The Hadoop distributed computing architecture

Hadoop for distributed data processing

Hadoop is a framework for running jobs on clusters of computers that provides a good abstraction of the underlying hardware and software.

"*Stripped to its core, the tools that Hadoop provides for building distributed systems—for data storage, data analysis, and coordination—are simple. If there's a common theme, it is about raising the level of abstraction—to create building blocks for programmers who just happen to have lots of data to store, or lots of data to analyze, or lots of machines to coordinate, and who don't have the time, the skill, or the inclination to become distributed systems experts to build the infrastructure to handle it.*[2]"

[2] White T. *Hadoop: The Definitive Guide*. O'Reilly, 1988.

Hadoop: some facts

Hadoop[3] is an open-source project of the Apache Software Foundation. The project was created to facilitate computations involving massive amounts of data.

- its core components are implemented in Java
- initially released in 2006. Last stable version is 3.3.1 from June 2021
- originally inspired by Google's MapReduce[4] and the proprietary GFS (Google File System)

[3] Apache Software Foundation. *Hadoop*. url: https://hadoop.apache.org.

[4] J. Dean and S. Ghemawat. "MapReduce: Simplified data processing on large clusters." In: *Proceedings of Operating Systems Design and Implementation (OSDI)*. 2004. url: https://www.usenix.org/legacy/publications/library/proceedings/osdi04/tech/full_papers/dean/dean.pdf.

Hadoop's features

Hadoop's features addressing the challenges of Big Data:

- scalability
- fault tolerance
- high availability
- distributed cache/data locality
- cost-effectiveness as it does not need high-end hardware
- provides a good abstraction of the underlying hardware
- easy to learn
- data can be queried trough SQL-like endpoints (Hive, Cassandra)

Mini-glossary of Hadoop's distinguishing features

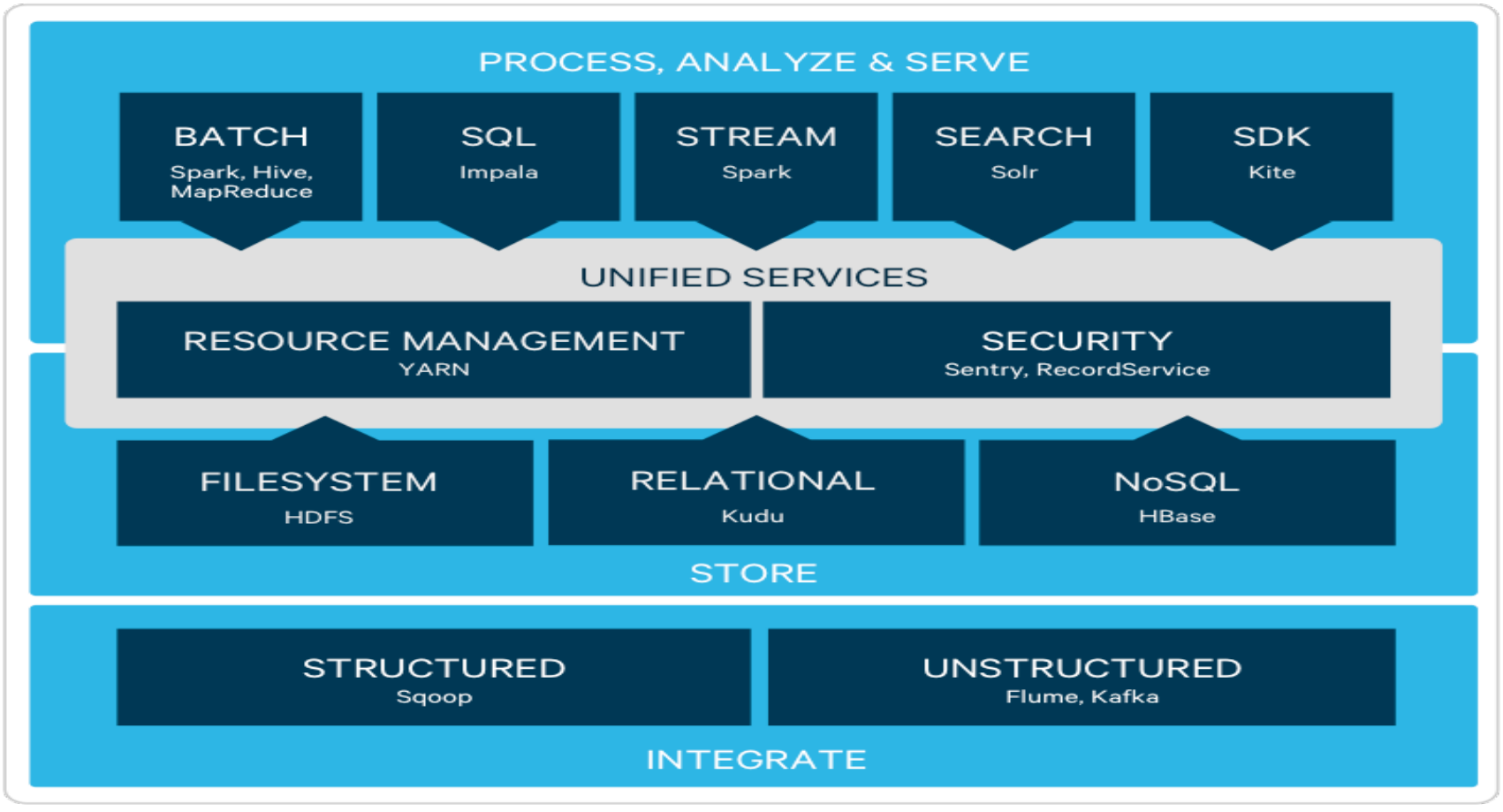
- ***fault tolerance***: the ability to withstand hardware or network failures (also: *resilience*)
- ***high availability***: this refers to the system minimizing downtimes by eliminating single points of failure
- ***data locality***: task are run on the node where data are located, in order to reduce the cost of moving data around

The Hadoop core

The core of Hadoop consists of:

- Hadoop common, the core libraries
- HDFS, the Hadoop Distributed File System
- MapReduce
- the YARN (Yet Another Resource Negotiator) resource manager

The Hadoop ecosystem



There's a whole constellation of open source components for collecting, storing, and processing big data that integrate with Hadoop.

Image source: Cloudera

The Hadoop Distributed File System (HDFS)

HDFS stands for Hadoop Distributed File System and it takes care of partitioning data across a cluster.

In order to prevent data loss and/or task termination due to hardware failures HDFS uses either

- replication (creating multiple copies —usually 3— of the data)
- erasure coding

Data redundancy (obtained through replication or erasure coding) is the basis of Hadoop's fault tolerance.

Replication vs. Erasure Coding

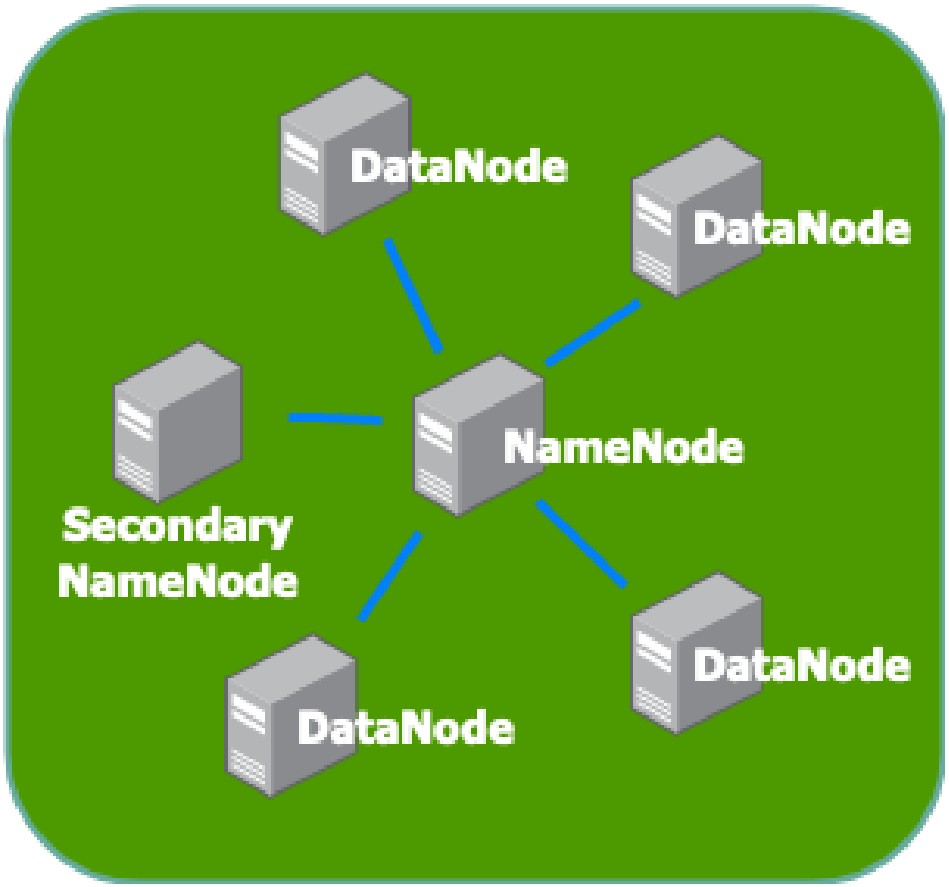
In order to provide protection against failures one introduces:

- data redundancy
- a method to recover the lost data using the redundant data

Replication is the simplest method for coding data by making n copies of the data. n-fold replication guarantees the availability of data for at most $n-1$ failures and it has a storage overhead of 200% (this is equivalent to a storage efficiency of 33%).

Erasure coding provides a better storage efficiency (up to to 71%) but it can be more costly than replication in terms of performance.

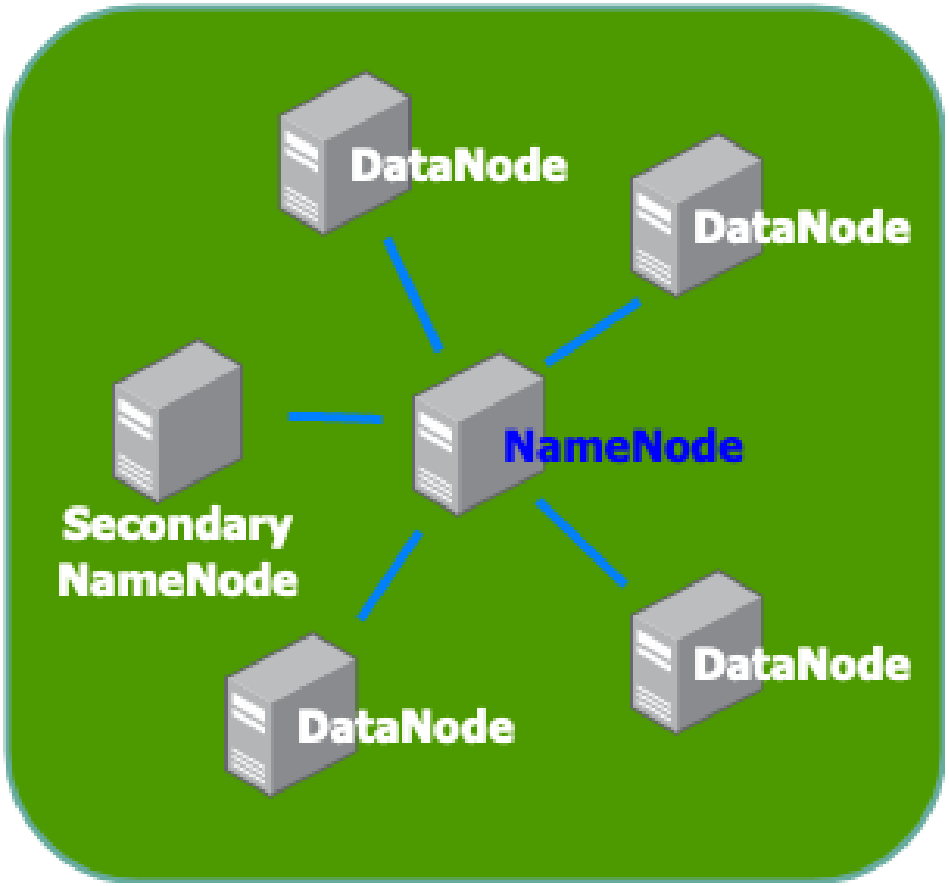
HDFS architecture



A typical Hadoop cluster installation consists of:

- a NameNode
- a secondary NameNode
- multiple DataNodes

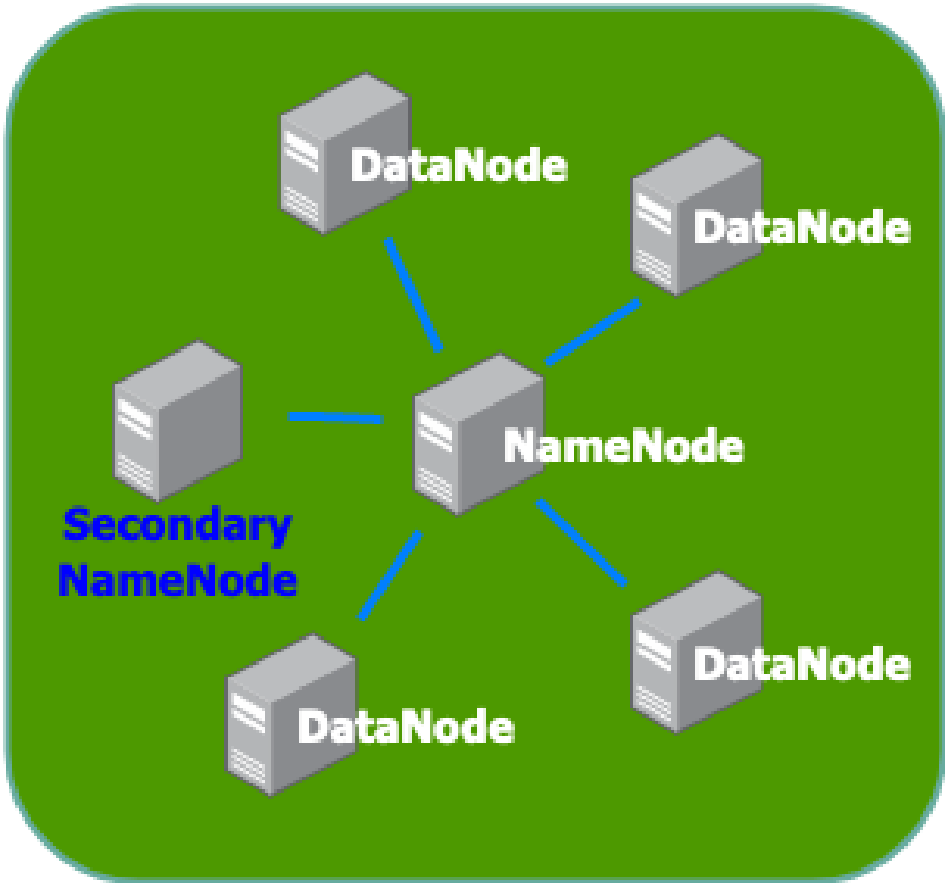
HDFS architecture: NameNode



NameNode

The NameNode is the main point of access of a Hadoop cluster. It is responsible for the bookkeeping of the data partitioned across the DataNodes, manages the whole filesystem metadata, and performs load balancing

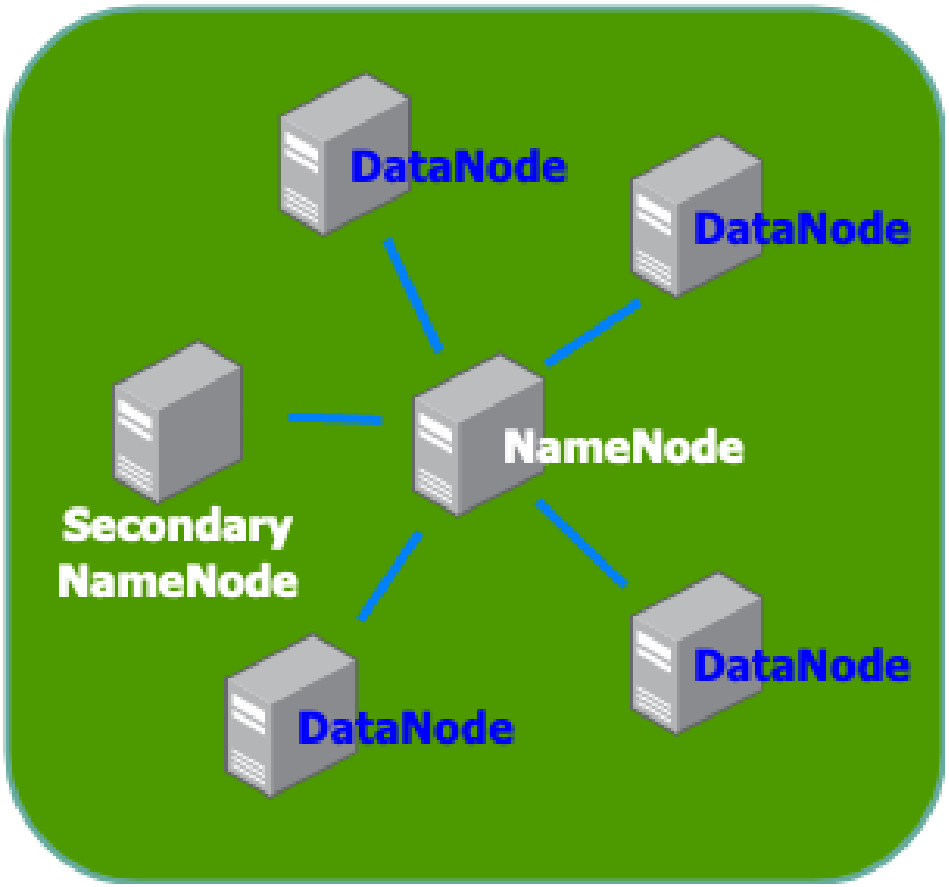
HDFS architecture: Secondary NameNode



Secondary NameNode

Keeps track of changes in the NameNode performing regular snapshots, thus allowing quick startup.
An additional *standby node* is needed to guarantee high availability (since the NameNode is a single point of failure).

HDFS architecture: DataNode



DataNode

Here is where the data is saved and the computations take place (data nodes should actually be called "data and worker nodes")

HDFS architecture: internal data representation

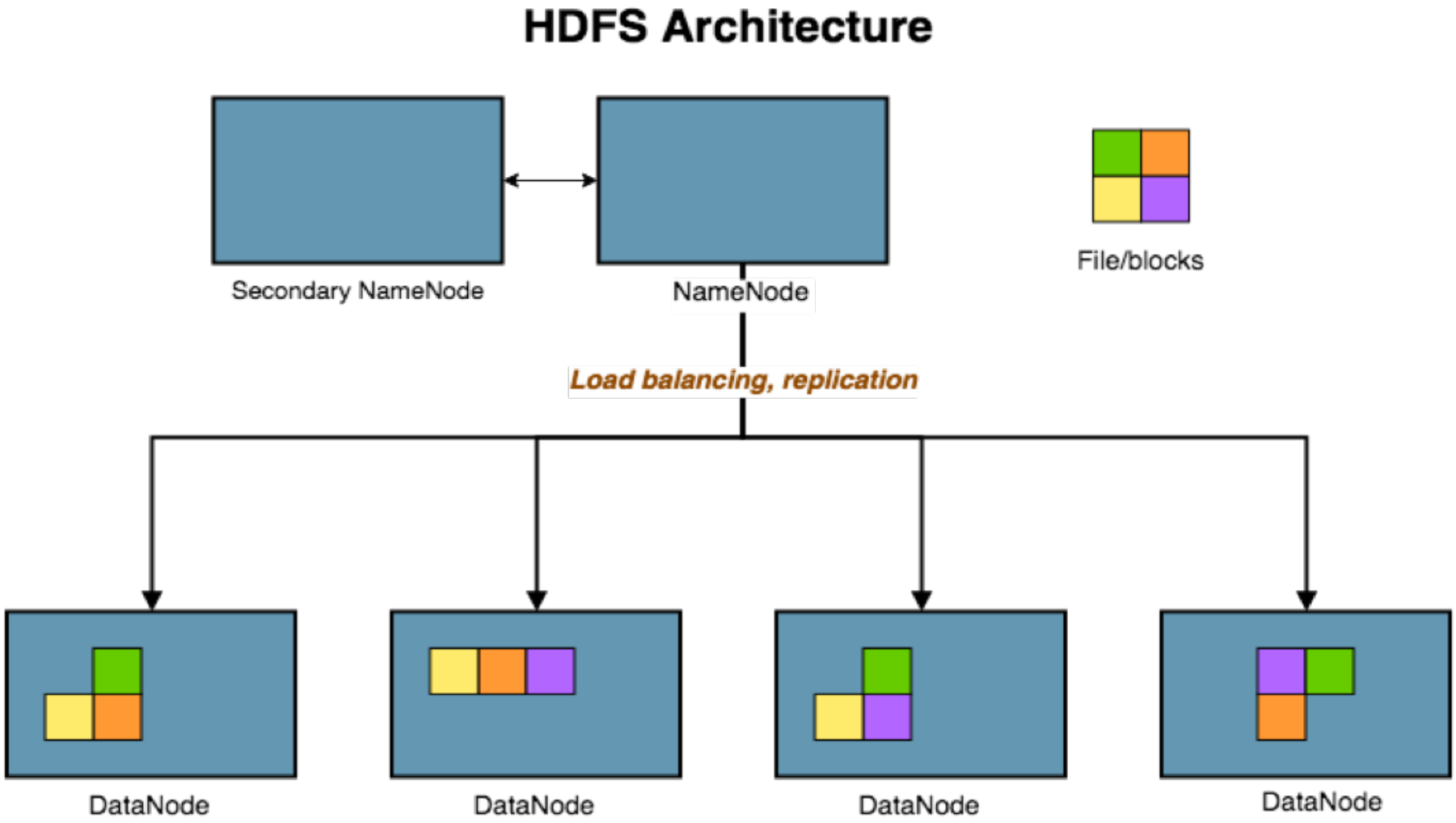
HDFS supports working with very large files.

Internally, data are split into *blocks*. One of the reason for splitting data into blocks is that in this way block objects all have the same size.

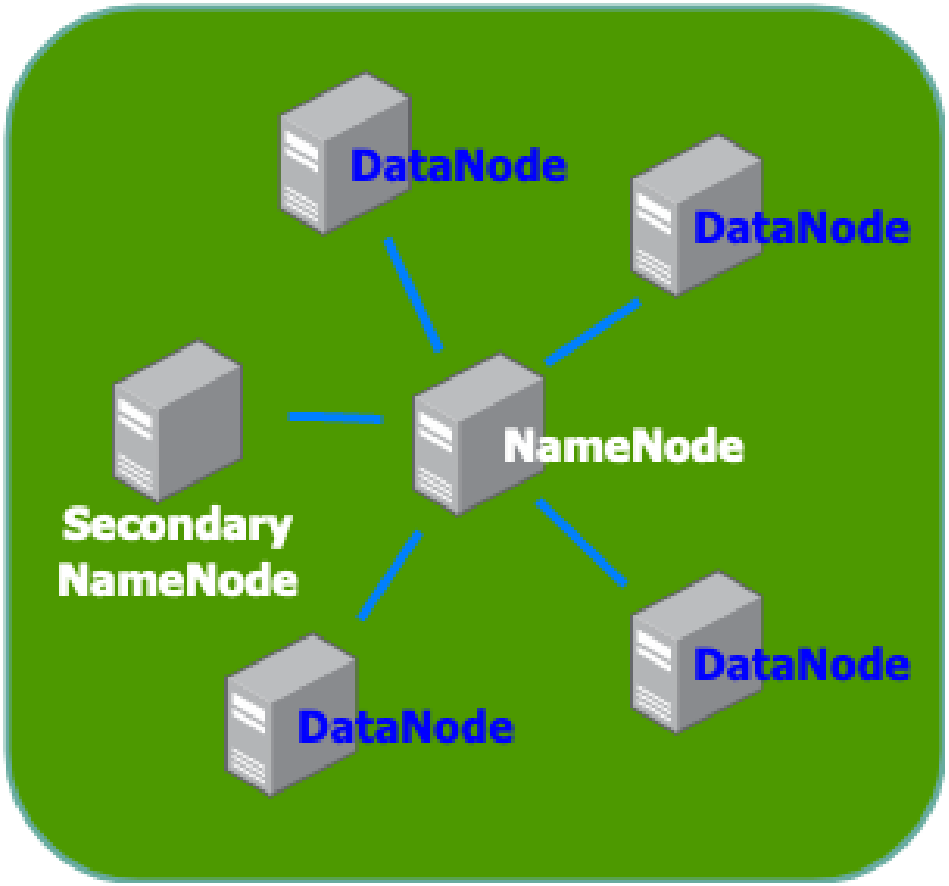
The block size in HDFS can be configured at installation time and it is by default 128MiB (approximately 134MB).

Note: Hadoop sees data as a bunch of records and it processes multiple files the same way it does with a single file. So, if the input is a directory instead of a single file, it will process all files in that directory.

HDFS architecture



DataNode failures



Each DataNode sends a Heartbeat message to the NameNode periodically. Whenever a DataNode becomes unavailable (due to network or hardware failure), the NameNode stops sending requests to that node and creates new replicas of the blocks stored on that node.

The WORM principle of HDFS

The Hadoop Distributed File System relies on a simple design principle for data known as Write Once Read Many (WORM).

"*A file once created, written, and closed need not be changed except for appends and truncates. Appending the content to the end of the files is supported but cannot be updated at arbitrary point. This assumption simplifies data coherency issues and enables high throughput data access.*[5]"

The data immutability paradigm is also discussed in Chapter 2 of "Big Data".[6]

[5] Apache Software Foundation. *Hadoop*. url: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html.

[6] Warren J. and Marz N. *Big Data*. Manning publications, 1988.

Outline/next

- Schedule
- What is Big Data?
- The Hadoop distributed computing architecture
- MapReduce
- HDFS hands-on exercises
- MapReduce hands-on
- The YARN resource manager
- MRjob
- Benchmarking I/O with testDFSio
- Concluding remarks

MapReduce

The origins of MapReduce

The 2004 paper "*MapReduce: Simplified Data Processing on Large Clusters*" by two members of Google's R&D team, Jeffrey Dean and Sanjay Ghemawat, is the seminal article on MapReduce.

The article describes the methods used to split, process, and aggregate the large amount of data for the Google search engine.

The open-source version of MapReduce was later released within the Apache Hadoop project.

MapReduce explained

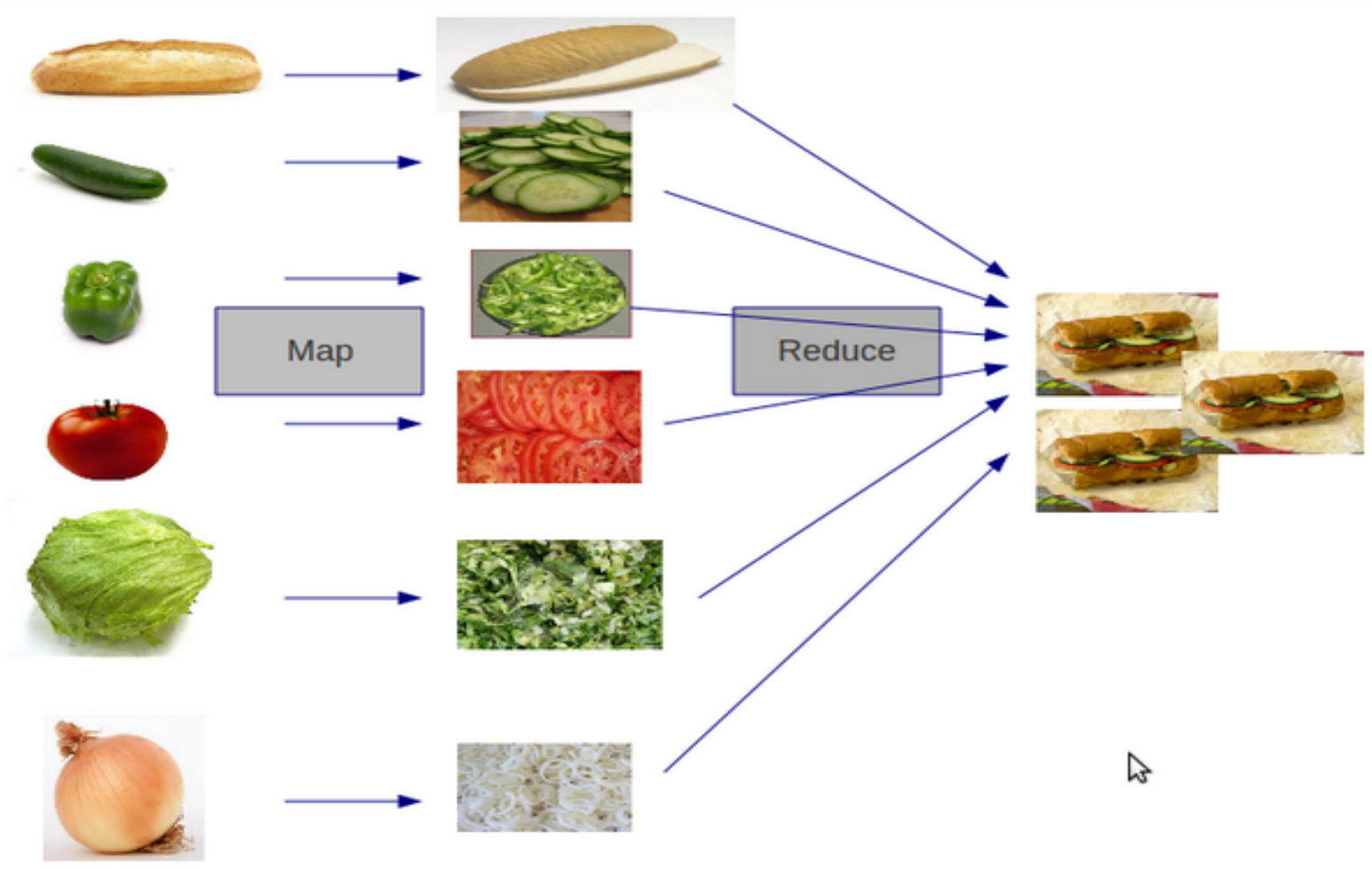


Image source: Stack Overflow

MapReduce explained

The MapReduce paradigm is inspired by the computing model commonly used in functional programming.

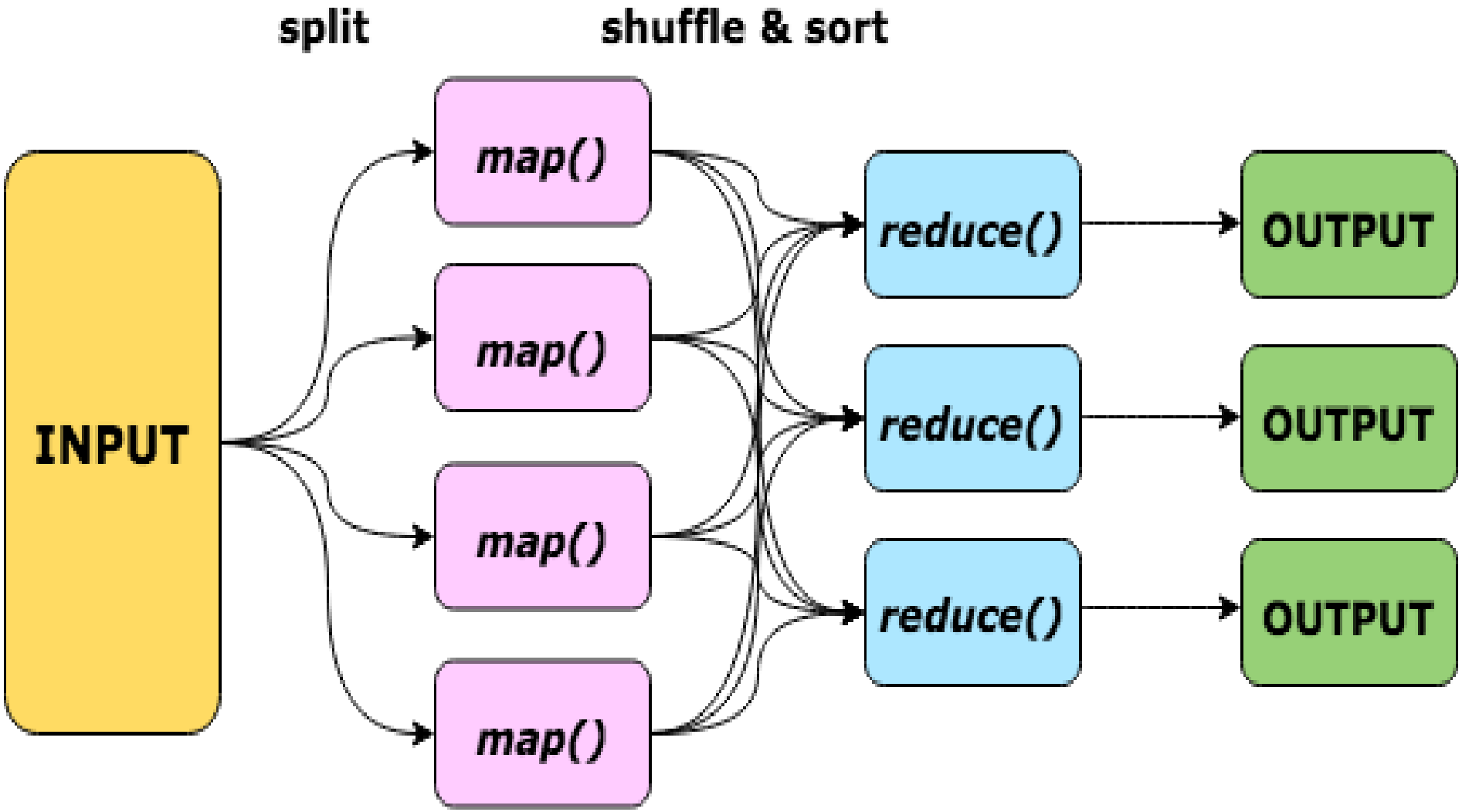
Applying the same function independently to items in a dataset either to transform (*map*) or collate (*reduce*) them into new values, works well in a distributed environment.

The phases of MapReduce

The phases of a MapReduce job:

- **split**: data is partitioned across several computer nodes
- **map**: apply a map function to each chunk of data
- **sort & shuffle**: the output of the mappers is sorted and distributed to the reducers
- **reduce**: finally, a reduce function is applied to the data and an output is produced

The phases of MapReduce



The phases of MapReduce

We have seen that a MapReduce job consists of four phases:

- **split**
- **map**
- **sort & shuffle**
- **reduce**

While splitting, sorting and shuffling are done by the framework, the map and reduce functions are defined by the user.

It is also possible for the user to interact with the splitting, sorting and shuffling phases and change their default behavior, for instance by managing the amount of splitting or defining the sorting comparator. This will be illustrated in the hands-on exercises.

MapReduce: some notes

Notes

- the same map (and reduce) function is applied to all the chunks in the data
- the map and reduce computations can be carried out in parallel because they're completely independent from one another.
- the split is not the same as the internal partitioning into blocks

MapReduce: shuffling and sorting

The shuffling and sorting phase is often the the most costly in a MapReduce job.

The mapper takes as input unsorted data and emits key-value pairs. The purpose of sorting is to provide data that is already grouped by key to the reducer. This way reducers can start working as soon as a group (identified by a key) is filled.

MapReduce: shuffling and sorting