Lecture-7
Big Data Systems (SEZG522/CCZG522)

BITS Pilani
Pilani Campus

Slides: Courtesy:.Prof. Anindya

innovate achieve lead

BITS Pilani
Pilani Campus

Second Semester

2024-25

Lecture-7 Contents

innovate achieve lead

- Hadoop architecture overview
  - ✓ Components
  - ✓ Hadoop 1 vs Hadoop 2
- HDFS
  - ✓ Architecture
  - ✓ Robustness
  - ✓ Blocks and replication strategy
  - ✓ Read and write operations
  - ✓ File formats
  - ✓ Commands

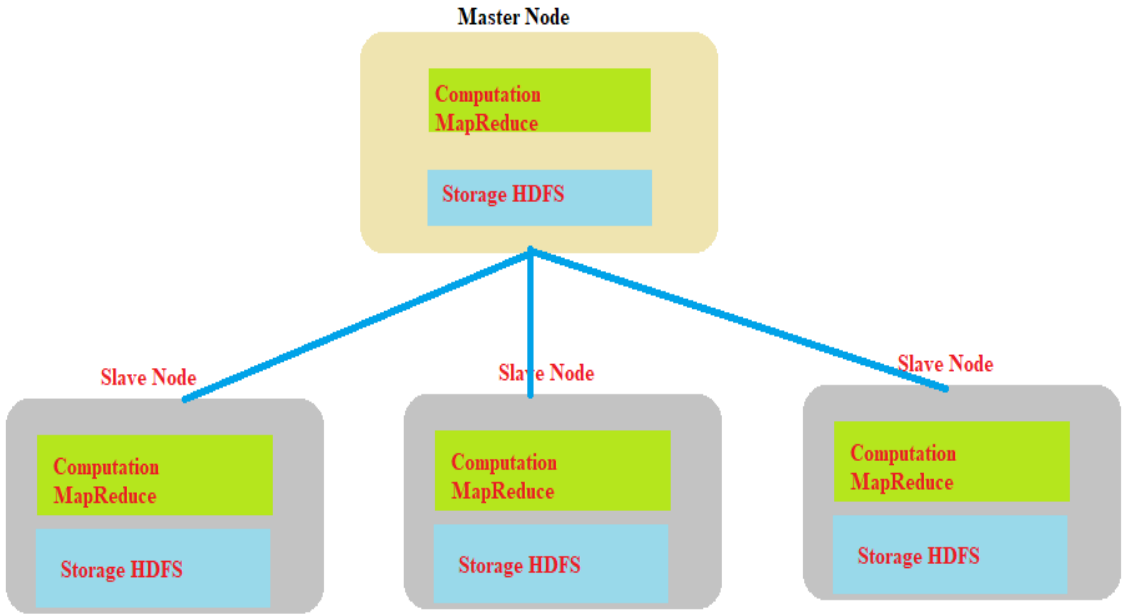
Hadoop - Data and Compute layers

A data storage layer

- A Distributed File System - HDFS

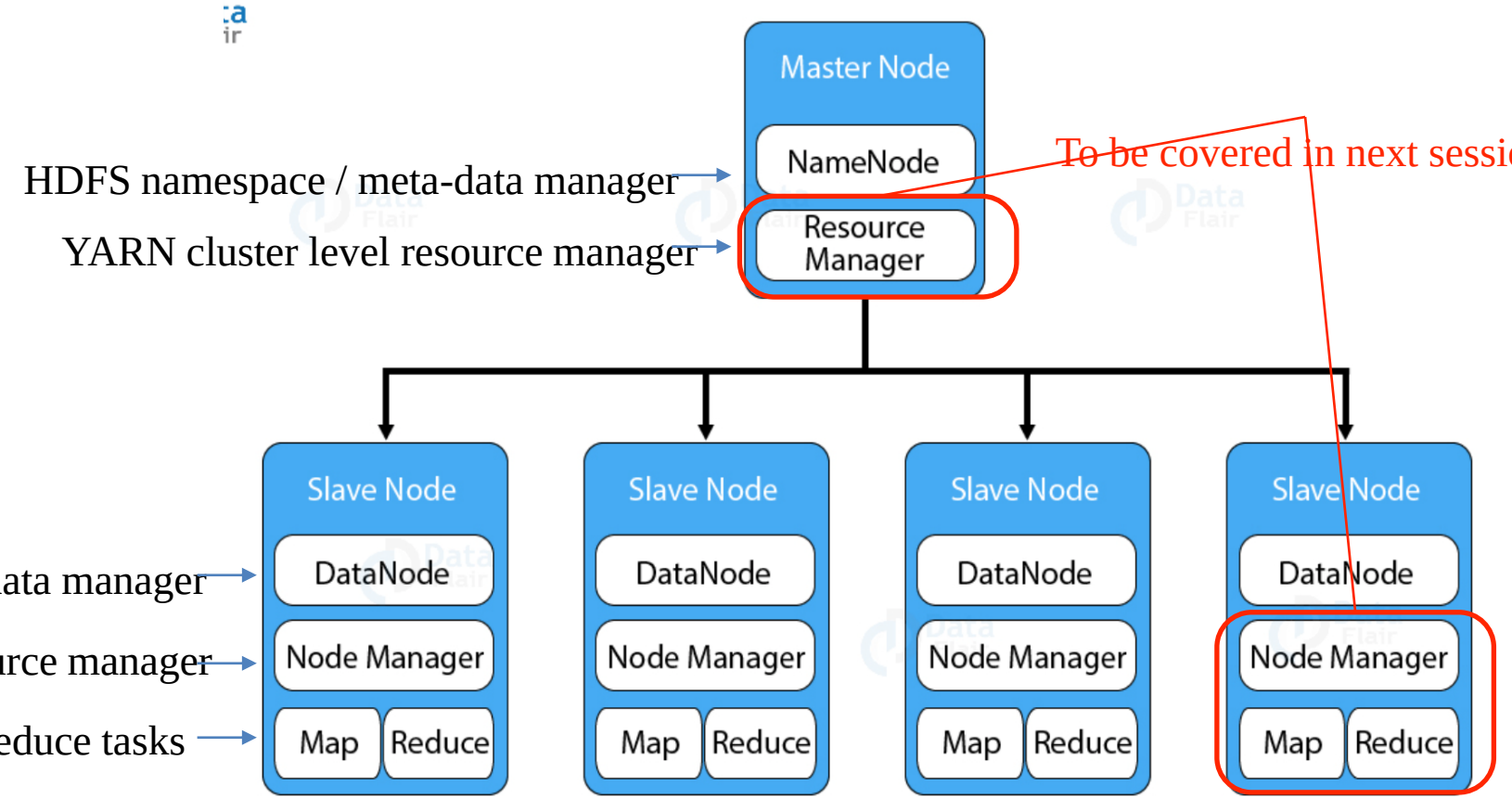
A data processing layer

- MapReduce programming



Hadoop 2 - Architecture

- Master-slave architecture for overall compute and data management
- Slaves implement peer-to-peer communication



***Note**: YARN Resource Manager also uses application level App Master processes on slave nodes for application specific resource management*

What changed from Hadoop 1 to Hadoop 2

Hadoop 1: MapReduce was coupled with resource management

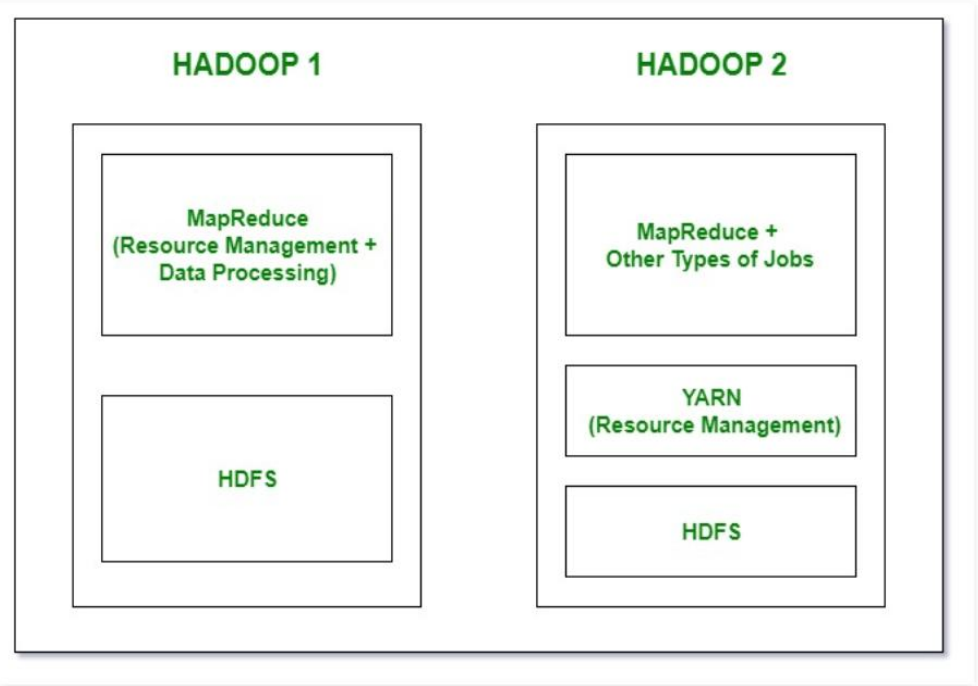
- Hadoop 2 brought in YARN as a resource management capability and MapReduce is only about data processing.

Hadoop 1: Single Master node with NameNode is a SPOF

- Hadoop 2 introduced active-passive and other HA configurations besides secondary NameNodes

Hadoop 1: Only MapReduce programs

- In Hadoop 2, non MR programs can be run by YARN on slave nodes (since decoupled from MapReduce) as well support for non-HDFS storage, e.g. Amazon S3 etc.



Hadoop Distributions

Open source Apache project

Core components :

- ✓ Hadoop Common
- ✓ Hadoop Distributed File System
- ✓ Hadoop YARN
- ✓ Hadoop MapReduce

HDFS Features

- A DFS stores data over multiple nodes in a cluster and allows multi-user access
  - Gives a feeling to the user that the data is on single machine
  - HDFS is a Java based DFS that sits on top of native FS
  - Enables storage of very large files across nodes of a Hadoop cluster
  - Data is split into large blocks : 128MB
- Scale through parallel data processing
  - 1 node with 1TB storage with IO bandwidth of 400MBps across 4 IO channels = 43 min
  - 10 nodes with partitioned 1 TB data can access in parallel that data in 4.3 min

Contd..

- **Fault tolerance through replication**
  - Default replication factor = 3 for every block (Hadoop 3 has some optimisations)
  - So 1 GB data can actually take 3GB storage
- **Consistency**
  - Write once and read many time workload
  - Files can be appended, truncated but not updated at any arbitrary point

Contd..

- **Cost:** Typically deployed using commodity hardware for low TCO - so adding more nodes is cost-effective

- **Variety and Volume of Data**: Huge data i.e. Terabytes & petabytes of data and different kinds of data - structured, unstructured or semi structured.

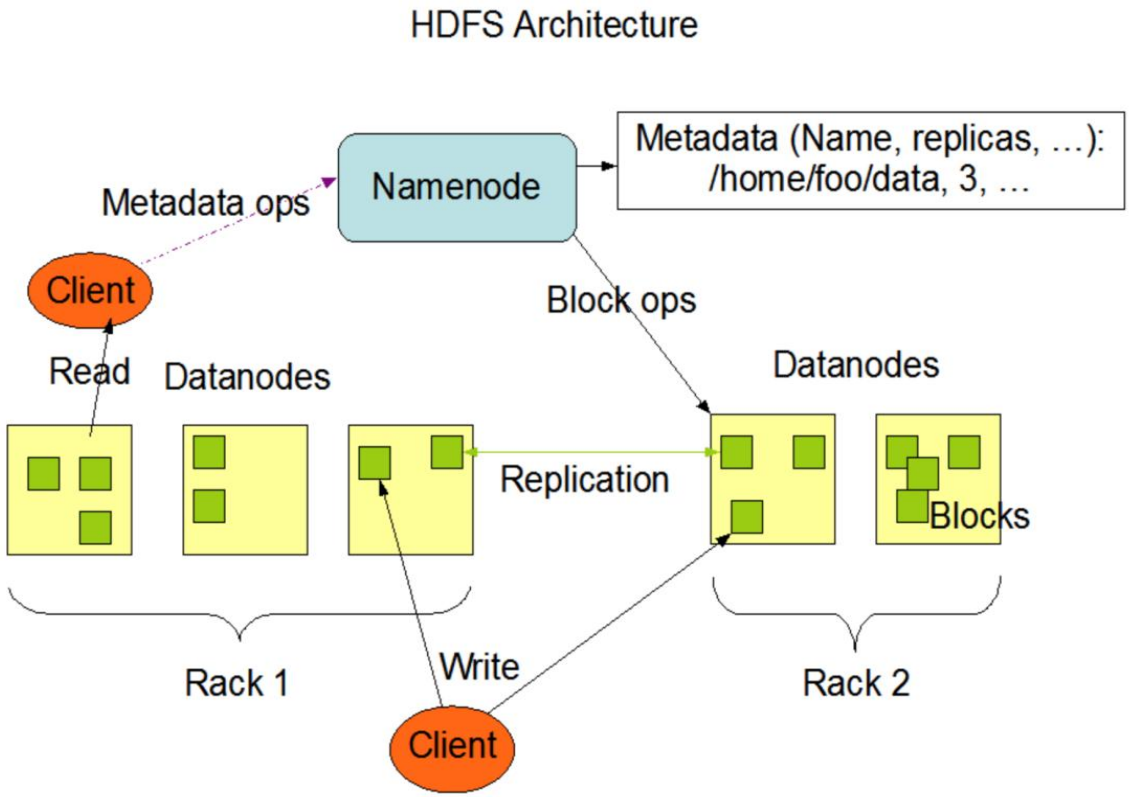
Contd..

- **Data Integrity:** HDFS nodes constantly verify checksums to preserve data integrity. On error, new copies are created and old copies are deleted.

- **Data Locality:** Data locality talks about moving processing unit to data rather than the data to processing unit. Bring the computation part to the data nodes where the data is residing. Hence, you are not moving the data, you are bringing the program or processing part to the data.

HDFS Architecture - Master node

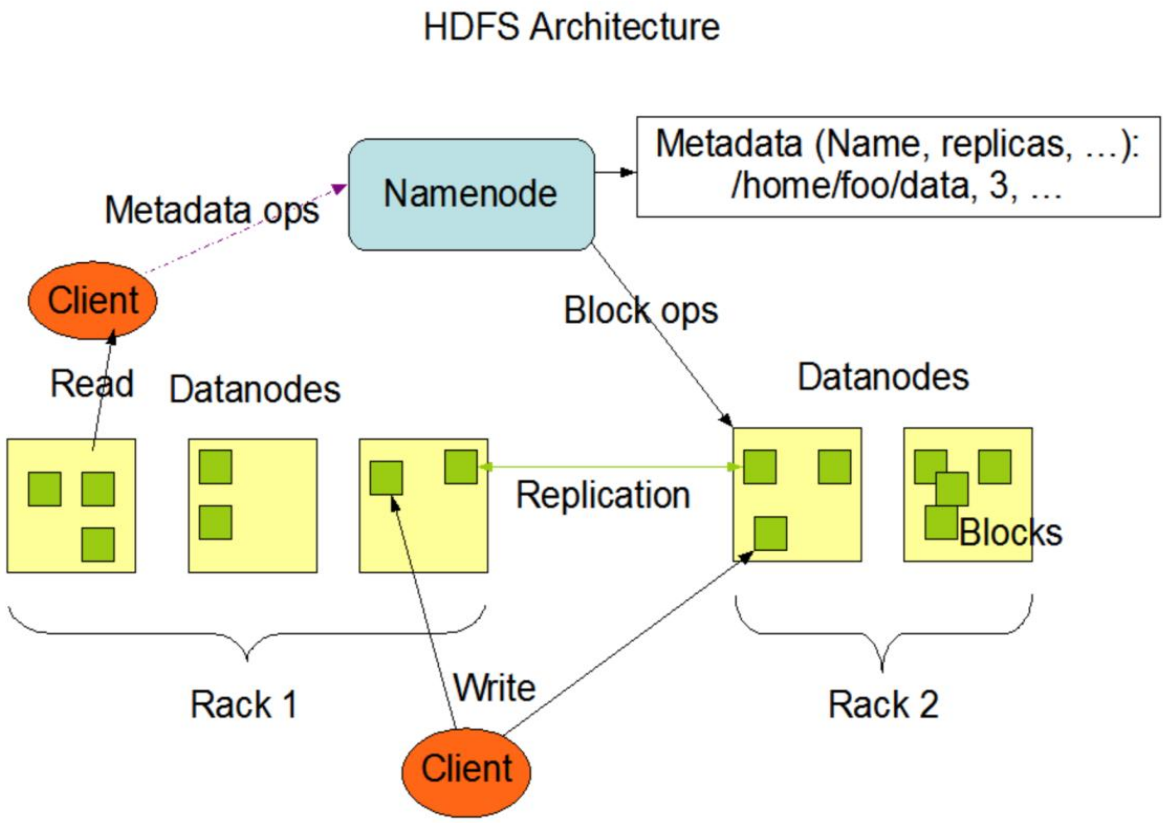
- Master slave architecture within a HDFS cluster
- One master node with NameNode
  - Maintains namespace - Filename to blocks and their replica mappings
  - Serves as arbitrator and doesn't handle actual data flow
  - HDFS client app interacts with NameNode for metadata



HDFS Architecture - Slave node

Multiple slave nodes with one DataNode per slave

- Serves block R/W from Clients
- Serves Create/Delete/Replicate requests from NameNode
- DataNodes interact with each other for pipeline reads and writes.



Functions of a NameNode

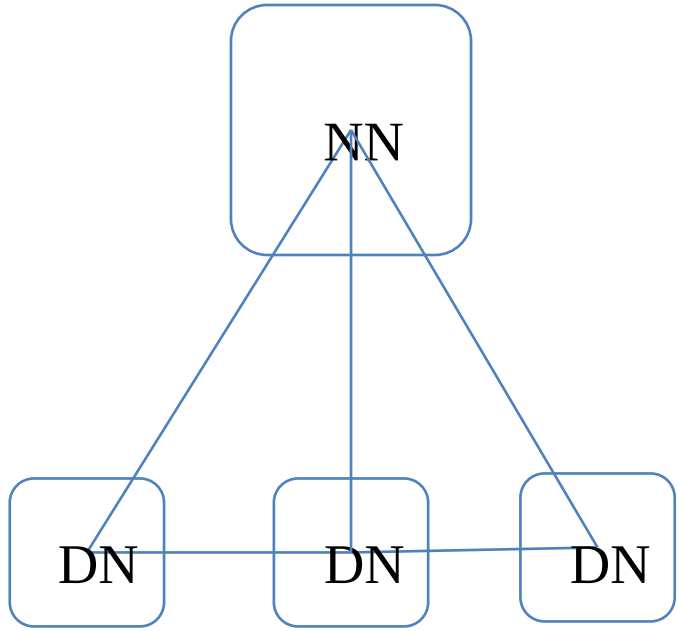
- Maintains namespace in HDFS with 2 files
  - FsImage: Contains mapping of blocks to file, hierarchy, file properties / permissions
  - EditLog: Transaction log of changes to metadata in FsImage
- Does not store any data - only meta-data about files
- Runs on Master node while DataNodes run on Slave nodes
- HA can be configured
- Records each change that takes place to the meta-data. e.g. if a file is deleted in HDFS, the NameNode will immediately record this in the EditLog.
- Receives periodic Heartbeat and a block report from all the DataNodes in the cluster to ensure that the DataNodes are live.
- Ensure replication factor is maintained across DataNode failures
  - In case of the DataNode failure, the NameNode chooses new DataNodes for new replicas, balance disk usage and manages the communication traffic to the DataNodes

Namenode - What happens on start-up

1. Enters into safe mode
   - ✓ Check for status of Data nodes on slaves
     - Does not allow any Datanode replications in this mode
     - Gets heartbeat and block report from Datanodes
     - Checks for minimum Replication Factor needed
   - ✓ Updates meta-data (this is also done at checkpoint time)
     - Reads FsImage and EditLog from disk into memory
     - Applies all transactions from the EditLog to the in-memory version of FsImage
     - Flushes out new version of FsImage on disk
     - Keeps latest FsImage in memory for client requests
     - Truncates the old EditLog as its changes are applied on the new FsImage
2. Exits safe mode
3. Continues with further replications needed and client requests

Functions of a DataNode

- Each slave in cluster runs a DataNode
- Nodes store actual data blocks and R/W data for the HDFS clients as regular files on the native file system, e.g. ext2 or ext3
- During pipeline read and write, DataNodes communicate with each other
  - We will discuss what's a pipeline
- No additional HA because blocks are anyway replicated



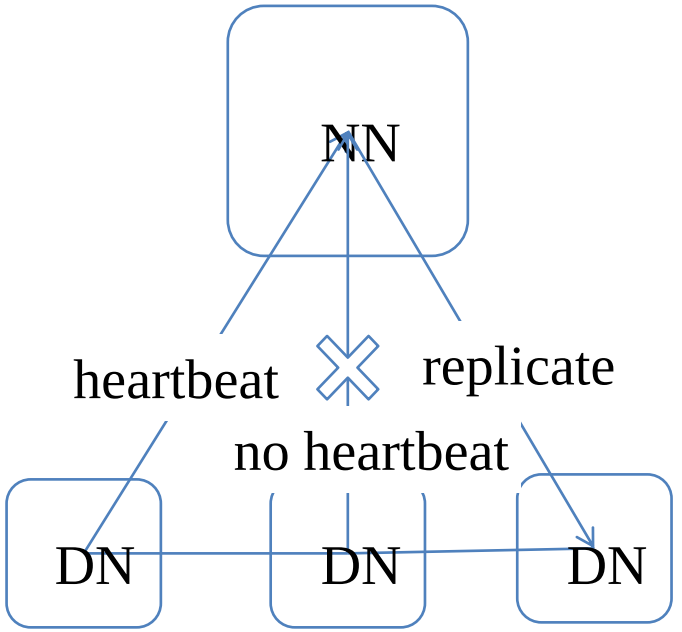
DN-2-DN pipeline for data transfer

Contd..

- DataNode continuously sends heartbeat to NameNode (default 3 sec)

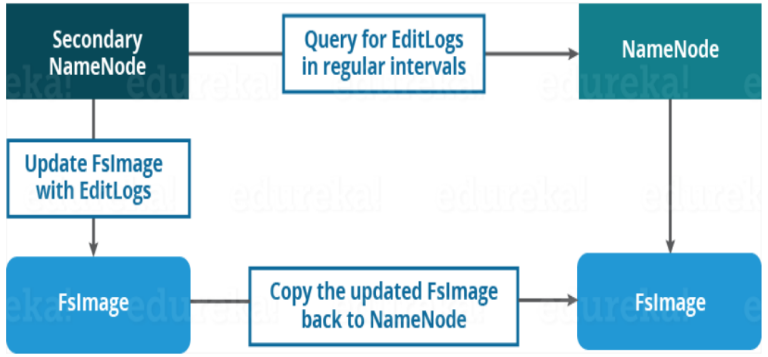
  To ensure the connectivity with NameNode

- If no heartbeat message from DataNode, NameNode replicates that DataNode within the cluster and removes the DN from the meta-data records
- DataNodes also send a BlockReport on start-up / periodically containing file list



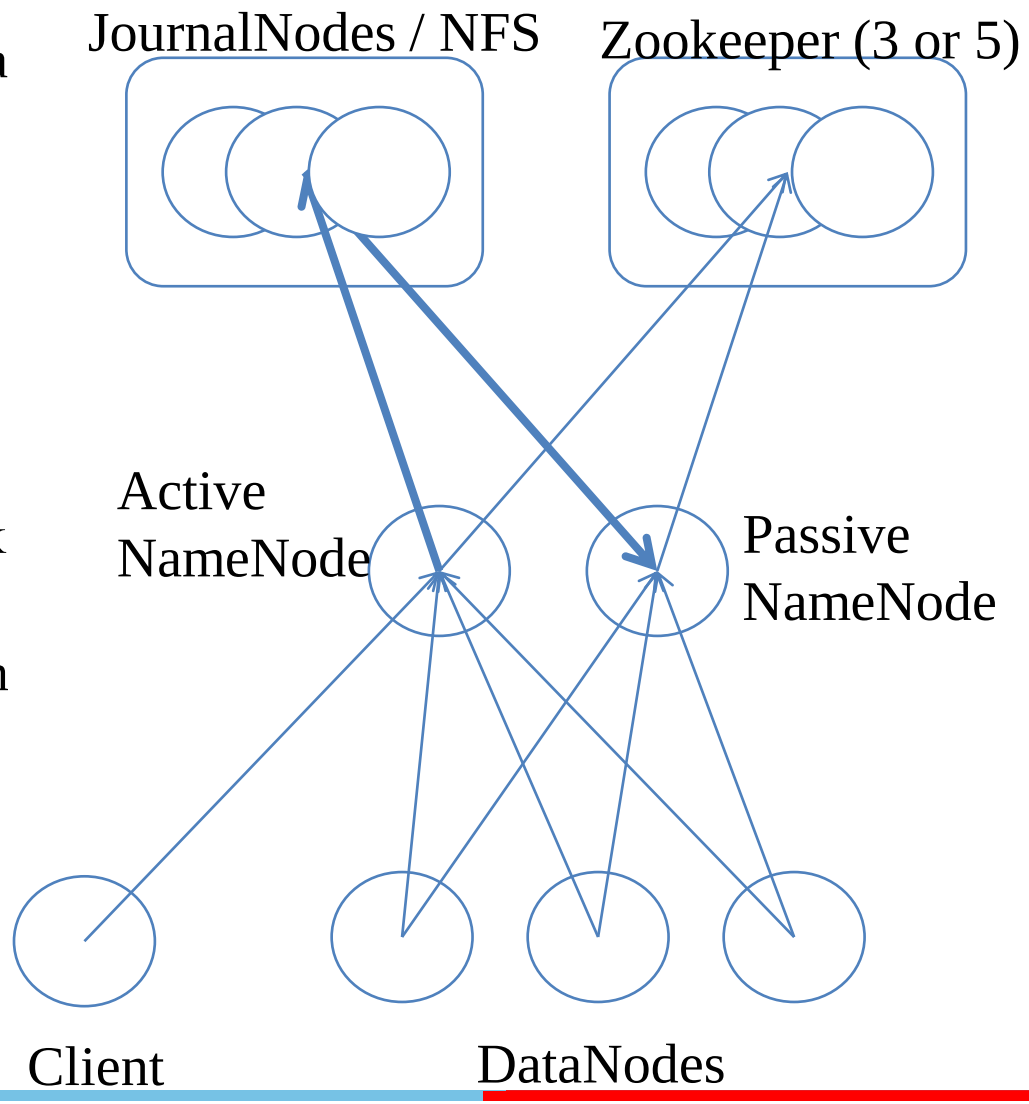
Hadoop 2: Introduction of Secondary NameNode

- In the case of failure of NameNode,
  - ✓ The secondary NameNode can be configured manually to bring up the cluster
- The Secondary NameNode constantly reads all the file systems and metadata from the RAM of the NameNode (snapshot) and writes to its local file system.
- It is responsible for combining the EditLogs with FsImage from the NameNode.
- It downloads the EditLogs from the NameNode at regular intervals and applies to FsImage.
- Hence, Secondary NameNode performs regular checkpoints in HDFS. Therefore, it is also called CheckpointNode.
- After recover from a failure, the new FsImage is copied back to the NameNode, which is used whenever the NameNode is started the next time.



HA configuration of NameNode

- Active-Passive configuration can also be setup with a standby NameNode
- Can use a Quorum Journal Manager (QJM) or NFS to maintain shared state
- DataNodes send heartbeats and updates to both NameNodes.
- Writes to JournalNodes only happens via Active NameNode - avoids “split brain” scenario of network partitions
- Standby reads from JournalNodes to keep updated on state as well as latest updates from DataNodes
- Zookeeper session may be used for failure detection and election of new Active



Other robustness mechanisms

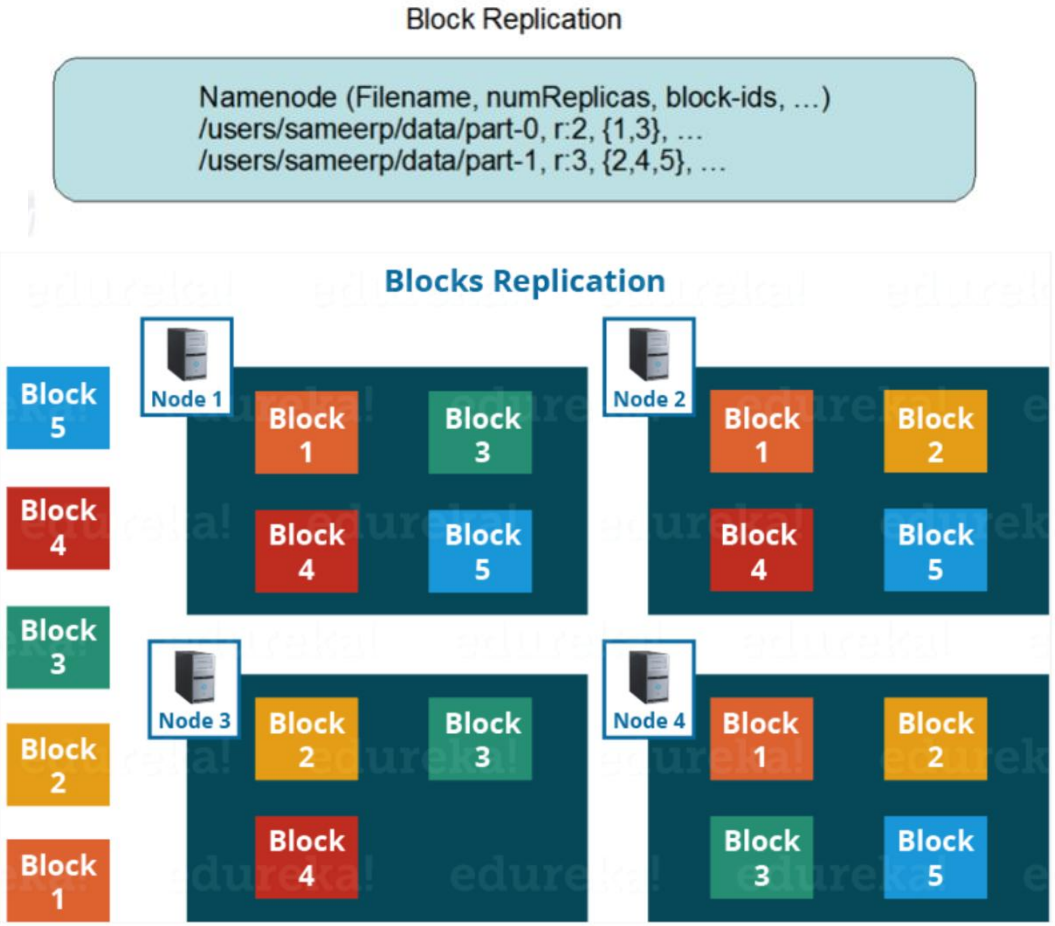
- Types of failures - DataNode, NameNode failures and network partitions
- Heartbeat from DataNode to NameNode for handling DN failures
  - When data node heartbeat times out (10min) NameNode updates state and starts pointing clients to other replicas.
  - Timeout (10min) is high to avoid replication storms but can be set lower especially if clients want to read recent data and avoid stale replicas.
- Cluster rebalancing by keeping track of RF per block and node usage
- Checksums stored in NameNode for blocks written to DataNodes to check data integrity on corruption on node / link and software bugs

Block in HDFS

- HDFS stores each file as blocks which are scattered throughout the Apache Hadoop cluster.
- The default size of each block is 128 MB in Apache Hadoop 2.x (64 MB in Apache Hadoop 1.x) which you can configure as per your requirement.
- It is not necessary that in HDFS, each file is stored in exact multiple of the configured block size (128 MB, 256 MB etc.).
  - A file of size 514 MB can have 4 x 128MB and 1 x 2MB blocks
- Why large block of 128MB ?
  - HDFS is used for TB/PB size files and small block size will create too much meta-data

Replica Placement Strategy - with Rack awareness

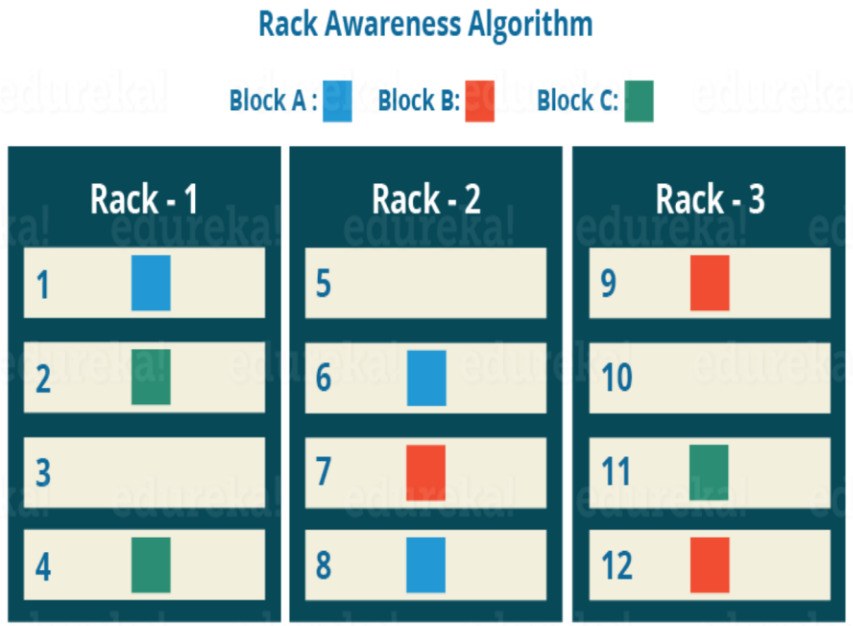
- First replica is placed on the same node as the client
- Second replica is placed on a node that is present on different rack
- Third replica is placed on same rack as second but on a different node
- Putting each replica on a different rack is expensive write operation
- For replicas > 3, nodes are randomly picked for 4th replica without violating upper limit per rack as (replicas-1) / racks + 2.
- Total replicas <= #DataNodes with no 2 replicas on same DN
- Once the replica locations are set, pipeline is built
- Shows good reliability
- NameNode collects block report from DataNodes to balance the blocks across nodes and control over/under replication of blocks



Why rack awareness ?

To improve the network performance: The communication between nodes residing on different racks is directed via switch. In general, you will find *greater network bandwidth* between machines in the same rack than the machines residing in different rack. So, the **Rack Awareness helps you to have reduce write traffic in between different racks** and thus providing a better write performance. Also, you will be gaining increased read performance because you are using the bandwidth of multiple racks.

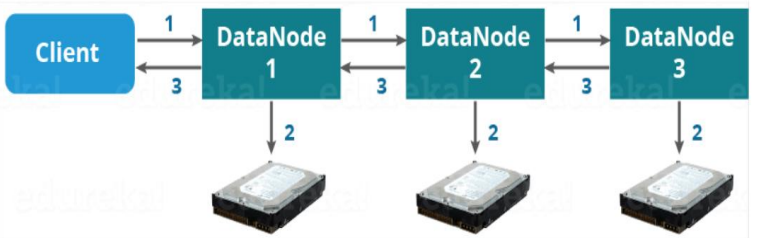
To prevent loss of data: We **don't have to worry about the data even if an entire rack fails** because of the switch failure or power failure. And if you think about it, it will make sense, as it is said that *never put all your eggs in the same basket.*



HDFS data writes

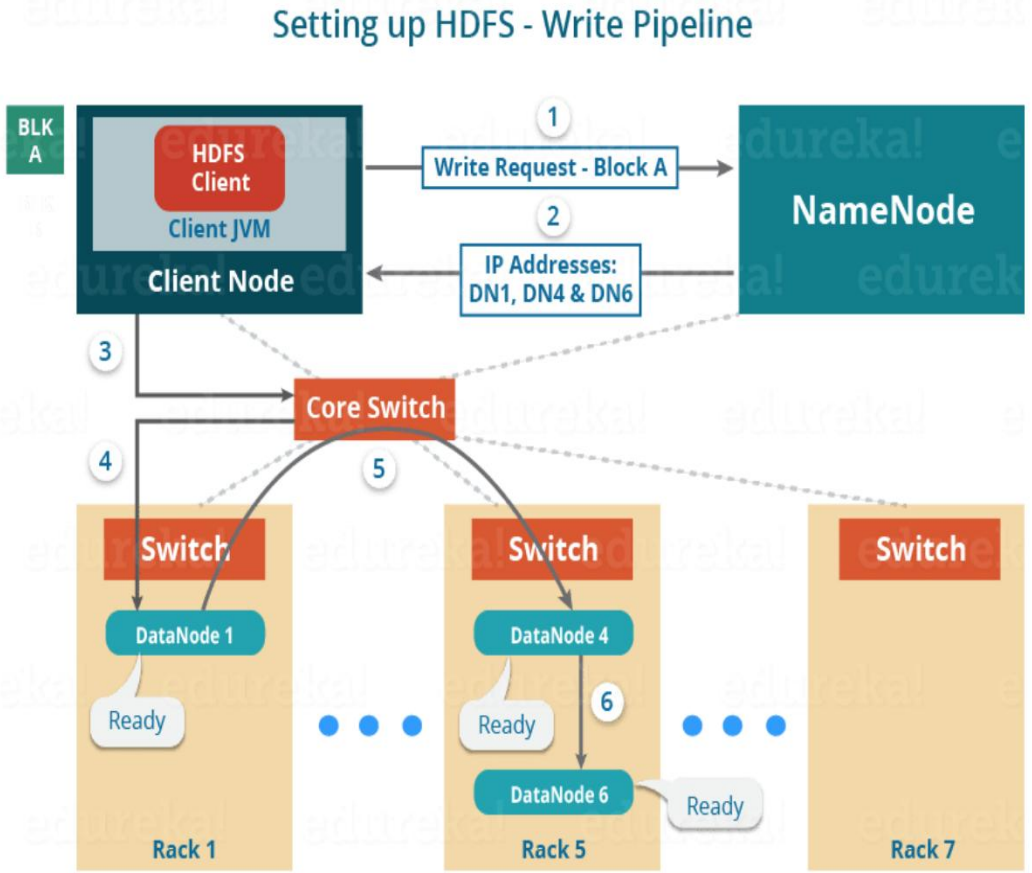
•Now, the following protocol will be followed whenever the data is written into HDFS:

- HDFS client contacts NameNode for Write Request against the two blocks, say, Block A & Block B.
- NameNode grants permission to client with IP addresses of the DataNodes to copy blocks
- Selection of DataNodes is randomized but factoring in availability, RF, and rack awareness
- For 3 copies, 3 unique DNs needed, if possible, for each block.
  - For Block A, list A = {DN1, DN4, DN6}
  - For Block B, set B = {DN3, DN7, DN9}
- Each block will be copied in three different DataNodes to maintain the replication factor consistent throughout the cluster.
- Now the whole data copy process will happen in three stages:
  - Set up of Pipeline
  - Data streaming and replication
  - Shutdown of Pipeline (Acknowledgement stage)



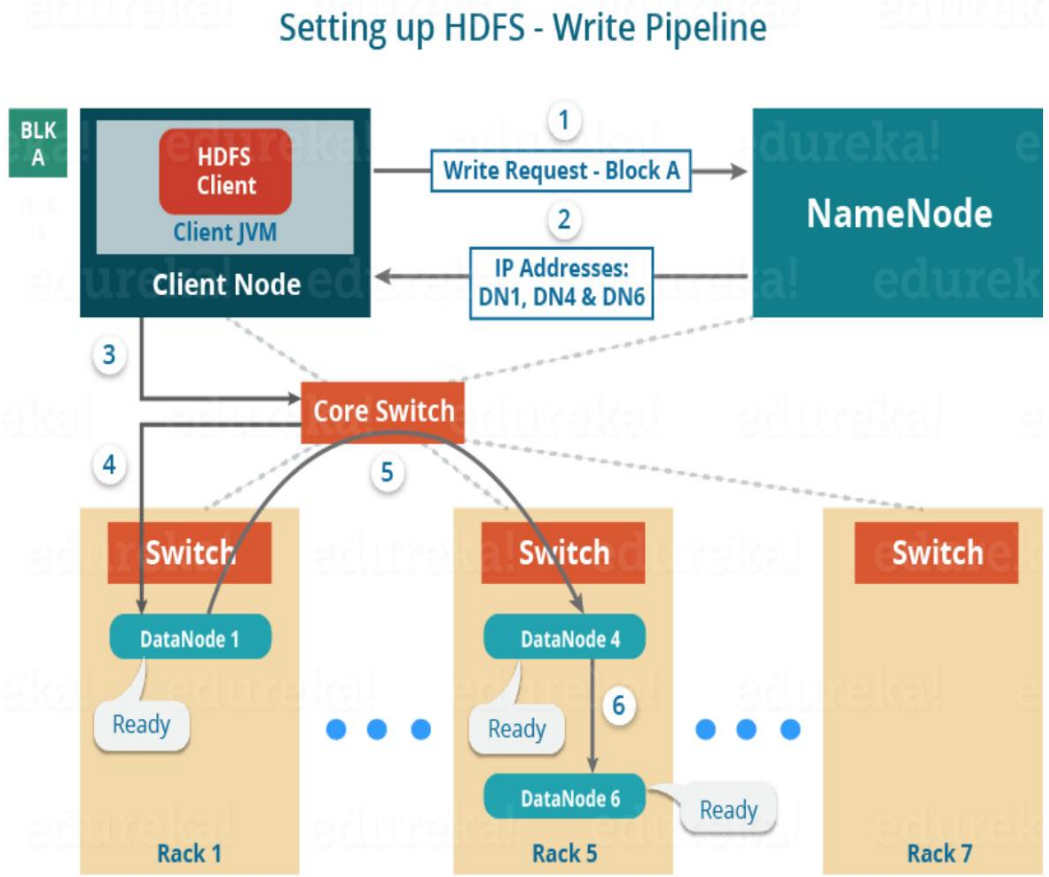
HDFS Write: Step 1. Setup pipeline

Client creates a pipeline for each of the blocks by connecting the individual DataNodes in the respective list for that block. Let us consider Block A. The list of DataNodes provided by the NameNode is DN1, DN4, DN6



HDFS Write: Step 1. Setup pipeline for a block

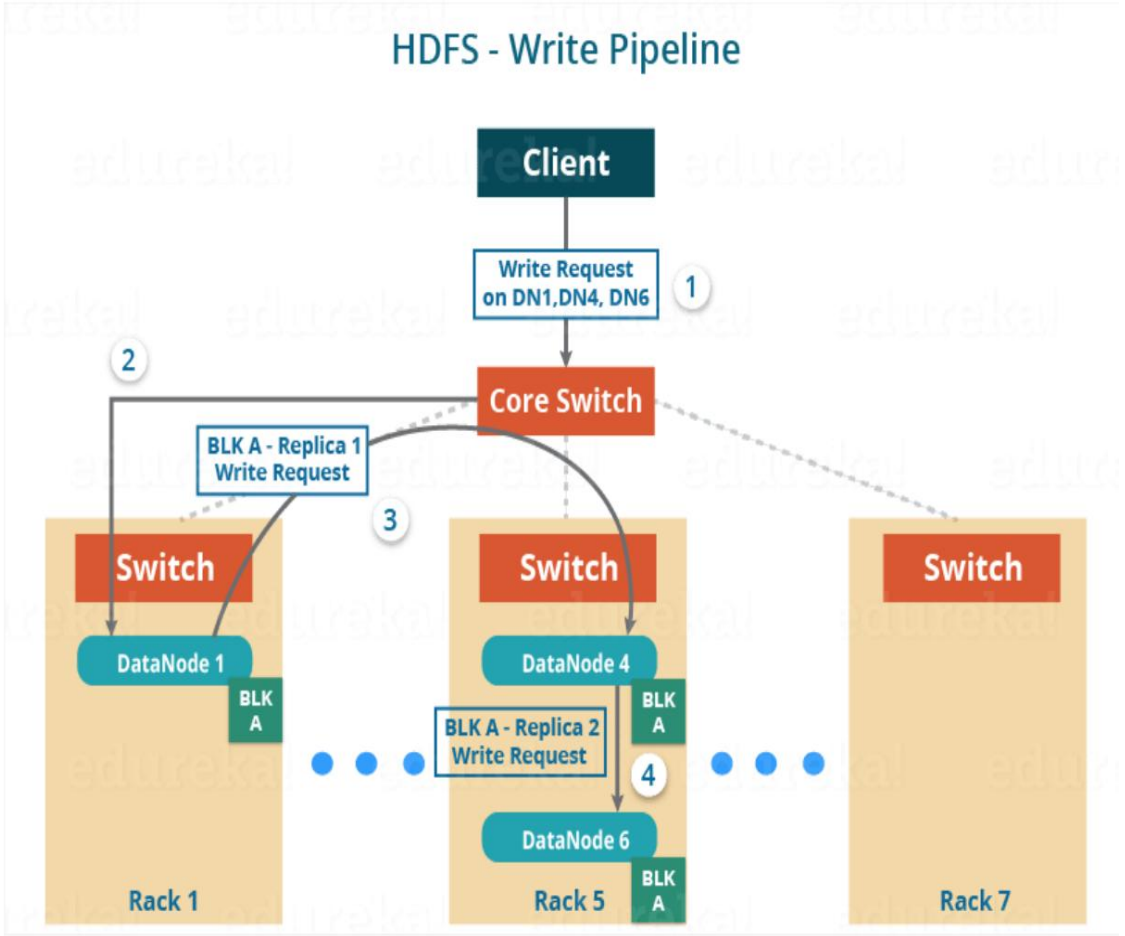
1. Client chooses the first DataNode (DN1) and will establish a TCP/IP connection.
2. Client informs DN1 to be ready to receive the block.
3. Provides IPs of next two DNs (4, 6) to DN1 for replication.
4. The DN1 connects to DN4 and informs it to be ready and gives IP of DN6. DN4 asks DN6 to be ready for data.
5. Ack of readiness follows the reverse sequence, i.e. from the DN6 to DN4 to DN1.
6. At last DN1 will inform the client that all the DNs are ready and a pipeline will be formed between the client, DataNode 1, 4 and 6.
7. Now pipeline set up is complete and the client will finally begin the data copy or streaming process.



HDFS Write: Step 2. Data streaming

Client pushes the data into the pipeline.

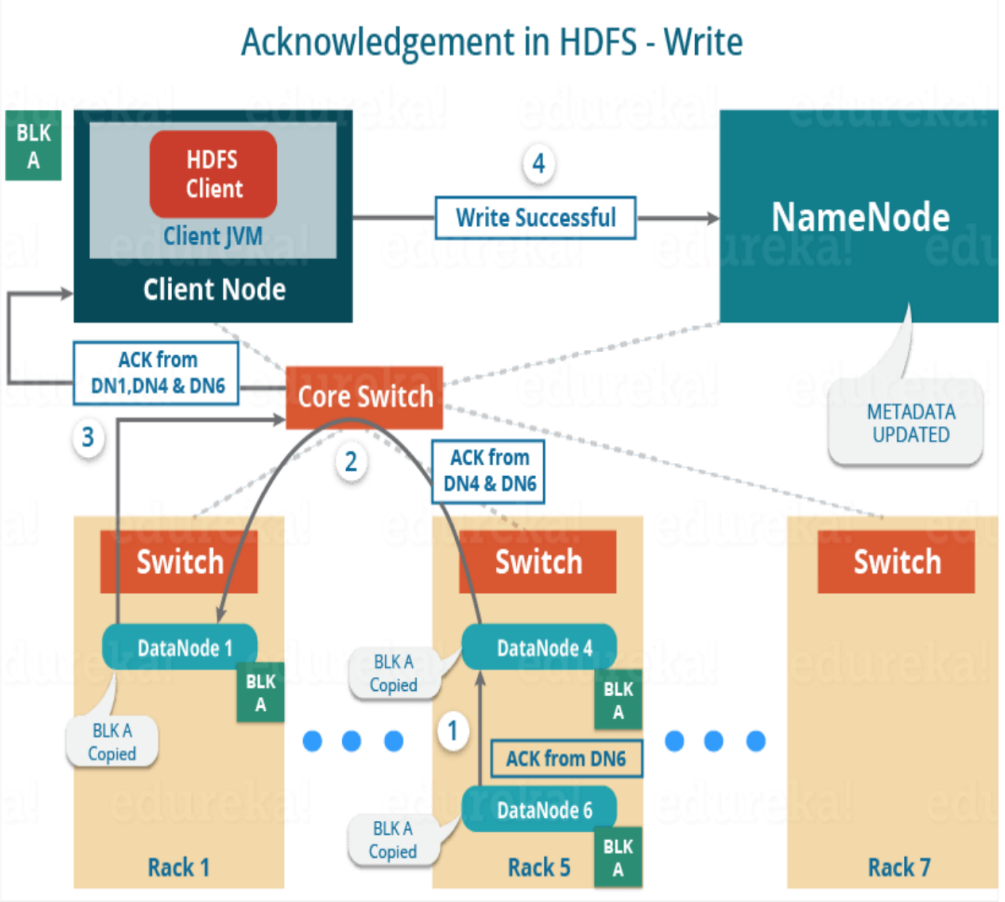
1. Once the block has been written to DataNode 1 by the client, DataNode 1 will connect to DataNode 4.
2. Then, DataNode 1 will push the block in the pipeline and data will be copied to DataNode 4.
3. Again, DataNode 4 will connect to DataNode 6 and will copy the last replica of the block.



HDFS Write: Step 3. Shutdown pipeline / ack

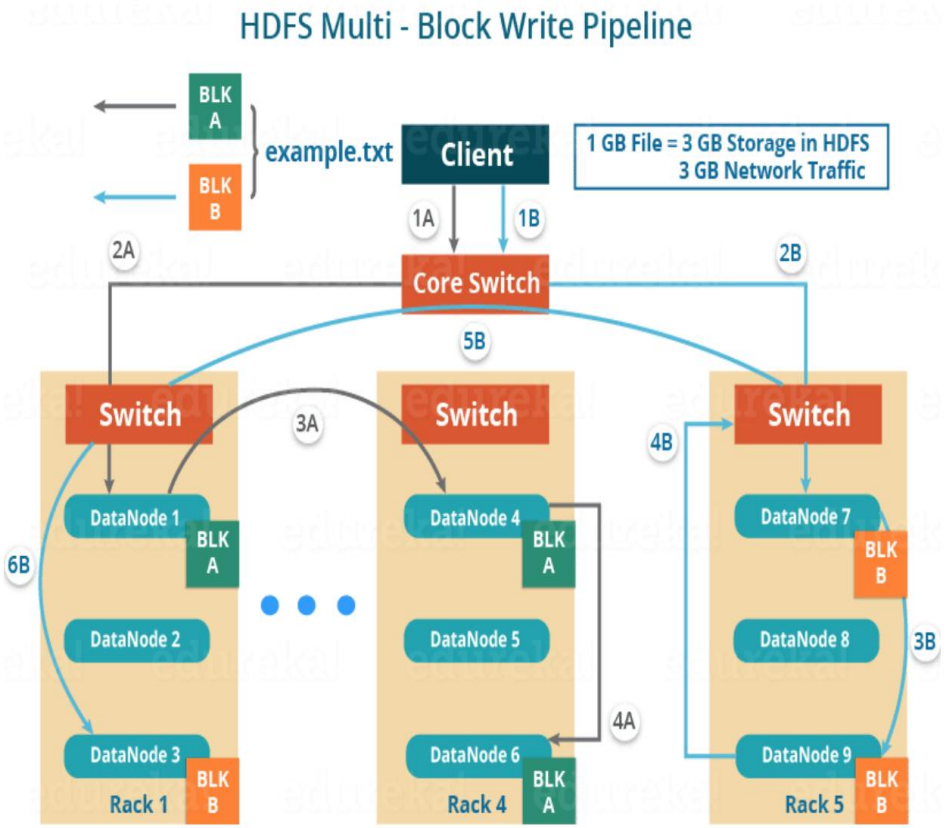
Block is now copied to all DNs. Client and NameNode need to be updated. Client needs to close pipeline and end TCP session.

1. Acknowledgement happens in the reverse sequence i.e. from DN 6 to 4 and then to 1.
2. DN1 pushes three acknowledgements (including its own) into pipeline and client.
3. Client informs NameNode that data has been written successfully.
4. NameNode updates metadata.
5. Client shuts down the pipeline.



Multi-block writes

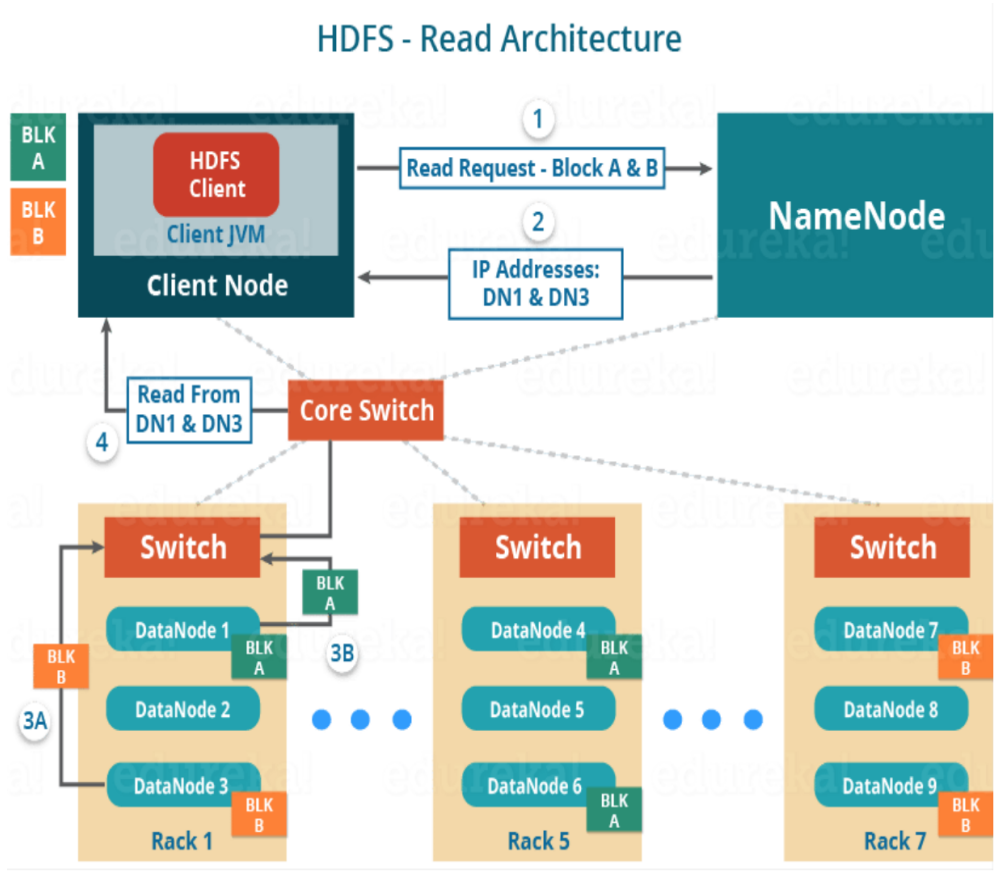
- The client will copy Block A and Block B to the first DataNode simultaneously.
- Parallel pipelines for each block
- Pipeline process for a block is same as discussed.
- E.g. 1A, 2A, 3A, … and 1B, 2B, 3B, … work in parallel



HDFS Read

1. Client contacts NameNode asking for the block metadata for a file
2. NameNode returns list of DNs where each block is stored
3. Client connects to the DNs where blocks are stored
4. The client starts reading data parallel from the DNs (e.g. Block A from DN1, Block B from DN3)
5. Once the client gets all the required file blocks, it will combine these blocks to form a file.

How are blocks chosen by NameNode ?
While serving read request of the client, HDFS selects the replica which is closest to the client. This reduces the read latency and the bandwidth consumption. Therefore, that replica is selected which resides on the same rack as the reader node, if possible.



Basic HDFS command reference

list files in the path of the file system

- **hadoop fs -ls <path>**

alters the permissions of a file where <arg> is the binary argument e.g. 777

- **hadoop fs -chmod <arg> <file-or-dir>**

change the owner of a file

- **hadoop fs -chown <owner>:<group> <file-or-dir>**

make a directory on the file system

- **hadoop fs -mkdir <path>**

copy a file from the local storage onto file system

- **hadoop fs -put <local-origin> <destination>**

Contd..

copy a file to the local storage from the file system

- **hadoop fs -get <origin> <local-destination>**

similar to the put command but the source is restricted to a local file reference

- **hadoop fs -copyFromLocal <local-origin> <destination>**

similar to the get command but the destination is restricted to a local file reference

- **hadoop fs -copyToLocal <origin> <local-destination>**

create an empty file on the file system

- **hadoop fs -touchz**

copy files to stdout

- **hadoop fs -cat <file>**