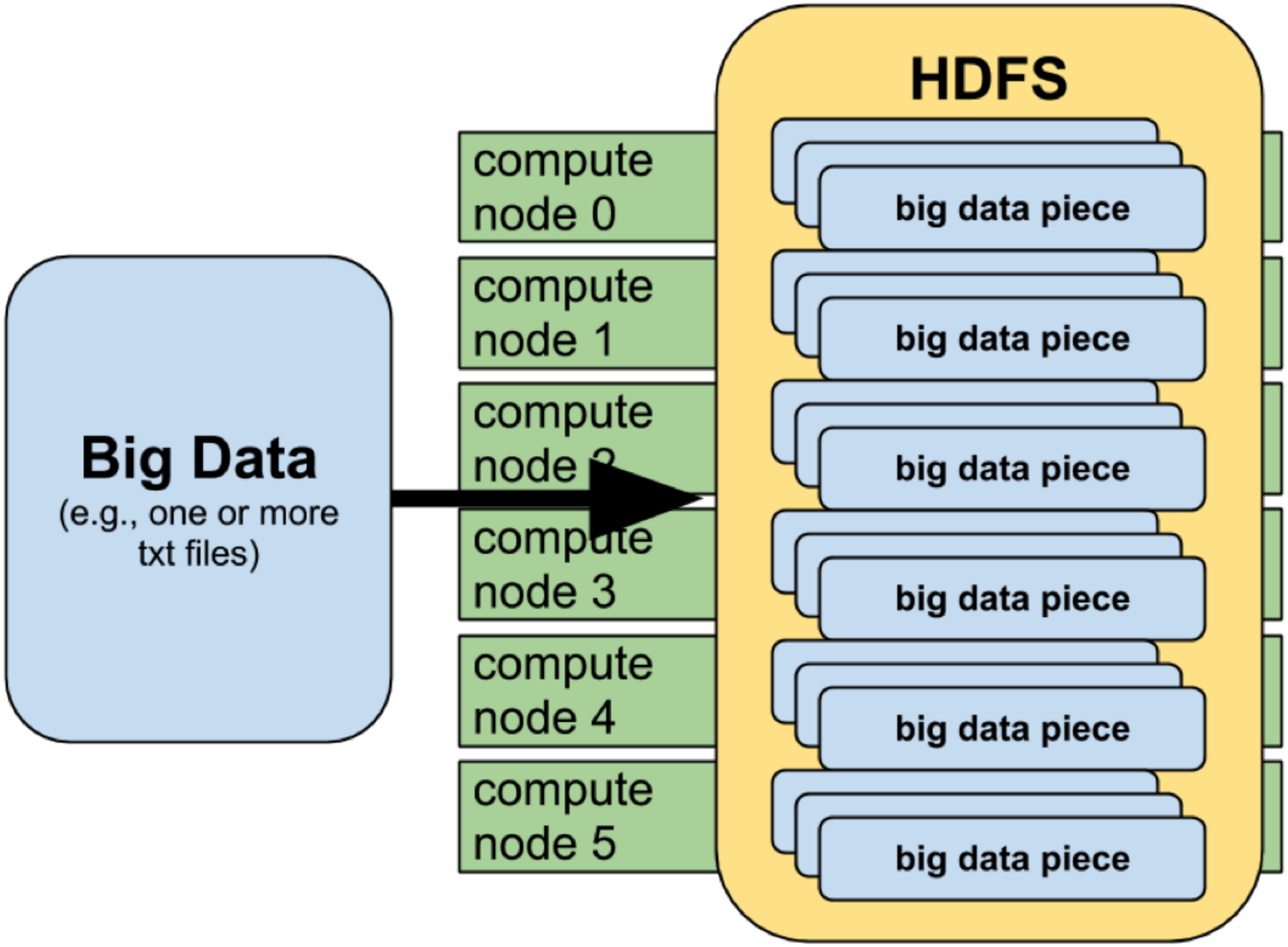
hadoop

HDFS

AGENDA

- CONCEPT: HDFS
- NameNode
- Data Node
- **Hadoop HDFS Daemons**
- **What is Secondary NameNode?**
- **What is Backup Node?**
- Data storage in HDFS
- **Blocks**
- **What is Replication Management?**
- HDFS Architecture
- **Read Operation** In HDFS
- Write Operation In HDFS
- Data Replication


Big Data
(e.g., one or more txt files)
HDFS
compute node 0
compute node 1
compute node 2
compute node 3
compute node 4
compute node 5
big data piece
big data piece
big data piece
big data piece
big data piece
big data piece

CONCEPT: HDFS

- **HDFS** is the world’s most reliable storage system. HDFS is a Filesystem of Hadoop designed for storing very large files running on a cluster of commodity hardware. It is designed on principle of storage of less number of large files rather than the huge number of small files. Hadoop HDFS also provides fault tolerant storage layer for Hadoop and its other components. HDFS Replication of data helps us to attain this feature. It stores data reliably even in the case of hardware failure. It provides high throughput access to application data by providing the data access in parallel. Let us move ahed in this HDFS tutorial with major areas of Hadoop Distributed File System.

CONCEPT: HDFS

- HDFS is a Java-based file system that provides scalable and reliable data storage, and it was designed to span large clusters of commodity servers.
- HDFS holds a very large amount of data and provides easier access.
- To store such huge data, the files are stored across multiple machines. These files are stored in a redundant fashion to rescue the system from possible data losses in case of failure.
- HDFS also makes applications available to parallel processing. HDFS is built to support applications with large data sets, including individual files that reach into the terabytes.
- It uses a master/slave architecture, with each cluster consisting of a single Name Node that manages file system operations and supporting Data Nodes that manage data storage on individual compute nodes.
- When HDFS takes in data, it breaks the information down into separate pieces and distributes them to different nodes in a cluster, allowing for parallel processing.

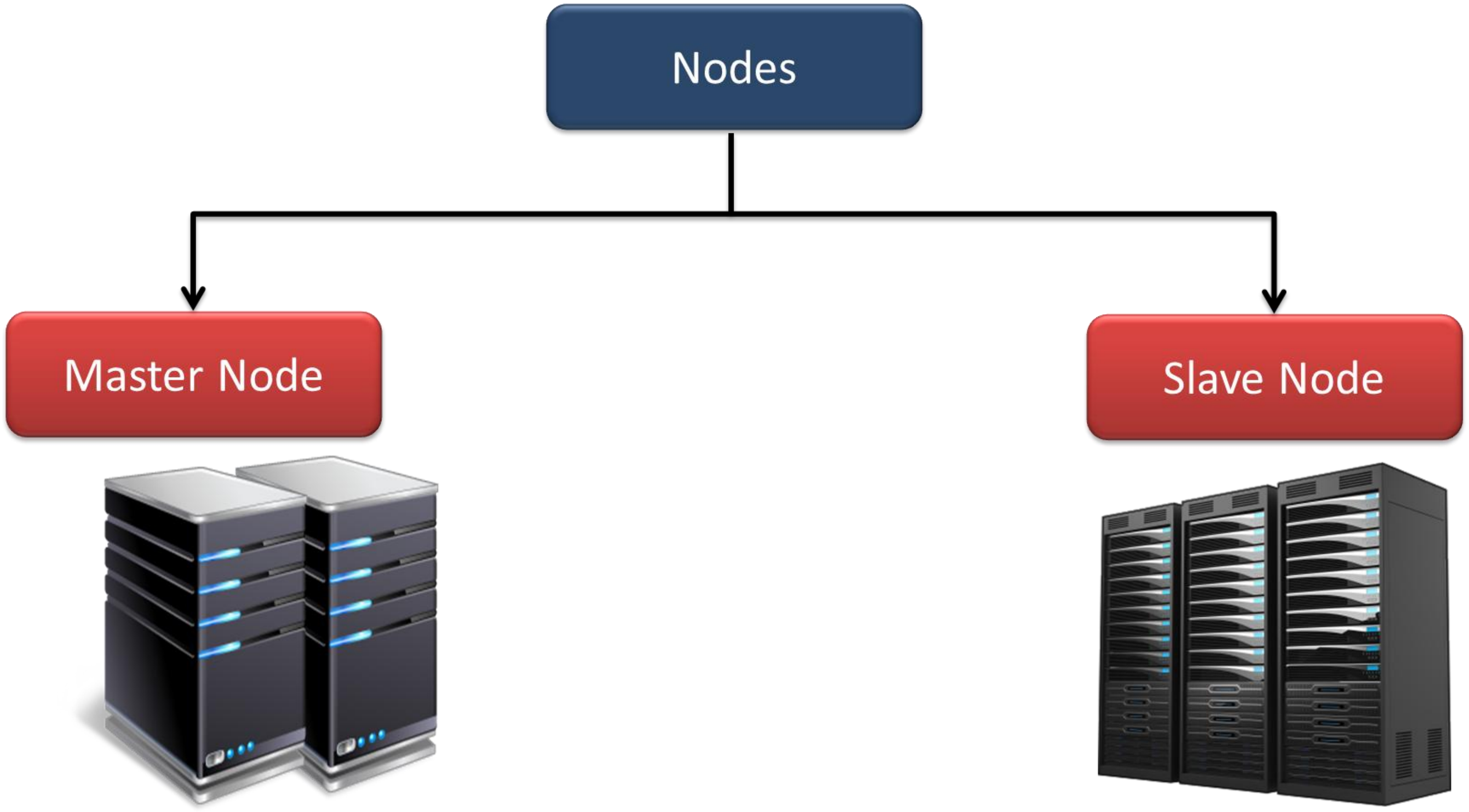
CONCEPT: HDFS

- A single physical machine gets saturated with its storage capacity as data grows. With this growth comes the impending need to partition your data across separate machines.
- This type of File system that manages storage of data across a network of machines is called a Distributed File System.
- HDFS is a core component of Apache Hadoop and is designed to store large files with streaming data access patterns, running on clusters of commodity hardware. With Hortonworks Data Platform (HDP), HDFS is now expanded to support heterogeneous storage media within the HDFS cluster.

CONCEPT: HDFS

- The file system also copies each piece of data multiple times and distributes the copies to individual nodes, placing at least one copy on a different server rack
- HDFS and YARN from the data management layer of Apache Hadoop.
- HDFS has demonstrated production scalability of up to 200 PB of storage and a single cluster of 4500 servers, supporting close to a billion files and blocks.
- HDFS is a scalable, fault-tolerant, distributed storage system that works closely with a wide variety of concurrent data access applications, coordinated by YARN. HDFS will “just work” under a variety of physical and systemic circumstances. By distributing storage and computation across many servers, the combined storage resource can grow linearly with demand while remaining economical at every amount of storage.

- As we know Hadoop works in **master-slave** fashion, HDFS also has 2 types of nodes that work in the same manner. There are **namenode(s)** and **datanodes** in the cluster..



HDFS Master (Namenode)

- Namenode regulates file access to the clients. It maintains and manages the slave nodes and assign tasks to them. Namenode executes file system namespace operations like opening, closing, and renaming files and directories. It should be deployed on reliable hardware.
- It is the master daemon that maintains and manages the DataNodes (slave nodes)
- It records the metadata of all the blocks stored in the cluster, e.g. location of blocks stored, size of the files, permissions, hierarchy, etc.
- It records each and every change that takes place to the file system metadata
- If a file is deleted in HDFS, the NameNode will immediately record this in the EditLog
- It regularly receives a Heartbeat and a block report from all the DataNodes in the cluster to ensure that the DataNodes are live
- It keeps a record of all the blocks in the HDFS and DataNode in which they are stored
- It has high availability and federation features which I will discuss in ***HDFS architecture*** in detail
- **Namenode is** Stores meta data, It keeps metadata in memory

Task of NameNode

- Manage file system namespace.
- Regulates client's access to files.
- It also executes file system execution such as naming, closing, opening files/directories.
- All DataNodes sends a Heartbeat and block report to the NameNode in the **Hadoop cluster.** It ensures that the DataNodes are alive. A block report contains a list of all blocks on a datanode.
- NameNode is also responsible for taking care of the *Replication Factor* of all the blocks.

HDFS Slave (Datanode)

- There are a number of slaves or datanodes in Hadoop Distributed File System which manage storage of data. These slave nodes are the actual worker nodes which do the tasks and serve read and write requests from the file system's clients.
- They also perform block creation, deletion, and replication upon instruction from the NameNode. Once a block is written on a datanode, it replicates it to other datanode and process continues until the number of replicas mentioned is created.
- Datanodes can be deployed on commodity Hardware and we need not deploy them on very reliable hardware.

HDFS Slave (Datanode)

- The data node is a commodity hardware having the GNU/Linux operating system and data node software. For every node (Commodity hardware/System) in a cluster, there will be a data node. These nodes manage the data storage of their system.
- Data nodes perform read-write operations on the file systems, as per client request.
- They also perform operations such as block creation, deletion, and replication according to the instructions of the name node.
- It is the slave daemon which run on each slave machine
- The actual data is stored on DataNodes
- It is responsible for serving read and write requests from the clients
- It is also responsible for creating blocks, deleting blocks and replicating the same based on the decisions taken by the NameNode
- It sends heartbeats to the NameNode periodically to report the overall health of HDFS, by default, this frequency is set to 3 seconds

Hadoop HDFS Daemons

- There are 2 daemons which run for HDFS for data storage:
- **Namenode:** This is the daemon that runs on all the masters. Name node stores metadata like filename, the number of blocks, number of replicas, a location of blocks, block IDs etc. This metadata is available in memory in the master for faster retrieval of data. In the local disk, a copy of metadata is available for persistence. So name node memory should be high as per the requirement.
- **Datanode:** This is the daemon that runs on the slave. These are actual worker nodes that store the data.

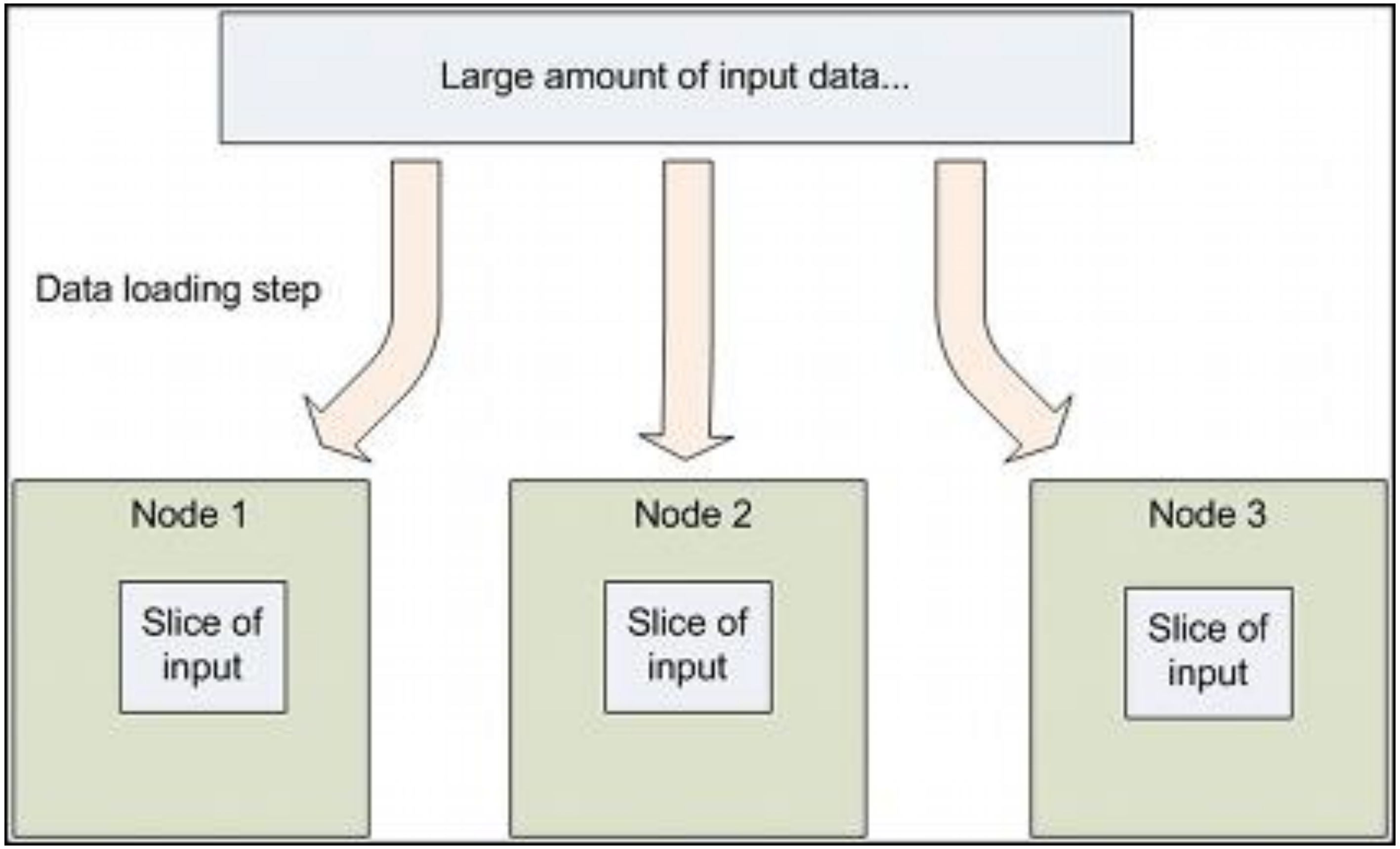
What is Secondary NameNode?

- In HDFS, when NameNode starts, first it reads HDFS state from an image file, FsImage. After that, it applies edits from the edits log file. NameNode then writes new HDFS state to the FsImage. Then it starts normal operation with an empty edits file. At the time of start-up, NameNode merges FsImage and edits files, so the edit log file could get very large over time. A side effect of a larger edits file is that next restart of Namenode takes longer.
- **Secondary Namenode** solves this issue. Secondary NameNode downloads the FsImage and EditLogs from the NameNode. And then merges EditLogs with the FsImage (FileSystem Image). It keeps edits log size within a limit. It stores the modified FsImage into persistent storage. And we can use it in the case of NameNode failure.
- Secondary NameNode performs a regular checkpoint in HDFS.

What is Backup Node?

- A **Backup node** provides the same checkpointing functionality as the Checkpoint node. In Hadoop, Backup node keeps an in-memory, up-to-date copy of the file system namespace.
- It is always synchronized with the active NameNode state. The backup node in HDFS Architecture does not need to download FsImage and edits files from the active NameNode to create a checkpoint. It already has an up-to-date state of the namespace state in memory. The Backup node checkpoint process is more efficient as it only needs to save the namespace into the local FsImage file and reset edits. NameNode supports one Backup node at a time.
- This was about the different types of nodes in HDFS Architecture. Further in this HDFS Architecture tutorial we will learn about the Blocks in HDFS, Replication Management, Rack awareness and read/write operations.

Name Node & DataNode



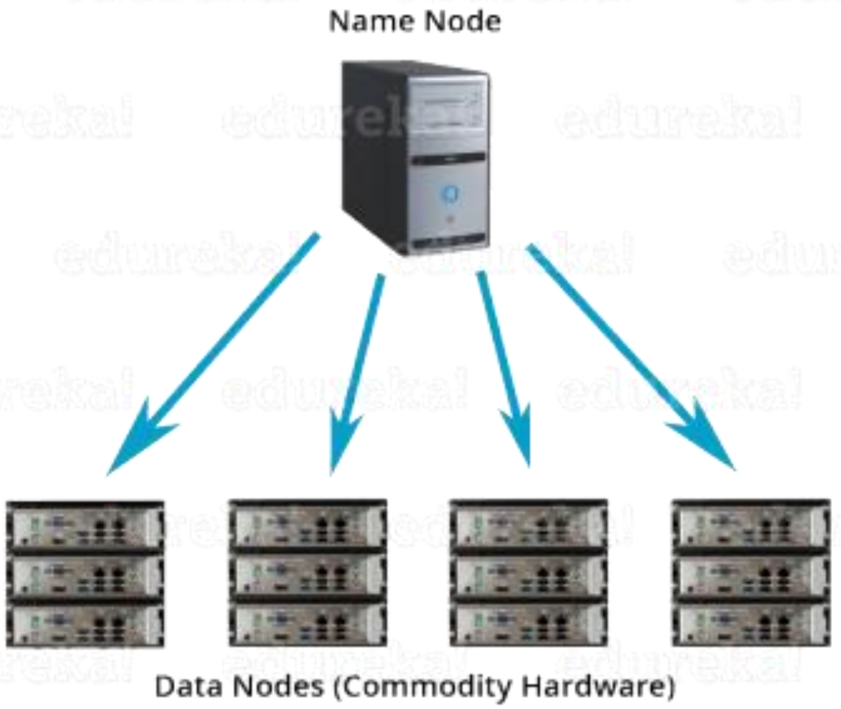
An HDFS cluster is comprised of a NameNode, which manages the cluster metadata, and DataNodes that store the data. Files and directories are represented on the NameNode by inodes. Inodes record attributes like permissions, modification and access times, or namespace and disk space quotas.

Expaination of Name Node & DataNode

- The file content is split into large blocks (typically 128 megabytes), and each block of the file is independently replicated at multiple DataNodes. The blocks are stored on the local file system on the DataNodes.
- The Namenode actively monitors the number of replicas of a block. When a replica of a block is lost due to a DataNode failure or disk failure, the NameNode creates another replica of the block. The NameNode maintains the namespace tree and the mapping of blocks to DataNodes, holding the entire namespace image in RAM.

NameNode / DataNode

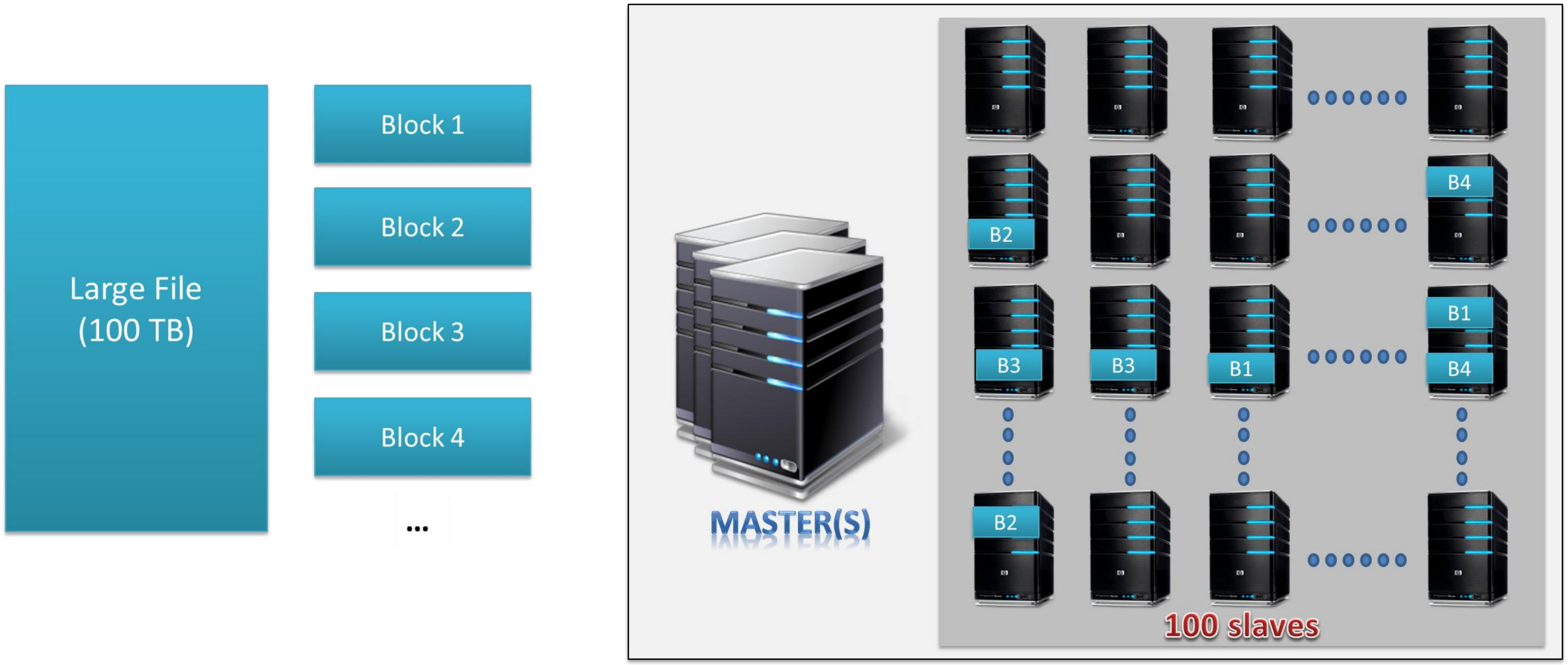
- The NameNode does not directly send requests to DataNodes. It sends instructions to the DataNodes by replying to heartbeats sent by those DataNodes. The instructions include
  - replicate blocks to other nodes,
  - remove local block replicas,
  - re-register and send an immediate block report, or
  - shut down the node.
  - commands to:


Name Node
Data Nodes (Commodity Hardware)

Data storage in HDFS

- Whenever any file has to be written in HDFS, it is broken into small pieces of data known as blocks. HDFS has a default block size of **128 MB** which can be increased as per the requirements. These blocks are stored in the cluster in distributed manner on different nodes. This provides a mechanism for **MapReduce** to process the data in parallel in the cluster.
- Multiple copies of each block are stored across the cluster on different nodes. This is a replication of data. By default, HDFS replication factor is 3. It provides fault tolerance, reliability, and **high availability**.
- A Large file is split into n number of small blocks. These blocks are stored at different nodes in the cluster in a distributed manner. Each block is replicated and stored across different nodes in the cluster.

Data storage in HDFS



Blocks

- As HDFS splits huge files into small chunks known as blocks. Block is the smallest unit of data in a filesystem. We (client and admin) do not have any control on the block like block location. Namenode decides all such things.
- HDFS default block size is 128 MB which can be increased as per the requirement. This is unlike OS filesystem where the block size is 4 KB.
- If the data size is less than the block size of HDFS, then block size will be equal to the data size. For example, if the file size is 129 MB, then 2 blocks will be created for it. One block will be of default size 128 MB and other will be 1 MB only and not 128 MB as it will waste the space (here block size is equal to data size). Hadoop is intelligent enough not to waste rest of 127 MB. So it is allocating 1 MB block only for 1 MB data.
- The major advantage of storing data in such block size is that it saves disk seek time and another advantage is in the case of processing as **mapper** processes 1 block at a time. So 1 mapper processes large data at a time.
- In Hadoop Distributed file system the file is split into blocks and each block is stored at different nodes with default 3 replicas of each block. Each replica of a block is stored at the different node to provide fault tolerant feature and the placement of these blocks on the different node is decided by Name node. Name node makes it as much distributed as possible. While placing a block on a data node, it considers how much a particular data node is loaded at that time.

What is Replication Management?

- Block replication provides **fault tolerance**. If one copy is not accessible and corrupted then we can read data from other copy. The number of copies or replicas of each block of a file is **replication factor**. The default replication factor is 3 which are again configurable. So, each block replicates three times and stored on different DataNodes.
- If we are storing a file of 128 MB in HDFS using the default configuration, we will end up occupying a space of 384 MB (3*128 MB).
- NameNode receives block report from DataNode periodically to maintain the replication factor. When a block is over-replicated/under-replicated the NameNode add or delete replicas as needed.

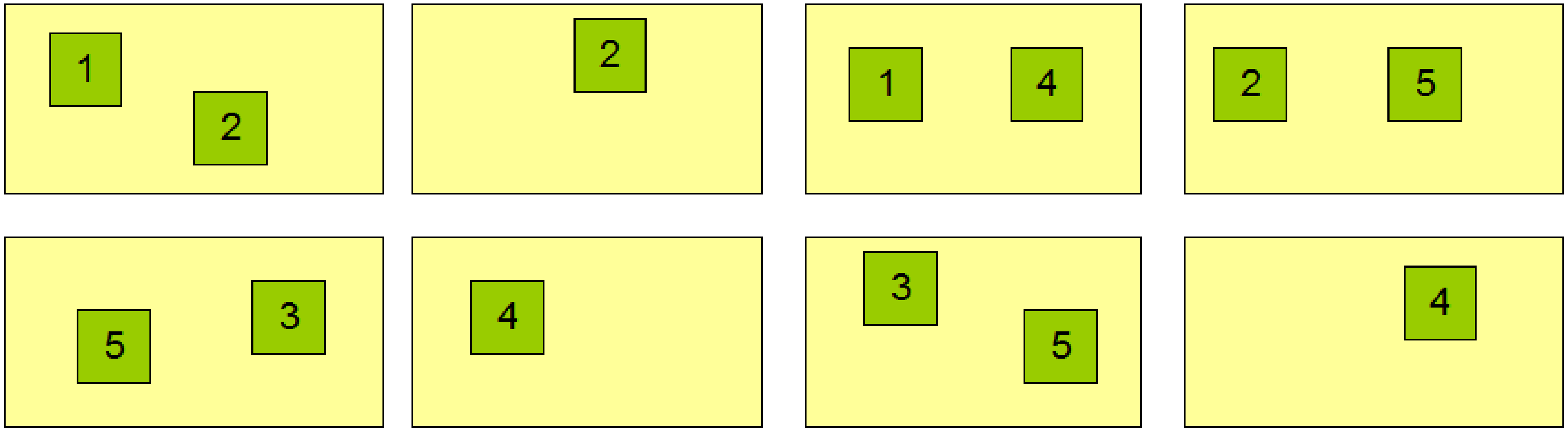
Data Replication Concept

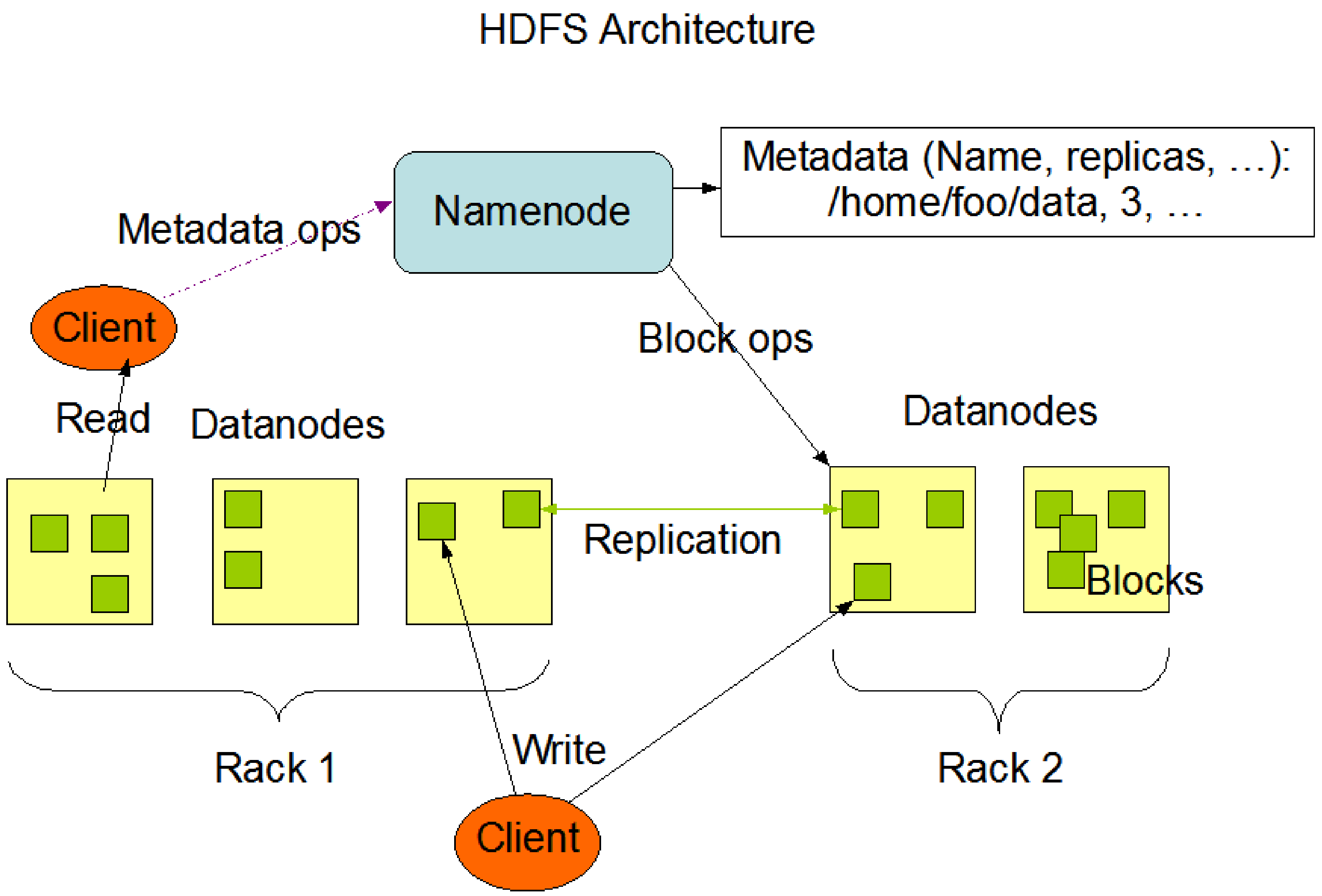
- HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once and have strictly one writer at any time.
- The NameNode makes all decisions regarding replication of blocks. It periodically receives a Heartbeat and a Blockreport from each of the DataNodes in the cluster. Receipt of a Heartbeat implies that the DataNode is functioning properly. A Blockreport contains a list of all blocks on a DataNode.

Block Replication

Namenode (Filename, numReplicas, block-ids, …)
/users/sameerp/data/part-0, r:2, {1,3}, …
/users/sameerp/data/part-1, r:3, {2,4,5}, …

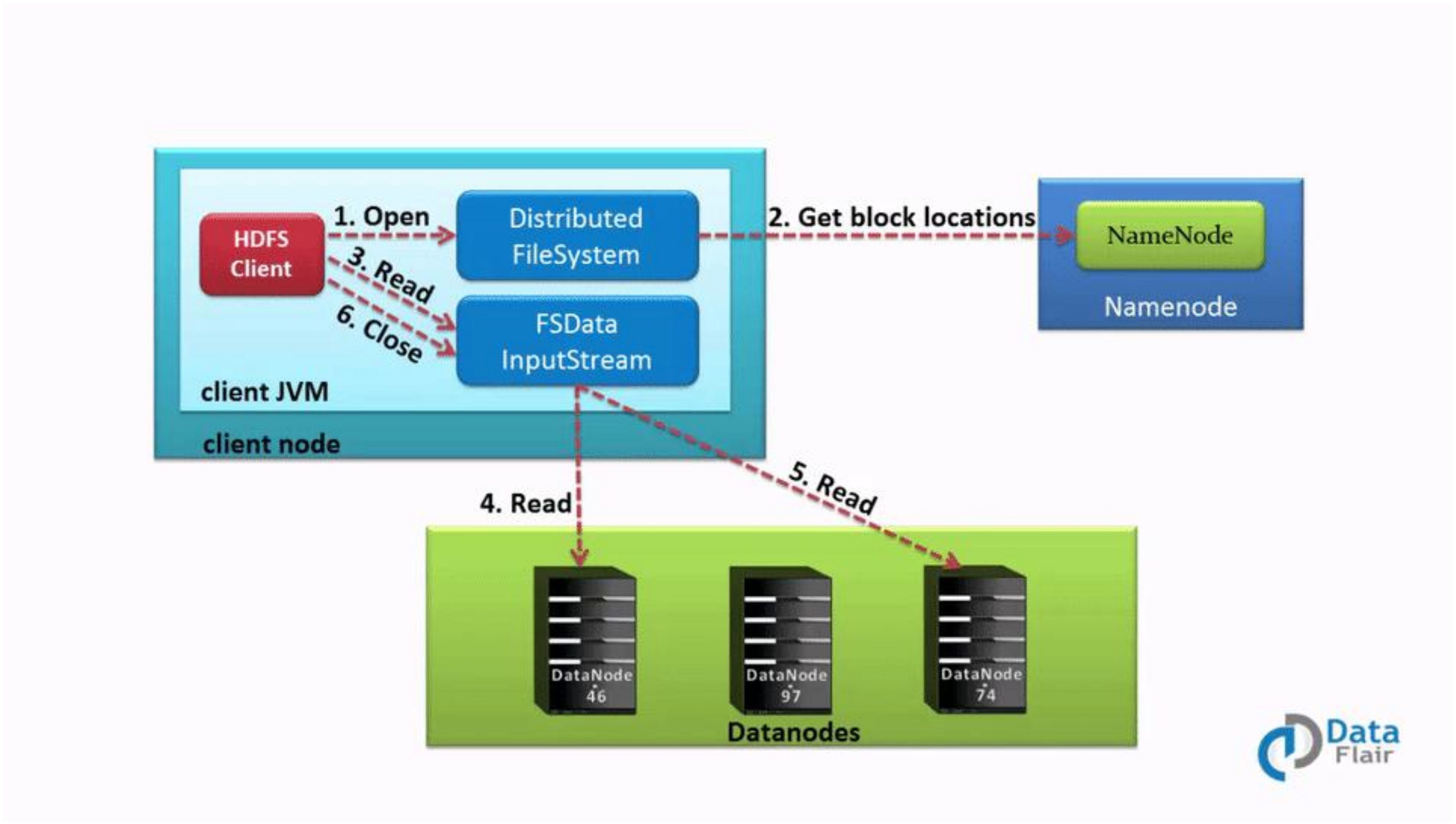
Datanodes





HDFS holds very large amount of data and provides easier access.
To store such huge data, the files are stored across multiple machines. These files are stored in redundant fashion to rescue the system from possible data losses in case of failure. HDFS also makes applications available to parallel processing.

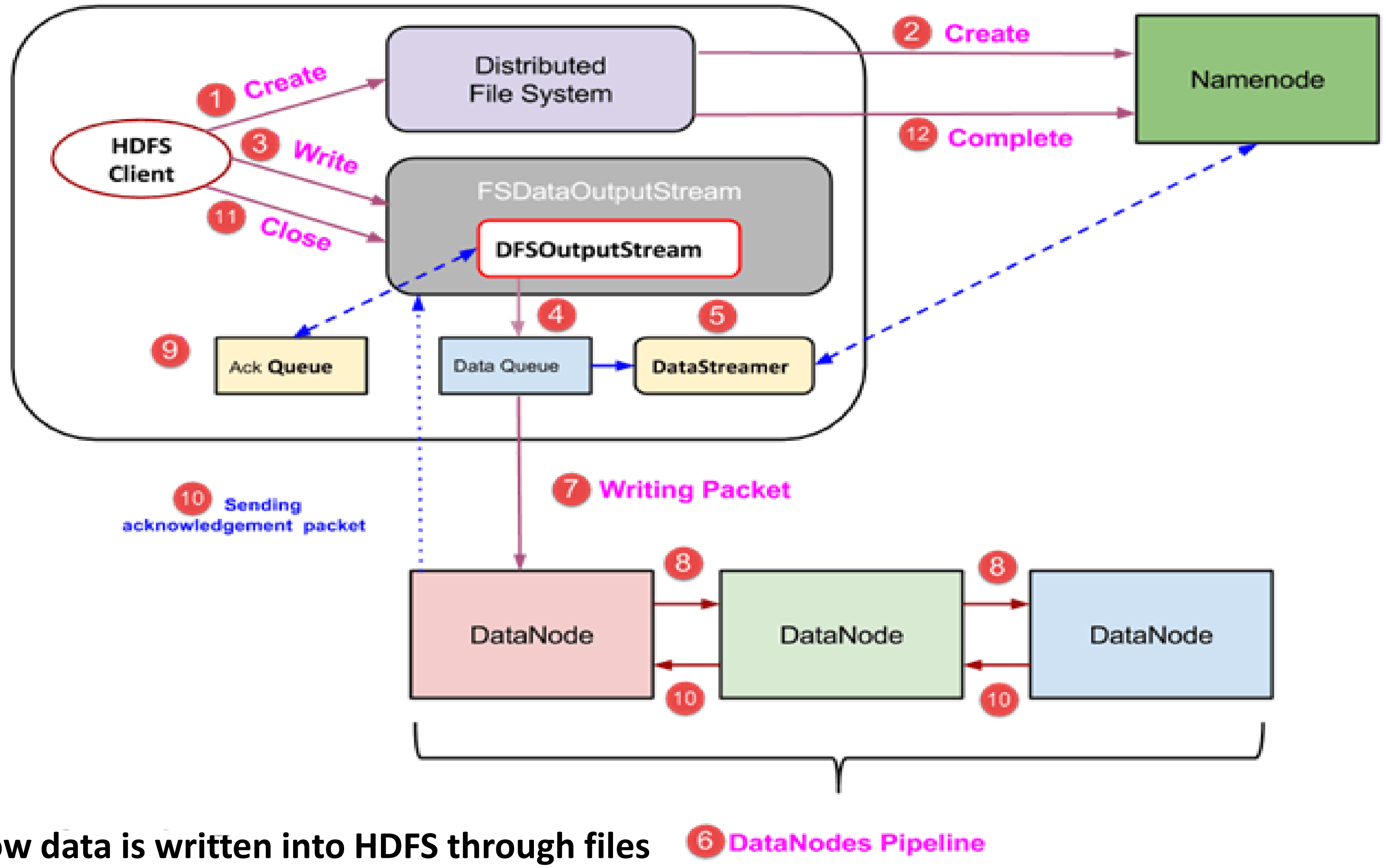
HDFS Read Operation



Explaination HDFS Read Operation

- Whenever a client wants to read any file from HDFS, the client needs to interact with namenode as Namenode is the only place which stores metadata about data nodes. Namenode specifies the address or the location of the slaves where data is stored. The client will interact with the specified data nodes and read the data from there. For security/authentication purpose, namenode provides token to the client which it shows to the data node for reading the file.
- In the HDFS read operation if the client wants to read data that is stored in HDFS, it needs to interact with namenode first. So client interacts with distributed file system API and sends a request to namenode to send block location. Thus, Namenode checks if the client is having sufficient privileges to access the data or not? Then namenode will share the address at which data is stored in data node.
- With the address, namenode also shares a security token with the client which it needs to show to datanode before accessing the data for authentication purpose.
- When a client goes to datanode for reading the file, after checking the token, datanode allows the client to read that particular block. A client then opens input stream and starts reading data from the specified datanodes. Hence, In this manner, the client reads data directly from datanode.
- If during the reading of a file datanode goes down suddenly, then a client will again go to namenode and namenode will share other location where that block is present

Write Operation In HDFS



how data is written into HDFS through files

Explaination of Write Operation In HDFS

- Client initiates write operation by calling 'create()' method of DistributedFileSystem object which creates a new file - Step no. 1 in above diagram.
- DistributedFileSystem object connects to the NameNode using RPC call and initiates new file creation. However, this file create operation does not associate any blocks with the file. It is the responsibility of NameNode to verify that the file (which is being created) does not exist already and client has correct permissions to create new file. If file already exists or client does not have sufficient permission to create a new file, then **IOException** is thrown to client. Otherwise, operation succeeds and a new record for the file is created by the NameNode.
- Once new record in NameNode is created, an object of type FSDataOutputStream is returned to the client. Client uses it to write data into the HDFS. Data write method is invoked (step 3 in diagram).
- FSDataOutputStream contains DFSOutputStream object which looks after communication with DataNodes and NameNode. While client continues writing data, **DFSOutputStream** continues creating packets with this data. These packets are en-queued into a queue which is called as **DataQueue**.
- There is one more component called **DataStreamer** which consumes this **DataQueue**. DataStreamer also asks NameNode for allocation of new blocks thereby picking desirable DataNodes to be used for replication.
- Now, the process of replication starts by creating a pipeline using DataNodes. In our case, we have chosen replication level of 3 and hence there are 3 DataNodes in the pipeline.

Explaination of Write Operation In HDFS

- The DataStreamer pours packets into the first DataNode in the pipeline.
- Every DataNode in a pipeline stores packet received by it and forwards the same to the second DataNode in pipeline.
- Another queue, 'Ack Queue' is maintained by DFSOutputStream to store packets which are waiting for acknowledgement from DataNodes.
- Once acknowledgement for a packet in queue is received from all DataNodes in the pipeline, it is removed from the 'Ack Queue'. In the event of any DataNode failure, packets from this queue are used to reinitiate the operation.
- After client is done with the writing data, it calls close() method (Step 9 in the diagram) Call to close(), results into flushing remaining data packets to the pipeline followed by waiting for acknowledgement.
- Once final acknowledgement is received, NameNode is contacted to tell it that the file write operation is complete.

Hadoop HDFS Commands

Hadoop HDFS Commands

- cat
- chgrp
- chmod
- chown
- copyFromLocal
- copyToLocal
- count
- cp
- du
- dus
- expunge
- get
- getfacl
- getmerge
- ls

lsr
mkdir
moveFromLocal
moveToLocal
mv
put
rm
rmr
setfacl
setrep
stat
tail
test
text
touchz

mkdir

- **Hadoop HDFS mkdir Command Usage**

```
mkdir <path>
```

- **Hadoop HDFS mkdir Command Example**

```
hdfs dfs -mkdir /user/dataflair/dir1
```

- **Hadoop HDFS mkdir Command Description**
  This HDFS command takes path URI’s as an argument and creates directories.
  Creates any parent directories in path that are missing (e.g., mkdir -p in Linux).

ls

- **Hadoop HDFS ls Command Usage**

```
ls <path>
```

- **Hadoop HDFS ls Command Example**

```
hdfs dfs -ls /user/dataflair/dir1
```

- **Hadoop HDFS ls Commnad Description**
  This Hadoop HDFS ls command displays a list of the contents of a directory specified by path provided by the user, showing the names, permissions, owner, size and modification date for each entry.
- **Hadoop HDFS ls Command Example**

```
hdfs dfs -ls -R
```

put

- **Hadoop HDFS put Command Usage**

```
put <localSrc> <dest>
```

- **Hadoop HDFS put Command Example**

```
hdfs dfs -put /home/dataflair/Desktop/sample
/user/dataflair/dir1
```

- **Hadoop HDFS put Command Description**
  This hadoop basic command copies the file or directory from the local file system to the destination within the DFS.

copyFromLocal

- **Hadoop HDFS copyFromLocal Command Usage**

```
copyFromLocal <localSrc> <dest>
```

- **Hadoop HDFS copyFromLocal Command Example**

```
hdfs dfs -copyFromLocal
    /home/dataflair/Desktop/sample /user/dataflair/dir1
```

- **Hadoop HDFS copyFromLocal Command Description** This hadoop shell command is similar to put command, but the source is restricted to a local file reference.

get

- **Hadoop HDFS get Command Description**
  This HDFS fs command copies the file or directory in HDFS identified by the source to the local file system path identified by local destination. This HDFS basic command retrieves all files that match to the source path entered by the user in HDFS, and creates a copy of them to one single, merged file in the local file system identified by local destination.
- **Hadoop HDFS get Command Usage**

  ```
  get [-crc] <src> <localDest>
  ```
- **Hadoop HDFS get Command Example**

  ```
  hdfs dfs -get /user/dataflair/dir2/sample /home/dataflair/Desktop
  ```
- **Hadoop HDFS get Command Example**

  ```
  hdfs dfs -getmerge /user/dataflair/dir2/sample
  /home/dataflair/Desktop
  ```

get

- **Hadoop HDFS get Command Description**
  This apache Hadoop command shows the Access Control Lists (ACLs) of files and directories. If a directory contains a default ACL, then getfacl also displays the default ACL.
  Options :
  -R: It displays a list of all the ACLs of all files and directories recursively.
  path: File or directory to list.
- This HDFS file system command displays if there is any extended attribute names and values for a file or directory.
  Options:
  -R: It recursively lists the attributes for all files and directories.
  -n name: It displays the named extended attribute value.
  -d: It displays all the extended attribute values associated with the pathname.
  -e encoding: Encodes values after extracting them. The valid converted coded forms are “text”, “hex”, and “base64”. All the values encoded as text strings are with double quotes (” “), and prefix 0x and 0s are used for all the values which are converted and coded as hexadecimal and base64.
  path: The file or directory.

copyToLocal

- **Hadoop HDFS copyToLocal Command Usage**
- copyToLocal <src> <localDest>
- **Hadoop HDFS copyToLocal Command Example**
- hdfs dfs -copyToLocal /user/dataflair/dir1/sample /home/dataflair/Desktop
- **Hadoop HDFS copyToLocal Description**
  Similar to get command, only the difference is that in this the destination is restricted to a local file reference.

cat

- **Hadoop HDFS cat Command Usage**
- cat <file-name>
- **Hadoop HDFS cat Command Example**
- hdfs dfs -cat /user/dataflair/dir1/sample
- **Hadoop HDFS cat Command Description**
  This Hadoop fs shell command displays the contents of the filename on console or stdout.

moveFromLocal

- **moveFromLocalHDFS moveFromLocal Command Usage**
- moveFromLocal <localSrc> <dest>
- **HDFS moveFromLocal Command Example**
- hdfs dfs -moveFromLocal /home/dataflair/Desktop/sample /user/dataflair/dir1
- **HDFS moveFromLocal Command Description**
  This HDFS command copies the file or directory from the local file system identified by the local source to destination within HDFS, and then deletes the local copy on success.

moveToLocal

- **HDFS moveToLocal Command Usage**
- moveToLocal <src> <localDest>
- **HDFS moveToLocal Command Example**
- hdfs dfs -moveToLocal /user/dataflair/dir2/sample /user/dataflair/Desktop
- **HDFS moveToLocal Command Description** This hadoop basic command works like -get, but deletes the HDFS copy on success.

rm

- **HDFS rm Command Usage**
- rm <path>
- **HDFS rm Command Example**
- hdfs dfs -rm /user/dataflair/dir2/sample
- **HDFS rm Command Description**
  This Hadoop command removes the file or empty directory present on the path provided by the user.
- **HDFS rm Command Example**
- hdfs dfs -rm -r /user/dataflair/dir2

chown

- **HDFS chown Command Usage**
- hdfs dfs -chown [-R] [OWNER][:[GROUP]] URI [URI]
- **HDFS chown Command Example**
- hdfs dfs -chown -R dataflair /opt/hadoop/logs
- **HDFS chown Command Description**
  This Hadoop HDFS Commmand changes the owner of files. With -R, changes are made recursively by way of the structure of the directory. A user should be the superuser.

Reference

- https://youtu.be/1_ly9dZnmWc
- https://hadoop.apache.org/docs/r2.4.1/hadoop-project-dist/hadoop-common/FileSystemShell.html
- https://www.tutorialspoint.com/hadoop/hadoop_command_reference.htm
- https://www.edureka.co/blog/hdfs-tutorial