Curso de extensão em Data Science

GERÊNCIA DE INFRAESTRUTURA PARA BIG DATA

Prof. Tiago Ferreto – tiago.ferreto@pucrs.br

PUCRS | ESCOLA POLITÉCNICA

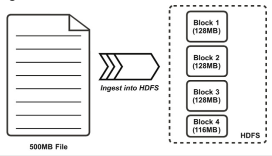
HDFS – HADOOP DISTRIBUTED FILE SYSTEM

HDFS Overview

- HDFS is Hadoop's primary input data source and target for data processing operations
  - Other filesystems are also supported
- Inspired by GoogleFS whitepaper (2003)
  - Focus on supporting the storage requirements of search engines
- Key design principles
  - Scalable (economically)
  - Fault tolerant
  - Uses commodity hardware
  - Supports high concurrency
  - Favors high sustained bandwidth over low latency random access

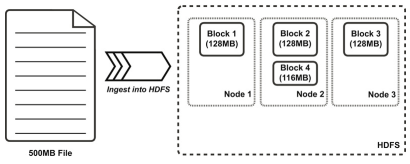
Files, Blocks, and Replication

- HDFS is a **virtual** filesystem
  - It appears as a single system, but its data is located in multiple different locations
  - It is deployed on top of native filesystems (e.g., ext3, ext4, xfs)
- Data stored in HDFS is immutable – **cannot** be update after being committed
  - WORM (write once, read many) filesystem
- Files are **split into blocks** when ingested into HDFS
  - Default size = 128 MB (configurable)



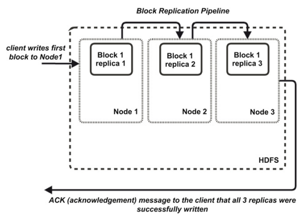
Files, Blocks, and Replication

- Blocks are **distributed**
  - Blocks are distributed among slave nodes in the cluster upon ingestion (considering a multi-node cluster)
  - This enables: shared nothing and parallel processing of data



Files, Blocks, and Replication

- Blocks are **replicated**
  - Replication occurs according to a preconfigured replication factor
    - Typically set to 3 (in a fully distributed cluster environment with 3 or more nodes)
    - In a pseudo-distributed Hadoop cluster this value is set to 1 (there is only one DataNode)
- Replication happens upon ingestion
- Goals for replication
  - Increase opportunities for data locality
  - Provide fault tolerance



Block Replication Pipeline
client writes first block to Node1
Block 1 replica 1
Node 1
Block 1 replica 2
Node 2
Block 1 replica 3
Node 3
HDFS
ACK (acknowledgement) message to the client that all 3 replicas were successfully written

DataNode or Block Failure Recovery

- Each object in HDFS has a replication factor
- NameNode gets regular block inventories (block reports) from each DataNode in the cluster
  - Default interval is 21600000 miliseconds = 6 hours (dfs.blockreport.intervalMsec in hdfs-site.xml)
  - NameNode verifies which blocks are corrupted or under-replicated (possibly due to a DataNode failure)
- NameNode also receives regular heartbeats to check DataNode's health
  - Default interval time is 3 seconds (dfs.heartbeat.interval in hdfs-site.xml)
  - NameNode waits for up to 10 missed heartbeats (30 seconds) before assuming a DataNode is dead
- When the NameNode detects that a block does not have the right number of replicas, it instructs a DataNode (with a valid replica) to replicate that block to another node

NameNode

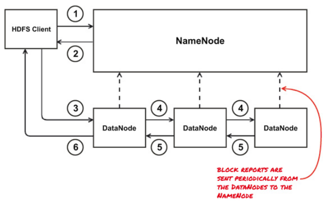
- HDFS master node process – coordinates the distributed filesystem
- Manages filesystem's metadata
  - Contains all directory and file objects with their properties and attributes (ACLs – Access Control Lists) – define users or groups with access to objects
  - Stored in memory – service client queries with low latency
  - Uses snapshot and journaling functions to ensure durability and crash consistency
  - Includes the locations of the blocks which comprise files in HDFS
    - Only representation of the relationship between files and blocks in HDFS
- Services queries from clients (via CLI, MapReduce, Spark, or other other application)
  - Clients interact with the NameNode to get block information and access directly DataNodes → NameNode is not used in the data read/write (Avoid bottleneck)

DataNodes

- Cluster nodes on which HDFS blocks are stored and managed
- Responsabilities
  - Participate in the block replication pipeline
  - Manage local volumes and storage
  - Provide block reports to the NameNode
    - Regular messages with an inventory of blocks stored on the DataNode
- Checksums are calculated upon ingestion into HDFS and are kept with the blocks
  - DataNode recalculates and compares there checkums periodically and reports mismatches to the NameNode
    - Since the filesystem is immutable, checksums never change
- Blocks are stored on local volumes on the DataNodes
- DataNodes only store HDFS blocks. There is no information regarding files or directories structure → It is not possible to reconstruct the filesystem if the NameNode's medatada is lost!

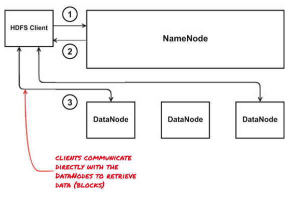
Writing Files into HDFS

1. HDFS Client requests to write a file block.
2. NameNode responds to the Client with the DataNode to which to write the block.
3. Client requests to write the block to the specified DataNode.
4. DataNode opens a block replication pipeline with another DataNode in the cluster, and this process continues until all configured replicas are written.
5. A write acknowledgment is sent back through the replication pipeline.
6. Client is informed that the write operation was successful.


HDFS Client
NameNode
DataNode
DataNode
DataNode
1
2
3
4
4
5
5
6
BLOCK REPORTS ARE SENT PERIODICALLY FROM THE DATANODES TO THE NAMENODE

Reading files from HDFS

1. HDFS Client requests to read a file
2. NameNode responds to the request with a list of DataNodes containing the blocks that comprise the file (all replicas)
   - If a DataNode is not available or a block is corrupt, Client can use another replica
3. Client communicates directly with DataNodes to retrieve blocks for the file.


HDFS Client
NameNode
DataNode
DataNode
DataNode
1
2
3
CLIENTS COMMUNICATE DIRECTLY WITH THE DATANODES TO RETRIEVE DATA (BLOCKS)

NameNode Metadata

- Most critical component in HDFS
  - Contains: link between blocks and DataNodes, and files and directories structure and attributes
- Resides in memory for fast lookup by clients
- Example (conceptual representation)

object	block_id	seq	locations	ACL	Checksum
/data/file.txt	blk_00123	1	[node1,node2,node3]	-rwxrwxrwx	8743b52063 ..
/data/file.txt	blk_00124	2	[node2,node3,node4]	-rwxrwxrwx	cd84097a65 ..
/data/file.txt	blk_00125	3	[node2,node4,node5]	-rwxrwxrwx	d1633f5c74 ..

Amount and size of files

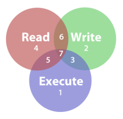
- Recommendation
  - HDFS prefers fewer larger files
    - Each object in HDFS (file or directory) consumes approximately 150-200 bytes of memory on the NameNode → fewer larger files are preferred over many smaller files
- Considering the ingestion of 10GB into HDFS with 128MB block size

Files	Name Entries	Block Metadata	Total Objects
10 × 1GB files	10	80	90
10,000 × 1MB files	10,000	10,000	20,000

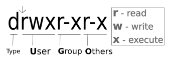
- Files can be concatenated upon ingestion

HDFS Access Control Lists and Permissions

- HDFS Objects have associated ACLs
  - Define object owner and permissions
- Uses a Unix-style object permissions mask
  - Bits with permission to: 4-read (r), 2-write (w), and 1-execute (x)
  - Execute is only used for directories (HDFS does not have executables)



- Example (using hdfs user)
  - `$ sudo -u hdfs hadoop fs -chown ferreto /data/books`
  - `$ sudo -u hdfs hadoop fs -chmod 777 /data/books`



- Caution: HDFS Security is considered weak!
  - It is advised to use additional security methods (e.g. Kerberos) in production clusters

On-Disk Structures and Consistency

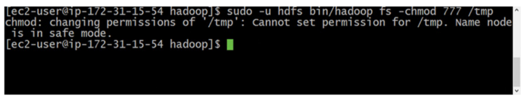
- NameNode metadata on-disk representation – consists of two components:
  - **fsimage file**
    - a point-in-time snapshot of the metadata without the specific block locations
    - typically only written to at the end of recovery or by the SecondaryNameNode
  - **edits files**
    - contains updates to the metadata
    - updated for every filesystem change - new data, deleted data, modified permissions (similar to a transaction log in a traditional database)

NameNode's Recovery

- NameNode's recovery process – happens on NameNode's startup
  1. fsimage snapshot is mounted
  2. edits (updates) are applied in sequence
  3. a new fsimage is created for the next recovery process
  4. new edits files are created to capture new changes post-recovery
- After recovery process, DataNodes start sending their block reports to the NameNode
  - NameNode starts associating block locations for block replicas in the in-memory representation of the metadata
- Block locations may change due to replication or rebalancing operations → only persist in memory and are not written in fsimage or edits file

SafeMode

- In SafeMode, HDFS allows only read operations
  - Used during NameNode startup and recovery processes



```
[ec2-user@ip-172-31-15-54 hadoop]$ sudo -u hdfs bin/hadoop fs -chmod 777 /tmp
chmod: changing permissions of '/tmp': Cannot set permission for /tmp. Name node
 is in safe mode.
[ec2-user@ip-172-31-15-54 hadoop]$
```

SecondaryNameNode

- Optional process located on a different host from the NameNode
- **Checkpoints** the filesystem periodically
  - Performs a recovery operation on behalf of the primary NameNode (performs the same sequence of operations as the primary NameNode)
  - Gets **fsimage** file, applies updates from **edits** file, and creates a new **fsimage** file
  - New **fsimage** file is replaced on the primary NameNode
    - Shortens recovery operations and reduces disk space consumed by the **edits** files
- SecondaryNameNode is not a HA (High availabiltiy) solution!
  - It only provides alternate storage location for the on-disk representation of the NameNode's metadata in case of primary NameNode's failure
  - HA is enabled using a Standy NameNode

Interacting with HDFS

- Main access interfaces
  - Filesystem shell (hadoop fs or hdfs fs)
  - Hadoop Filesystem Java API
  - RESTful proxy interfaces – HttpFS and WebHDFS

- HDFS is based on the POSIX standard
  - Uses POSIX conventions found in Unix/Linux for representations of files and directories
- HDFS shell uses verbs similar to FTP commands (put, get, etc)

- HDFS has no concept of current directory (no cd command) → every command starts from a relative path beginning in the user's home directory in HDFS (`/user/<username>`)

HDFS Shell – Uploading (or Ingesting a File)

- Example:
  - Uploading a local file named warandpeace.txt into an existing directory in HDFS called /data/books

  - `$ hadoop fs -put warandpeace.txt /data/books/`

- Synonymous commands
  - `$ hadoop fs -copyFromLocal warandpeace.txt /data/books`
  - `$ hdfs dfs -put warandpeace.txt /data/books`

HDFS Shell – Downloading a File

- Many applications cannot interact directly with HDFS → it is necessary to retrieve the file from HDFS
- Example:
  - Retrieve a file called report.csv from /data/reports in HDFS and place it into the user's current directory
  - `$ hadoop fs -get /data/reports/report.csv`

HDFS Shell – Listing directory contents

- Example:
  - To list the contents of /data/reports
  - `$ hadoop fs -ls /data/reports`

HDFS Shell – Deleting Objects

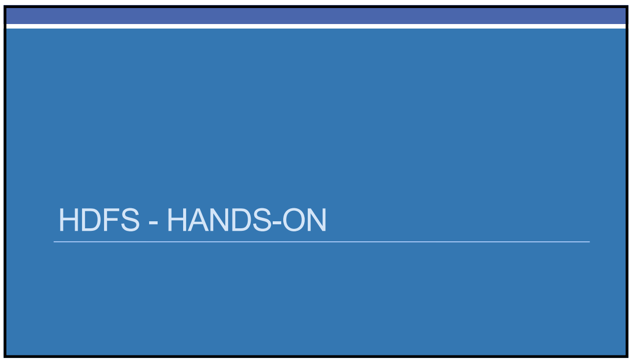
- Example:
  - To delete report.csv from /data/reports in HDFS
  - `$ hadoop fs -rm /data/reports/report.csv`

  - To delete the entire /data/reports directory
  - `$ hadoop fs -rm -r /data/reports`

HDFS Trash Folder

- HDFS has the concept of a Trash folder or Recycling Bin

- Configured by a parameter called fs.trash.interval (file hdfs-site.xml)
  - Defines the amount of time (in minutes) to keep a deleted object in a hidden Trash directory before it is permanently removed from the filesystem
  - Default is 0 → all deletes are immediate and irreversible



HDFS - HANDS-ON