Big Data

Lecture 2 – Hadoop and its ecosystem: HDFS

Gianluca Quercini

gianluca.quercini@centralesupelec.fr

Centrale DigitalLab, 2021

What you will learn

In this lecture you will learn:

- The main properties of **HDFS**.
- The **main components** of HDFS.
- The role of the **NameNode** and of the **DataNodes**.
- How **replication** is implemented in HDFS.
- What is **high availability** in HDFS.
- How read and write operations are realized in HDFS.

Clusters of computers

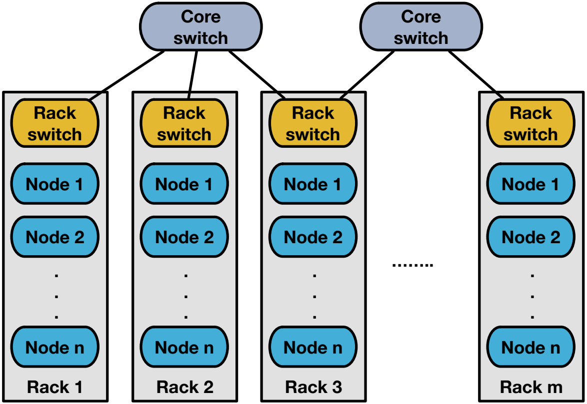
Definition (Cluster)

A **cluster** is a type of parallel or distributed processing system, which consists of a collection of interconnected stand-alone computers (a.k.a., **nodes**) cooperatively working together as a single, integrated computing resource ▸ Source

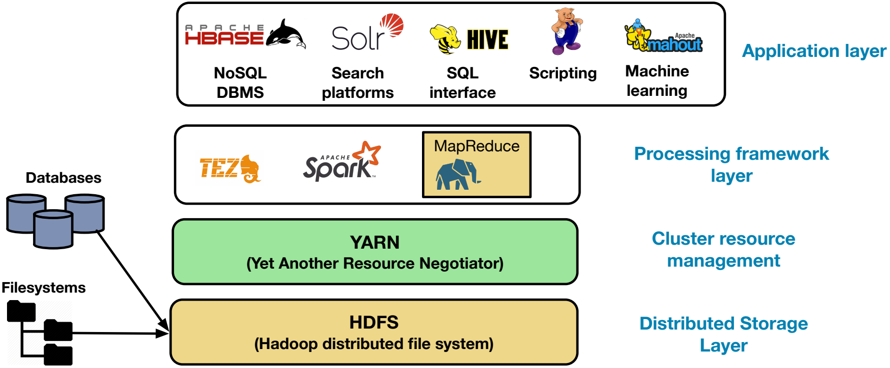
Properties

- Clusters are built with **commodity**, or **off-the-shelf**, hardware.
  - Inexpensive, not necessarily too powerful.
- **Shared-nothing architecture.** Each node has its own CPU, memory and storage.
  - No contention on the resources.
  - Excellent scalability.

Cluster of computers



Hadoop architecture



HDFS: definition and properties

Definition (HDFS)

The **Hadoop Distributed File System (HDFS)** is a *distributed file system* designed to run on *commodity* (i.e., low-cost) hardware. ▸ Source

HDFS characteristics

- **Fault-tolerance.** Detection of **faults** and quick **recovery**.
  - Hardware failure is **frequent** in a computer cluster.
- **Batch processing.** Support for **batch processing** rather than **interactivity**.
- **Big Data.** Support for applications that use large datasets.
  - Support for **large files**.

HDFS: definition and properties

Definition (HDFS)

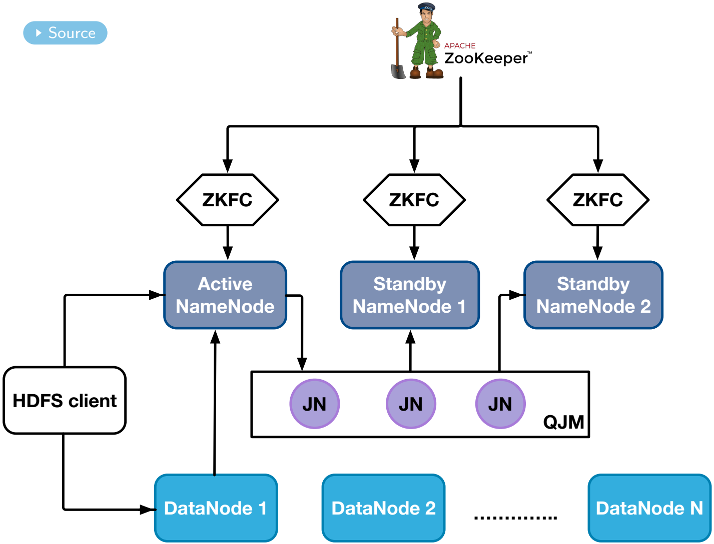
The **Hadoop Distributed File System (HDFS)** is a *distributed file system* designed to run on *commodity* (i.e., low-cost) hardware. ▸ Source

HDFS properties

- **Simple coherency model.** Support for applications that write a file once and read it many times.
  - Files cannot be modified at arbitrary points once they are written.
  - Content can only be **appended** to a file.
- **Computation near data.** Move the processing tasks near the data.
  - Avoids moving large parts of the dataset from one node to another.
- **Portability.** HDFS can be used across different operating systems and hardware.

HDFS logical architecture

Source



NameNode and DataNodes

- A HDFS cluster consists of (**master-slave architecture**):
  - A **master** node, called the **NameNode**.
  - Several **slave** nodes, called **DataNodes**.
  - Both types of nodes are implemented in **Java**.

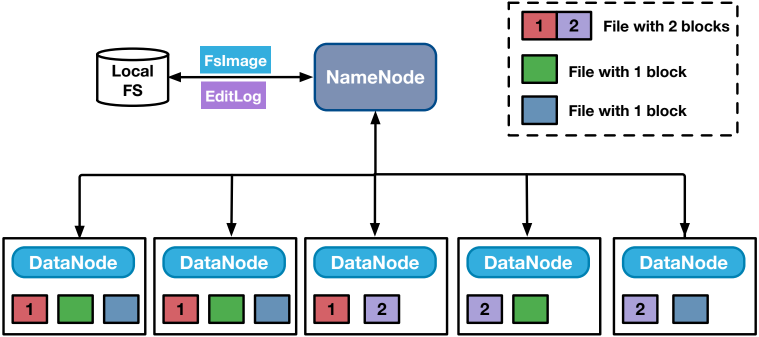
NameNode

- Manages the file system namespace.
  - File metadata (e.g., name, location, last access time...).
- Regulates the access to files by client applications.

DataNodes

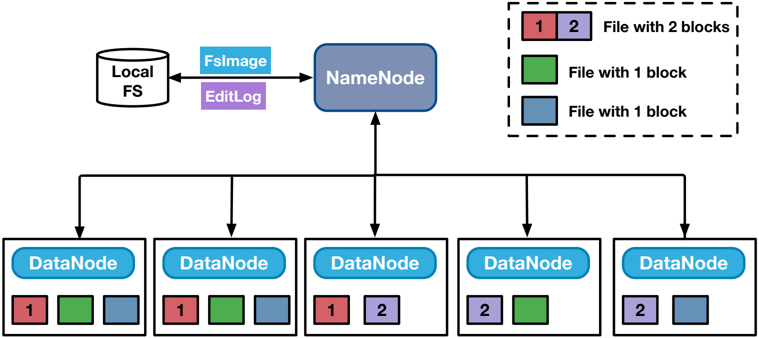
- Manage the storage of the nodes on which they run.
- Typically, there is one DataNode on each node of the cluster.

NameNode and DataNodes



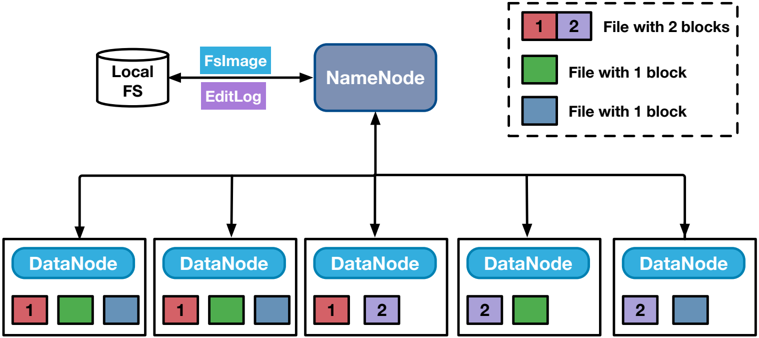
- Each file consists of one or several **blocks**.
  - Typically, the block size is 64 MB or 128 MB.
- **Blocks** are stored in a set of **DataNodes**.
- The **NameNode** determines the mapping of blocks to DataNodes.

NameNode and DataNodes



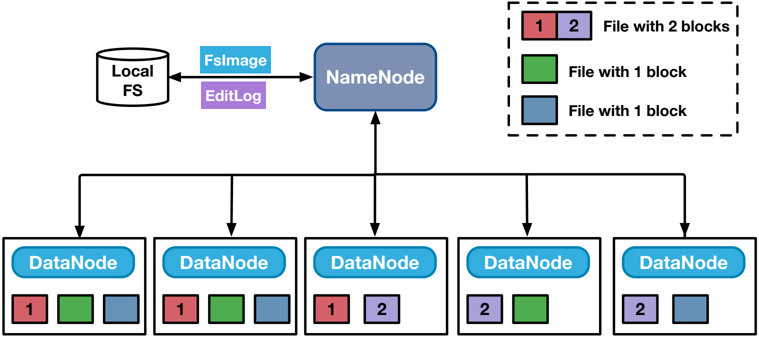
- The **NameNode** executes file system namespace operations.
  - Open, close and rename files and directories.
- **DataNodes** serve read and write requests from the clients.
- **DataNodes** create, delete and replicate blocks based on the **NameNode** instructions.

NameNode and DataNodes



- The **NameNode** stores the entire **file system namespace** in a file called **FsImage**.
- The file **FsImage** includes: mapping of blocks to files, mapping of blocks to DataNodes, last access times...

NameNode and DataNodes



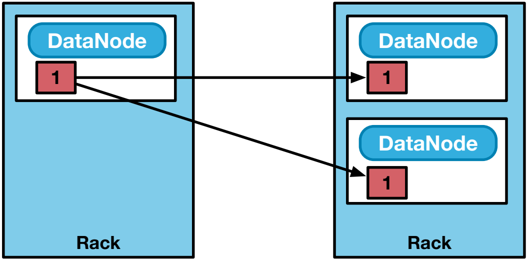
- The **NameNode** keeps the whole content of **FsImage** in RAM.
- Modifications on the FS metadata are recorded in the **EditLog**.
- **Checkpoint.** At regular intervals, the **NameNode** applies all the transactions from the EditLog to the in-memory FsImage and writes it to disk.

Block replication

- Each block is replicated across the machines of the cluster.
- The number of replicas of a block is called the **replication factor**.
  - The typical replication factor in a HDFS cluster is 3.
- Two replicas of the same block are **never** stored in the same DataNode.
  - If the DataNode fails, both replicas are lost.
- If possible, two blocks of the same file are stored on different DataNodes.
- HDFS uses a **rack-aware** replica placement policy.

Rack-aware replica placement

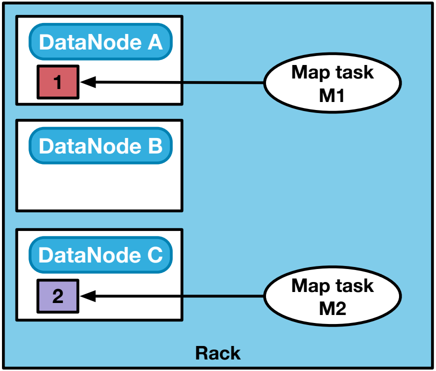
- One replica is stored in a DataNode in the **local rack**.
- Two replicas are stored in different DataNodes of the same **remote rack**.
- Best trade-off between:
  - Storing all the replicas in the **same rack** (not **fault tolerant**).
  - Storing all replicas in different racks (write operations too slow).



Rack awareness in Hadoop MapReduce

Data local

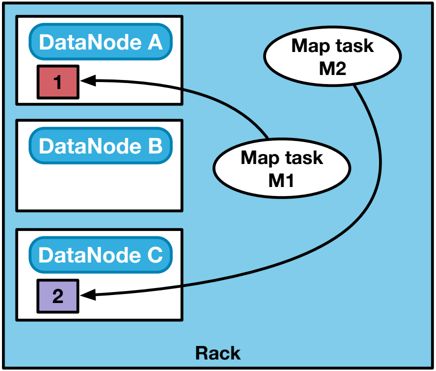
If possible, Hadoop schedules a task on the **same DataNode** where the input data are stored.



Rack awareness in Hadoop MapReduce

Rack local

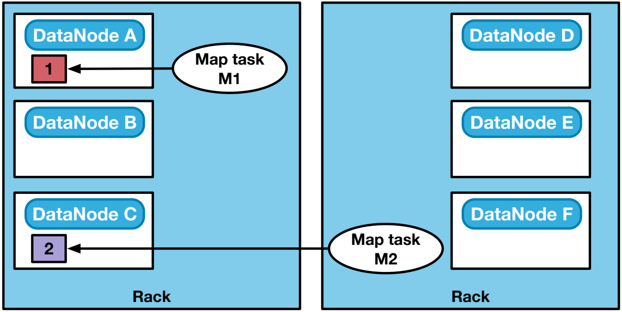
If data locality is not possible, Hadoop schedules a task on a DataNode in the **same rack** as the input data.



Rack awareness in Hadoop MapReduce

Off rack

As the last resort, a task is scheduled in a DataNode **off rack**. This will result in a inter-rack data transfer.



Replication factor

Why is the default replication factor 3?

- Suppose that a block is replicated on machine A and B.
  - The **simultaneous failure** of A and B results in the irreversible loss of the block.
  - If A fails, all the processing on the block must be redirected to B.

- A simultaneous failure/unavailability of three machines is **much less frequent**.

- Replicating blocks more than three times results in **high storage costs**.

Block size

Why is the block size set to 64MB or 128MB?

- The time T_{read} to **read a block** of data from a disk is given by:

$$T_{read} = T_{seek} + T_{transfer}$$

where:

  - T_{seek} is the time needed to **locate** the beginning of the block.
  - $T_{transfer}$ is the time needed to **transfer** the block.

- **Objective.** We want $0.01 T_{transfer} > T_{seek}$.
  - Assuming $T_{seek} = 10ms$ and a **transfer rate** of 100 MB/s, we have

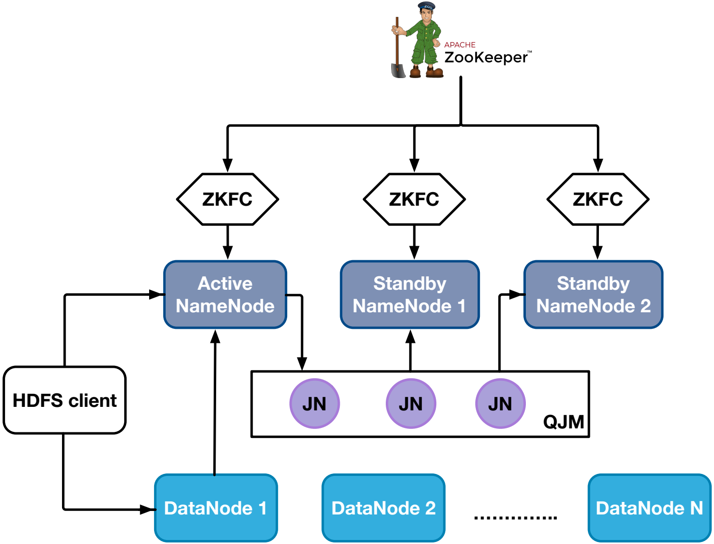
$$0.01 * B/100 > 10 * 10^{-3} \implies B > 100MB$$

Block size

Why is the block size set to 64MB or 128MB?

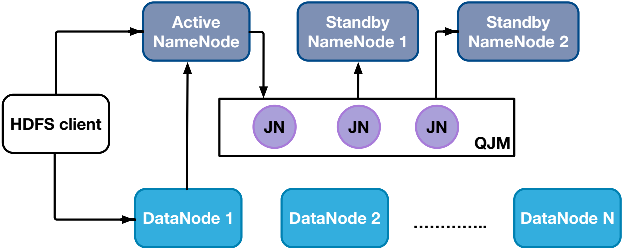
- In new generation SSD disks, T_{seek} is much lower (≈ 0.16ms).
- However:
  - Blocks **too small**: NameNodes and DataNodes have to manage more blocks.
    - More metadata to handle.
    - More **overhead** in data processing.
  - Blocks **too large**: We reduce the parallelism of data processing.
    - **Example.** Map tasks work on large chunks of data.

Back to the HDFS logical architecture



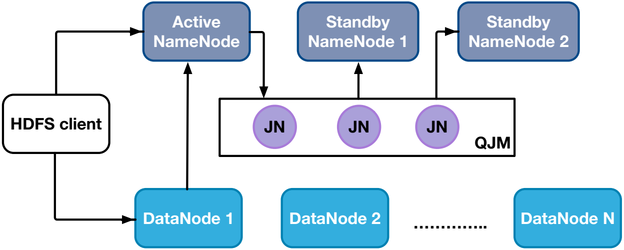
HDFS high availability

- In Hadoop version 1, the **NameNode** (NN) was a **single point of failure**.
- Hadoop version 3 provides an **active NameNode** and one to several **standby Namenodes**.
- The **active** NN serves the client requests.
- The **standby** NN takes over, should the active NN fail.



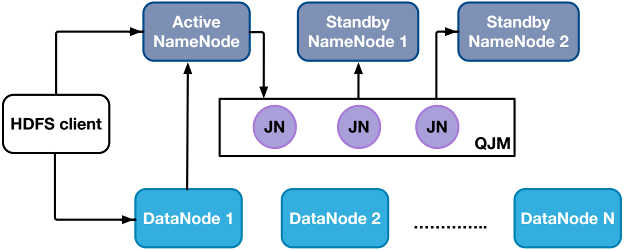
HDFS high availability

- **Standby** NNs keep their state **synchronized** with the **active** NN.
- **Standby** and **active** NNs communicate with a group of separate processes called **JournalNodes** (JN).
- All namespace modifications performed by the **active** NN, are logged to a majority of the JNs.
- The **standby** NNs read the edits from the JNs and apply them to their own **FsImage** file.



HDFS high availability

- **DataNodes** are configured to send block location information to all NNs.
- **Standby** NNs have up-to-date information regarding the location of blocks in the cluster.
- This ensures **fast failover** (when a **standby** NN becomes **active**).



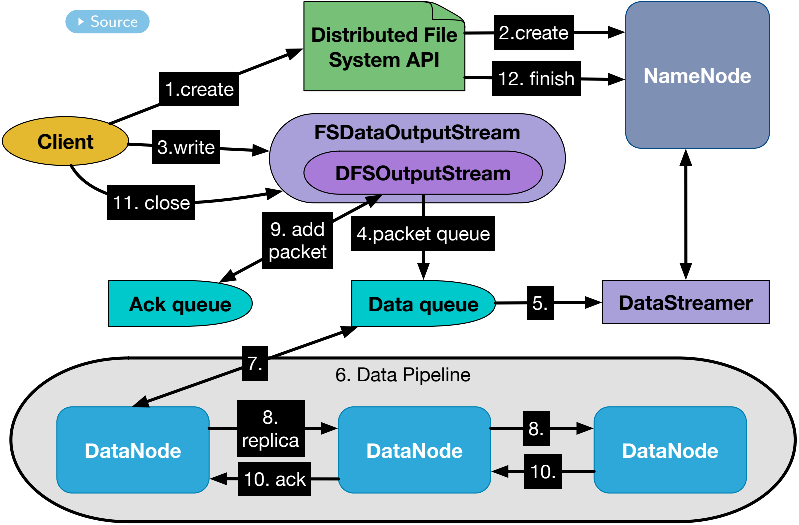
☞ There must be at most one active NameNode.

Zookeeper

- **Zookeeper** is the component that makes **automatic failover** possible.
- Zookeeper maintains an active persistent session with each NameNode.
    - Every **NameNode** communicates with **Zookeeper** through the **ZookeeperFailoverController (ZKFC)**.
- ZKFC checks the health of a NameNode with periodic **health check pings**.
    - If a NameNode doesn't respond in time, ZKFC informs Zookeeper.
- If the **active NameNode** is not healthy, Zookeeper informs the other NameNodes to start the failover procedure.
    - The first NameNode that takes a special exclusive lock in Zookeeper is elected.

Write workflow

Source

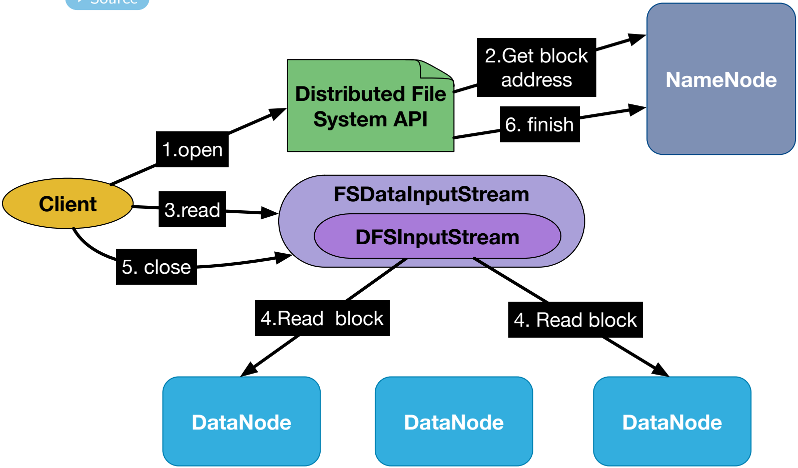


Write workflow

- The NameNode checks for the existence of the file before creation.
  - If the file exists, it **raises an exception**.
- The NameNode checks the user's permissions to create a file.
  - If the user hasn't the permissions, it **raises an exception**.
- The object *FSDataOutputStream* is used by the client to write the data.
- The object *DFSOutputStream* splits the data into blocks.
- The *DataStreamer* is a linked list of blocks.
  - For each block, the NameNode provides the DataNode where the data block is to be stored.
- The block is replicated across DataNodes.

Read workflow

Source



Read workflow

- The NameNode checks the permissions of the user to read the file.
  - If the user hasn't the permissions, it **raises an exception**.
- The NameNode returns the metadata on the **blocks** of the file.
  - For each block, its gives the list of the DataNodes containing the block.
  - DataNodes are sorted based on the **proximity** to the client.
- *DFSInputStream* is responsible for getting the data from the DataNodes.
  - If a DataNode fails to return a block, *DFSInputStream* tries the next DataNode in the list.

☞ In both read and write operations the client goes first through the NameNode.

References

- Singh, Chanchal, and Manish Kumar. *Mastering Hadoop 3: Big data processing at scale to unlock unique business insights*. Packt Publishing Ltd, 2019. Click here