 3x): data is available in different nodes to provide fault tolerance. This incurs a 200% overhead in storage space and other resources (e.g., bandwidth when writing data)
- **Erasure coding** uses far less storage (approx. by 50% compared with 3x replication == 1.5x)
  - Erasure coding is a branch of information theory which extends a message with redundant data for fault tolerance
  - Implementing this in Hadoop is not trivial! Requires changes across many parts of the HDFS

https://blog.cloudera.com/introduction-to-hdfs-erasure-coding-in-apache-hadoop/

HDFS – Basic Concepts

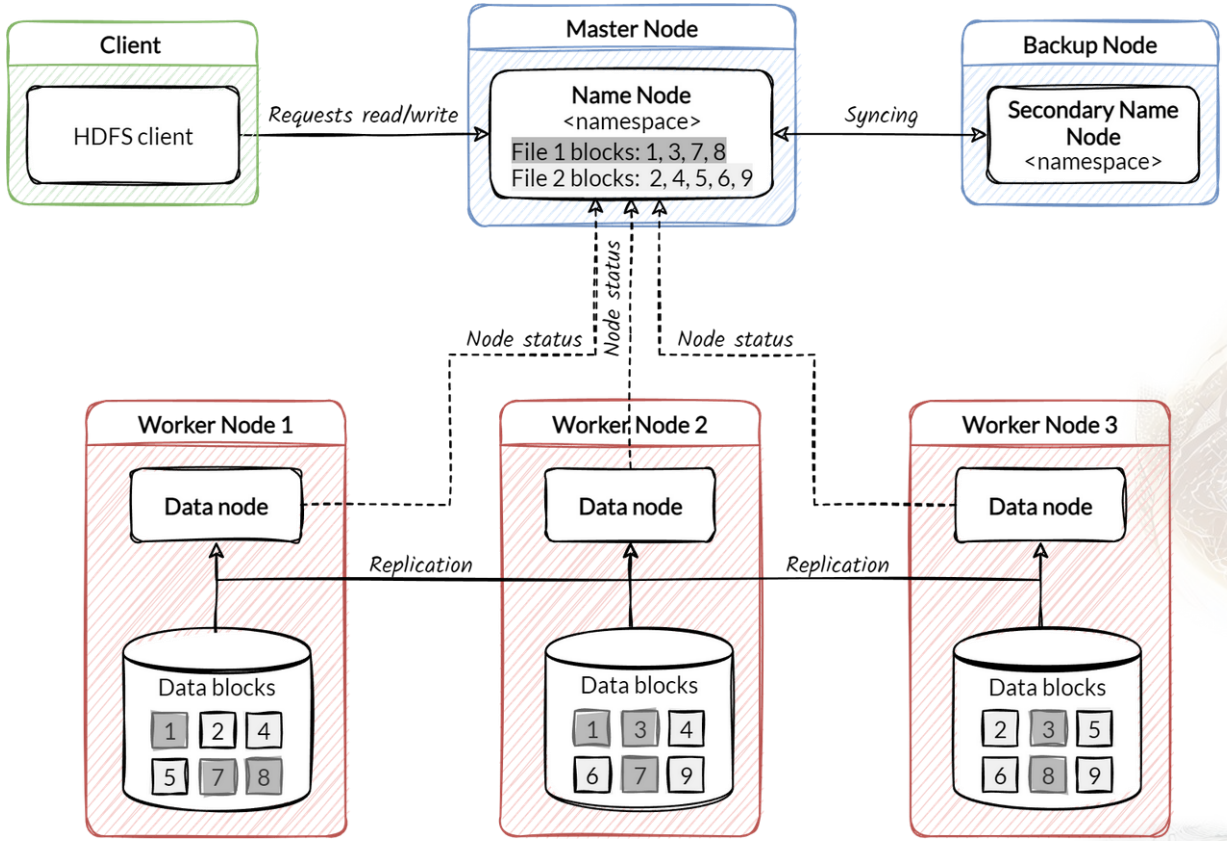
- A **block** in HDFS
  - Minimum data unit that it can read or write
  - In traditional filesystems the block size is usually small, i.e., 512 bytes, and it is transparent to the user
- In HDFS, blocks are bigger (64MB by default, or 128MB)
  - Files are also broken into block-sized chunks stored independently
  - In HDFS, a 1MB file will not take out 64/128MB of memory
- Why is the block size so large in HDFS?
  - Aim to minimise seek time

HDFS – Basic Concepts

- The concept of **block** is also important in HDFS. **But why?**
  - Allows us to store files that **are larger than a single disk!**
    - There is nothing stopping us from storing the blocks of a file on different disks/machines (enabling the **data locality**)
  - **Simplifies storage management**
    - Allows separation of metadata and data blocks
    - Free space
  - Help with the idea of **replication** for providing fault tolerance and availability
    - Typically, each block is replicated 3 times in different machines

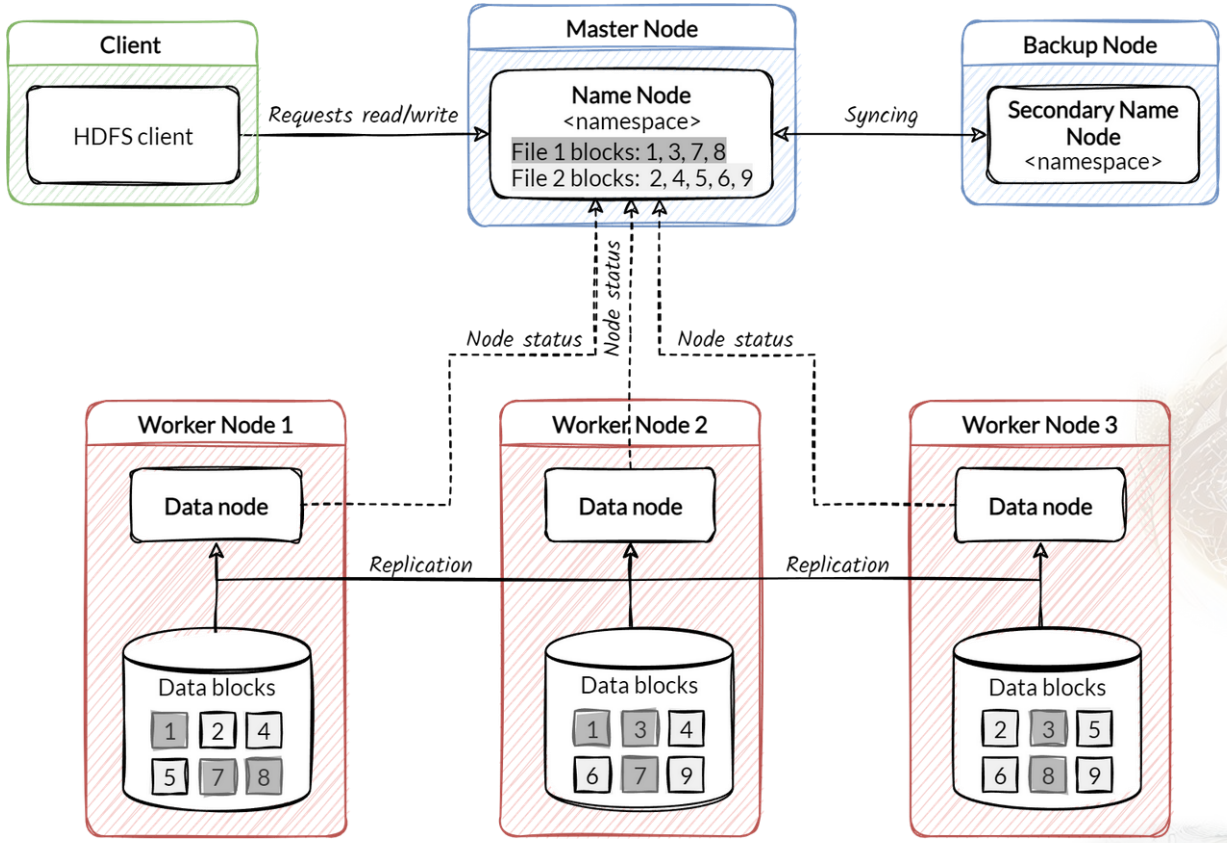
HDFS – Basic Concepts

- Operate in a Master-worker pattern
  - **`Name Node`**: Filesystem namespace and data block locations (not persistently)
    - Secondary `Name Node` for resiliency!
  - **`Data Nodes`**: store and retrieve data blocks when asked to
    - Send Heartbeats to `Name Node`



HDFS – Basic Concepts

- A client process allows users to communicate with the `Name Node`.
  - It offers a POSIX-like interface to hide away the implementation details (`Name Node`, `Data Nodes`)
  - But not integrated with the Unix Virtual File System ☹ and no tab-completion ☹☹☹

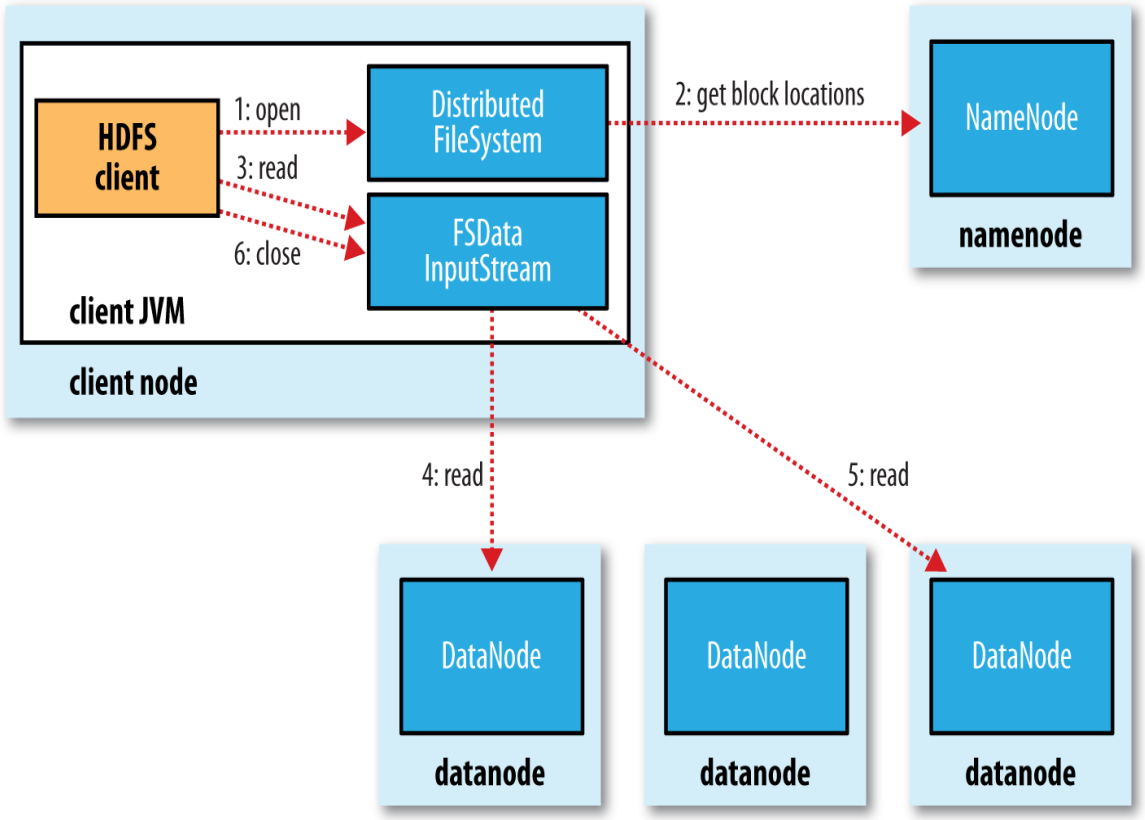


HDFS – Differences between v1 and v2

- The `Name Node` was the single point of failure
  - Costly to start the secondary name node
  - Needed the SysAdmin to act (> 30 minutes)
- From v2:
  - They added HDFS federation and HDFS High-Availability
  - Federation allows for multiple independent Name Spaces
  - High Availability means continuous sync between primary and secondary `Name Nodes`
  - System recovery in seconds
- From v3: erasure coding to reduce data replication

HDFS – Reading from HDFS

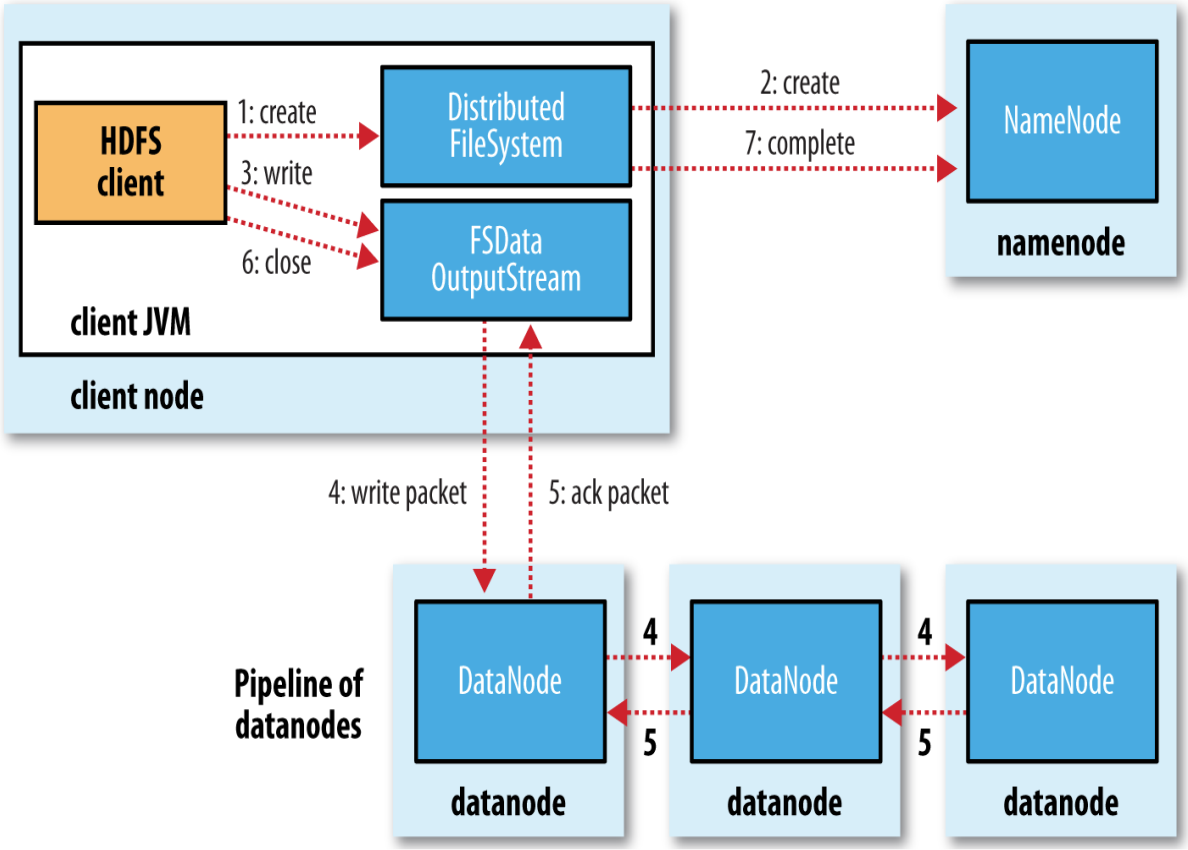
- **Open file**: Client requests the `Name Node` the location of the blocks (1,2 in the figure)
- **Read data blocks**: Client reads the file (3 to 5)
  - For each data block, the nearest `Data Node` is chosen (might be same worker)
- **Close file**



A client reading data to HDFS (From Hadoop Definite Guide)

HDFS – Writing to HDFS

- **Create file**: Client requests the Name Node to create a file. Namespace is created without blocks (steps 1 and 2).
- **Write data blocks**: One by one
  - Client starts writing on a Data Node
  - Propagated across node for data replication
- Name Node waits for **final confirmation**



A client writing data to HDFS (From Hadoop Definite Guide)

HDFS – Working with it

- Basic Operations:

`hadoop fs -ls <path>`	List files
`hadoop fs -cp <src> <dst>`	Copy files from HDFS to HDFS
`hadoop fs -mv <src> <dst>`	Move files from HDFS to HDFS
`hadoop fs -rm <path>`	Remove files in HDFS
`hadoop fs -rmr <path>`	Remove recursively in HDFS
`hadoop fs -cat <path>`	Show the content of a file in HDFS
`hadoop fs -mkdir <path>`	Make a folder HDFS
`hadoop fs -put <localsrc> <dst>`	Copy files from Local to HDFS
`hadoop fs -copyToLocal <src> <localdst>`	Copy files from HDFS to local.
Also:	
`hadoop fs -get <src>`	

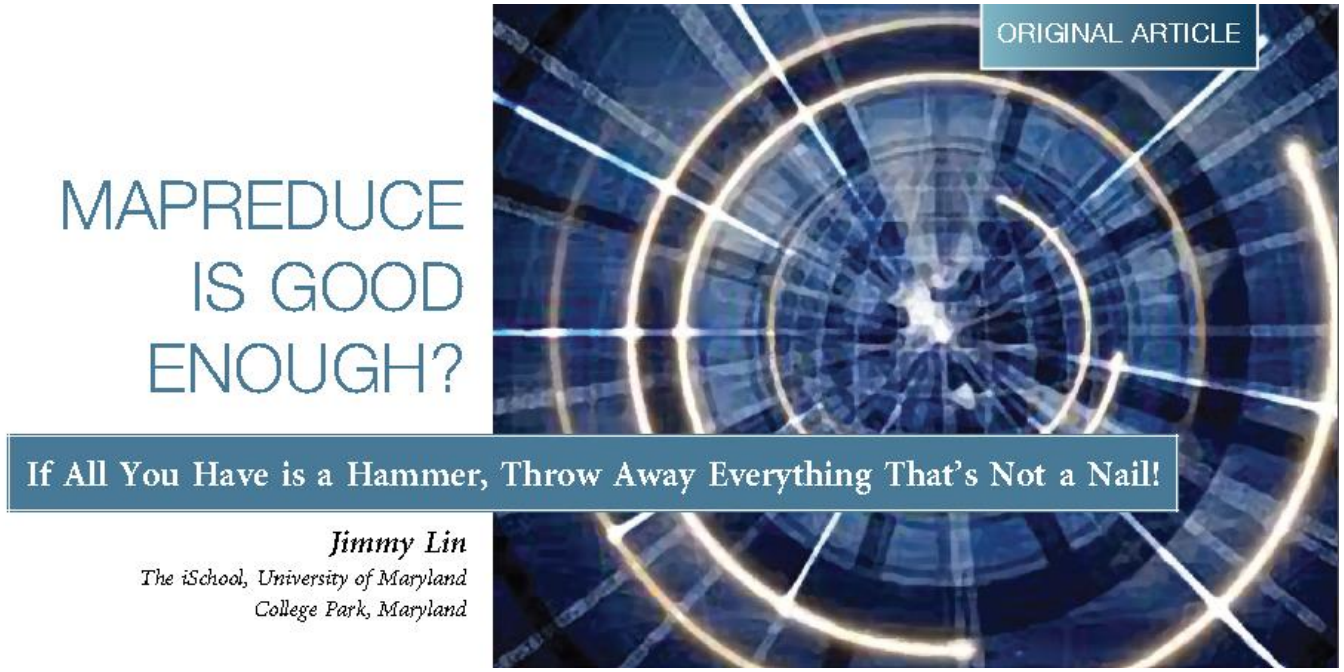
Hadoop MapReduce: summary

- **Advantages compared to classical distributed models**
  - Simplicity and fault tolerant mechanism!
- **Main keys**
  - **Scalable**: no matter about underlying hardware
  - **Cheaper**: Hardware, programming and administration savings!

- **WARNING**: MapReduce could not solve any kind of problems, BUT when it works, it may save a lot time!

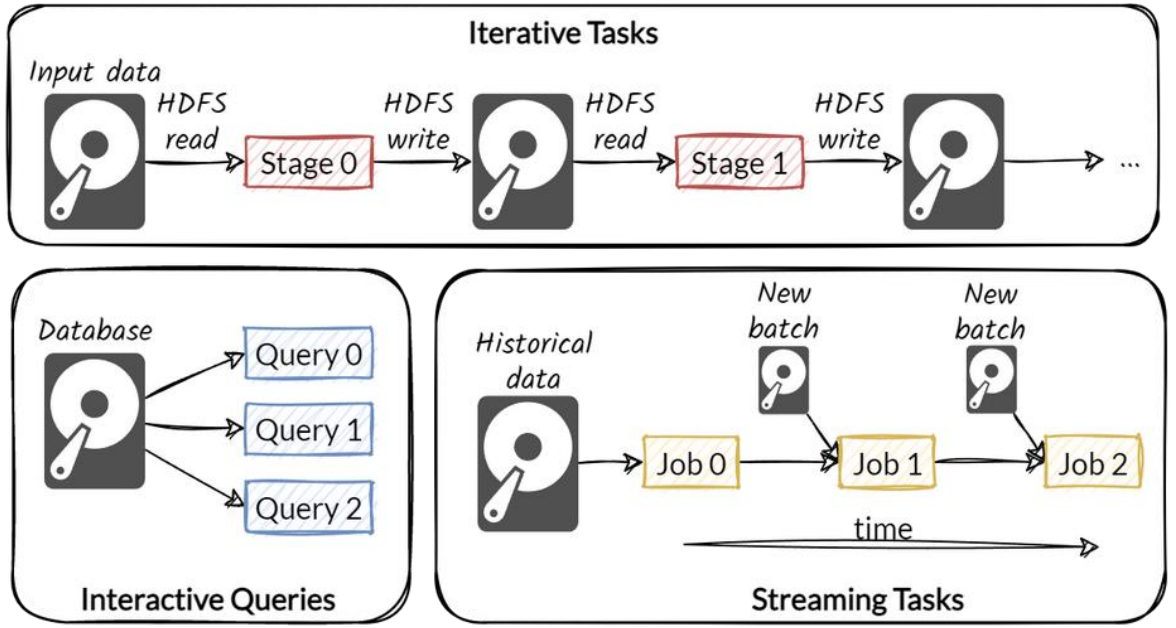
Hadoop MapReduce: Limitations

"If all you have is a hammer, then everything looks like a nail."



Hadoop MapReduce: Limitations

- Hadoop simplified big data processing, but:
  - **Cumbersome Programming style**
  - **Speed**: Inefficient for apps that share data across multiple steps, e.g., Iterative algorithms,Graph-models, interactive queries

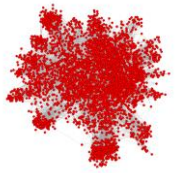


Slow due to replication, serialization, and disk IO

Inpired by the Apache Spark Documentation

New platforms to overcome Hadoop's limitations

GIRAPH (APACHE Project)
(http://giraph.apache.org/)
Iterative Processing of Graphs



GPS - A Graph Processing System, (Stanford)
http://infolab.stanford.edu/gps/
para Amazon's EC2

Distributed GraphLab (Carnegie Mellon Univ.) *https://github.com/graphlab-code/graphlab*
Amazon's EC2

Spark (UC Berkeley)
(Apache Foundation)
http://spark.incubator.apache.org/research.html

Twister (Indiana University)
http://www.iterativemapreduce.org/
For you own cluster

PrIter (University of Massachusetts Amherst, Northeastern University-China)
http://code.google.com/p/priter/
For your own cluster and Amazon EC2 cloud

HaLoop
(University of Washington)
http://clue.cs.washington.edu/node/14
http://code.google.com/p/haloop/
Amazon's EC2

GPU based platforms

Mars
Grex
GPMR

Hadoop Ecosystem

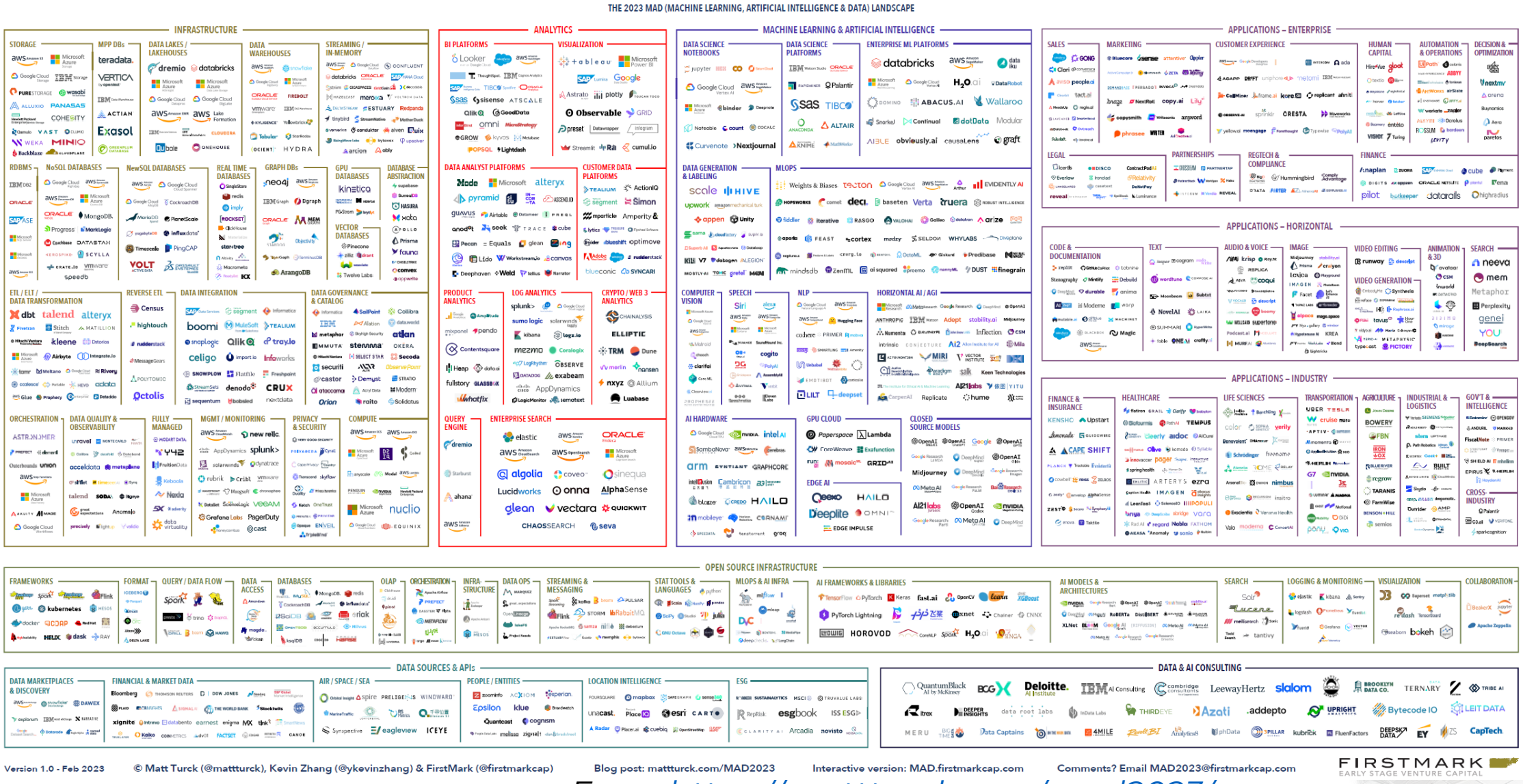
- It's not the end for Hadoop… actually it is the base of everything.

- **Hadoop Data Services**: Tools to manipulate and process data easily
  - Apache Hive
  - Apache Pig
  - Apache HCatalog
  - Apache HBase
  - Apache Sqoop
  - Apache Flume

- **Hadoop Operational Services**: Helpful tools to manage the operations of a Hadoop Cluster
  - Apache Ambari
  - Apache Oozie
  - Apache ZooKeeper

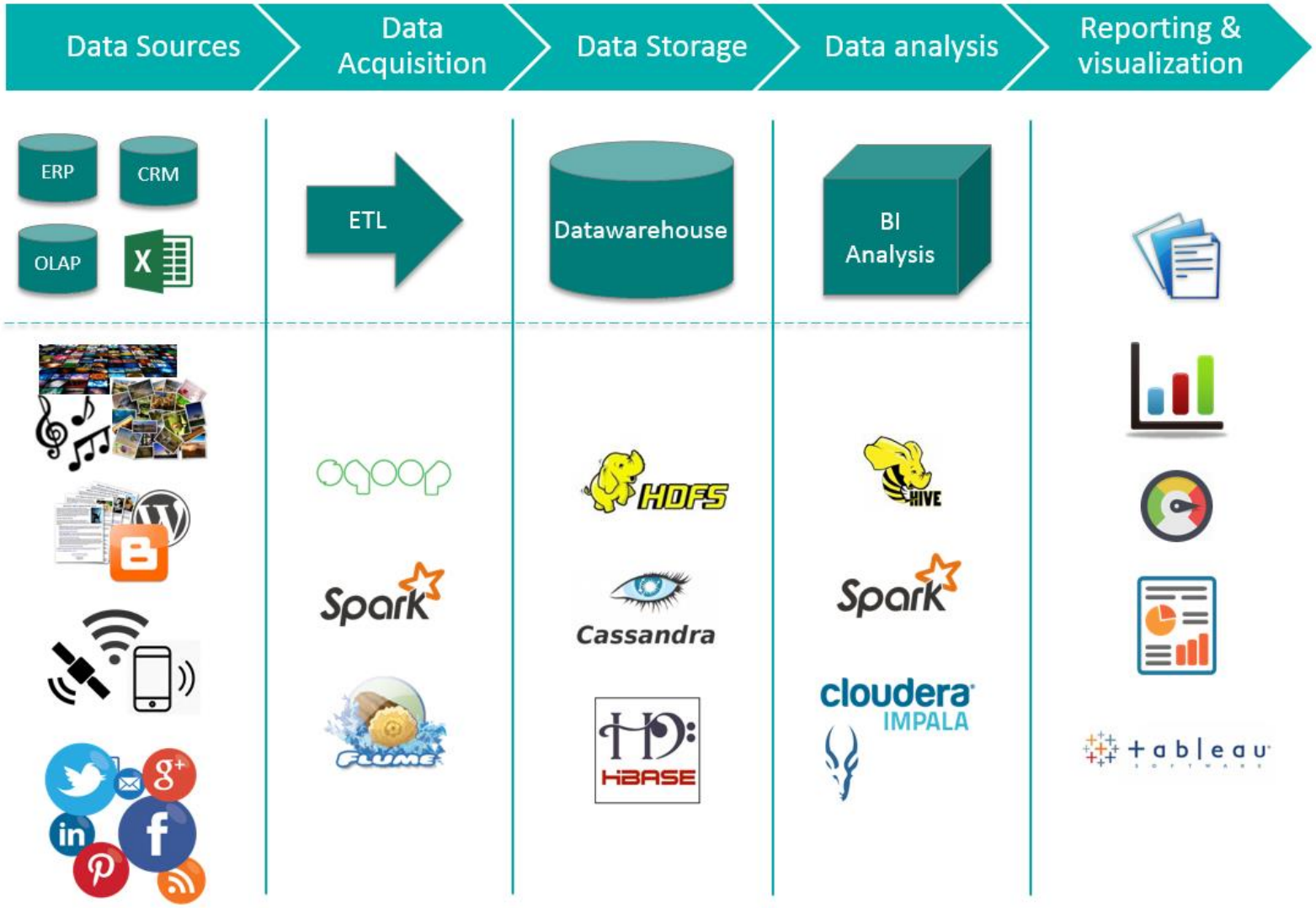
Hadoop Ecosystem

The big data tools landscape is BIG



From https://mattturck.com/mad2023/

Choose depending on the purpose



From clearpeaks.com

Challenge

- Name a Machine Learning algorithm that you believe won't work well within Hadoop MapReduce and explain why.

Take-home message

- The design principles of the HDFS
  - Large files, high-throughput, commodity hardware
- Hadoop MapReduce is not enough (not even for data analytics only)
  - The world of Big Data Technologies is very Big
  - *Choosing the right technology is like choosing the right data structure in a program*

What's next?

- Apache Spark as a central tool for mining and analytics in big data

To Learn More

- **Apache Hadoop**
  - Hadoop: The Definitive Guide (White, 2015)
  - Apache Hadoop website
  - Erasure coding (Xia et al., 2015)
- **Other concepts**
  - Serialization and how it works in Java and Python with pickle
  - Linux Containers and Virtualization (Jain, 2020)

CAMBRIDGE UNIVERSITY PRESS

Chapter 3

Hadoop II