

25 YEARS ANNIVERSARY
SOICT

HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY



HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Chapter 2

Hadoop distributed file system (HDFS)

Overview of HDFS

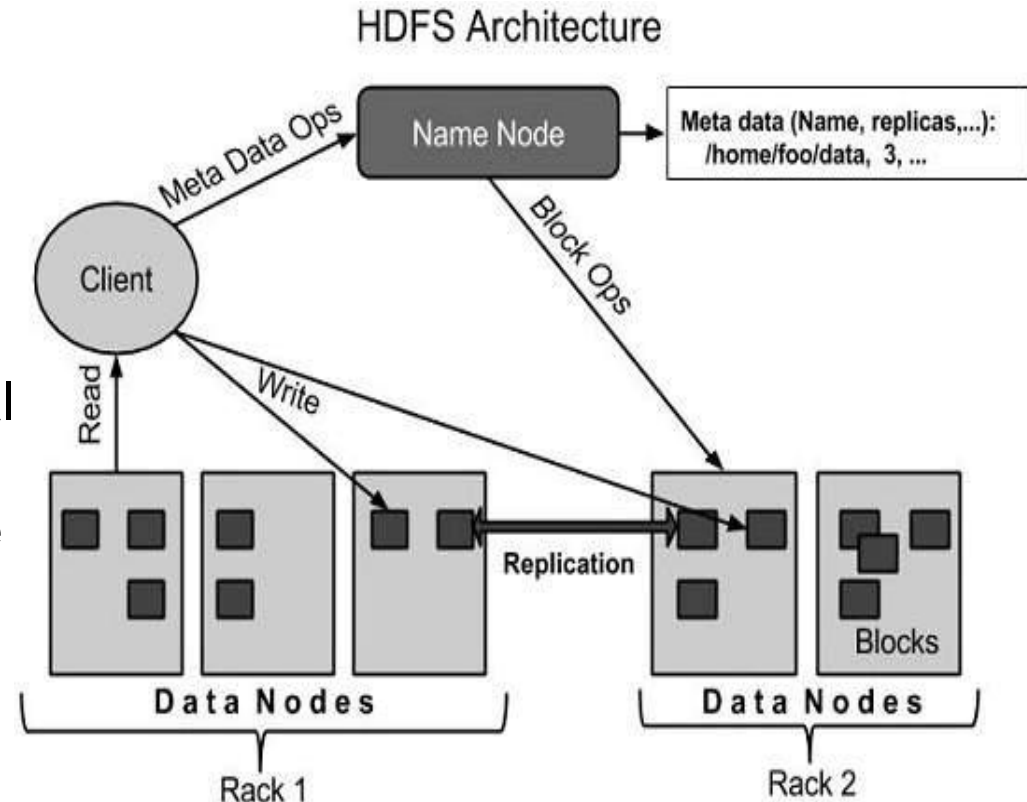
- Provides inexpensive and reliable storage for massive amounts of data
- Designed for
 - Big files (100 MB to several TBs file sizes)
 - Write once, read many times (Appending only)
 - Running on commodity hardware
- Hierarchical UNIX style file systems
 - (e.g., /hust/soict/hello.txt)
 - UNIX style file ownership and permissions

HDFS main design principles

- I/O pattern
 - Append only → reduce synchronization
- Data distribution
 - File is splitted in big chunks (64 MB)
 - reduce metadata size
 - reduce network communication
- Data replication
 - Each chunk is usually replicated in 3 different nodes
- Fault tolerance
 - Data node: re-replication
 - Name node
 - Secondary Namenode
 - Standby, Active Namenodes

HDFS Architecture

- Master/slave architecture
- HDFS master: Namenode
 - Manage namespace and metadata
 - Monitor Datanode
- HDFS slaves: Datanodes
 - Handle read/write the actual data {chunks}
 - Chunks are local files in the local file systems



Functions of a Namenode

- Manages File System Namespace
 - Maps a file name to a set of blocks
 - Maps a block to the Datanodes where it resides
- Cluster Configuration Management
- Replication Engine for Blocks

Namenode metadata

- Metadata in memory
 - The entire metadata is in main memory
 - No demand paging of metadata
- Types of metadata
 - List of files
 - List of Blocks for each file
 - List of Datanodes for each block
 - File attributes, e.g. creation time, replication factor
- A Transaction Log
 - Records file creations, file deletions etc

Datanode

- A Block Server
 - Stores data in the local file system (e.g. ext3)
 - Stores metadata of a block (e.g. CRC)
 - Serves data and metadata to Clients
- Block Report
 - Periodically sends a report of all existing blocks to the Namenode
- Facilitates Pipelining of Data
 - Forwards data to other specified Datanodes
- Heartbeat
 - Datanodes send heartbeat to the Namenode
 - Once every 3 seconds
 - Namenode uses heartbeats to detect Datanode failure

Data replication

- Chunk placement
 - Current Strategy
 - One replica on local node
 - Second replica on a remote rack
 - Third replica on same remote rack
 - Additional replicas are randomly placed
 - Clients read from nearest replicas
- Namenode detects Datanode failures
 - Chooses new Datanodes for new replicas
 - Balances disk usage
 - Balances communication traffic to Datanodes

Data rebalance

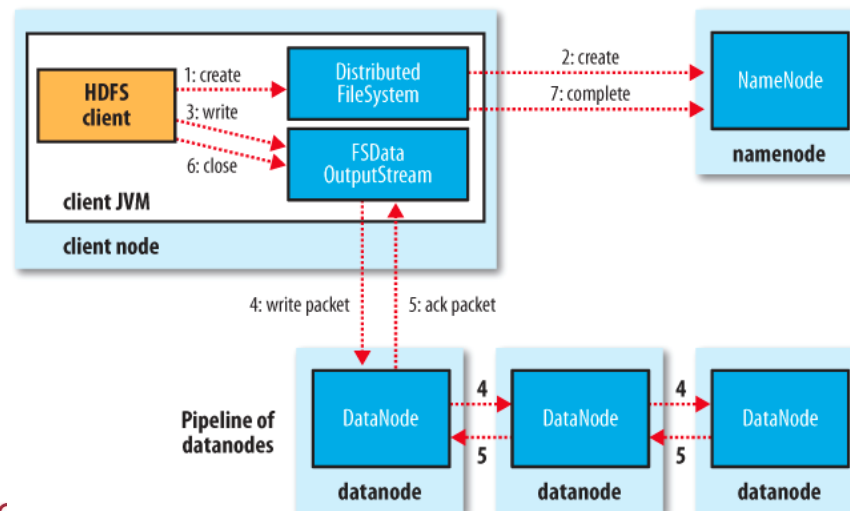
- Goal: % disk full on Datanodes should be similar
 - Usually run when new Datanodes are added
 - Cluster is online when Rebalancer is active
 - Rebalancer is throttled to avoid network congestion
 - Command line tool

Data correctness

- Use Checksums to validate data
 - Use CRC32
- File Creation
 - Client computes checksum per 512 bytes
 - Datanode stores the checksum
- File access
 - Client retrieves the data and checksum from Datanode
 - If Validation fails, Client tries other replicas

Data pipelining

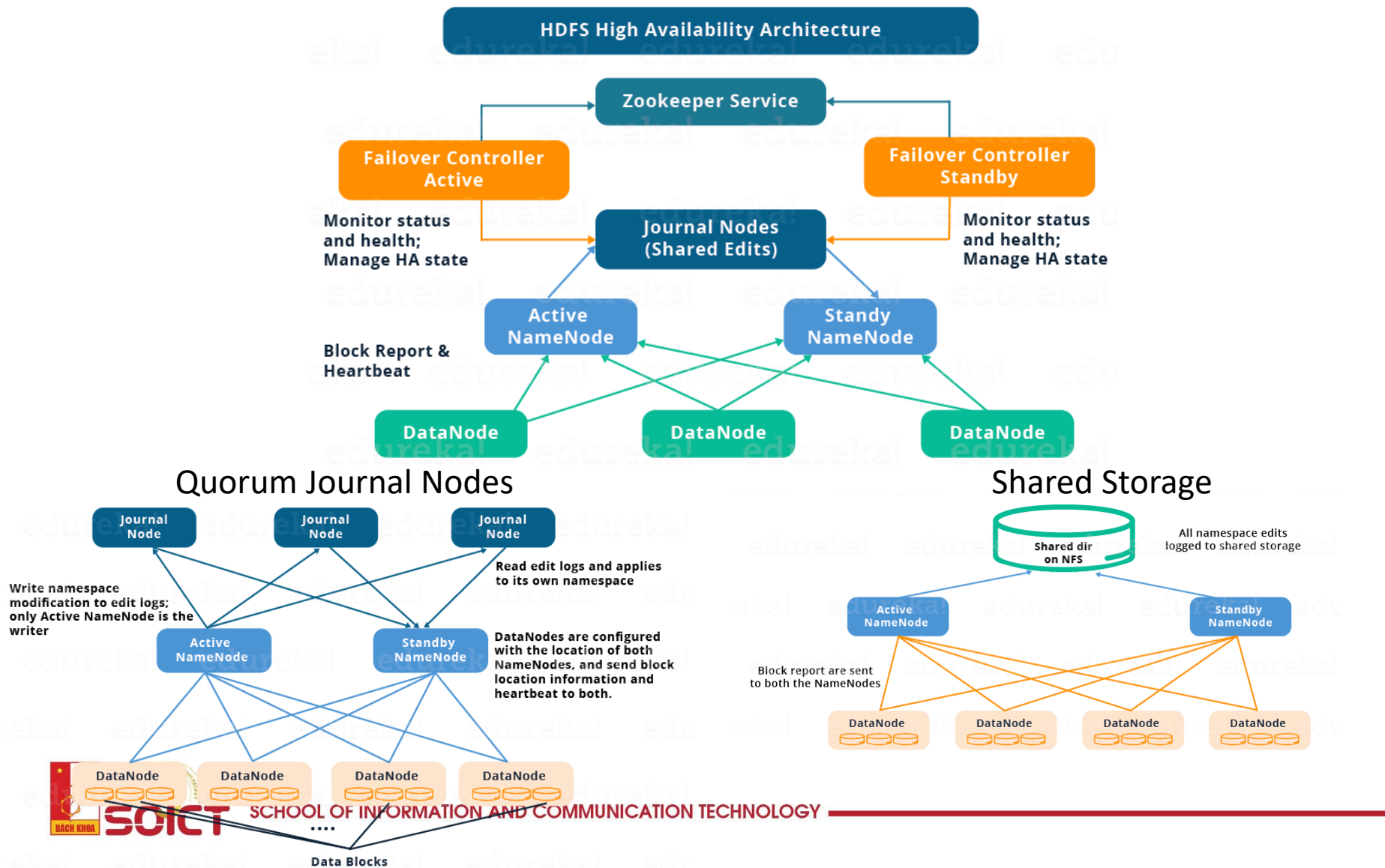
- Client retrieves a list of Datanodes on which to place replicas of a block
- Client writes block to the first Datanode
- The first Datanode forwards the data to the next node in the Pipeline
- When all replicas are written, the Client moves on to write the next block in file



Secondary Name node

- Namenode is a single point of failure
- Secondary Namenode
 - Checkpointing latest copy of the FsImage and the Transaction Log files.
 - Copies FsImage and Transaction Log from Namenode to a temporary directory
- When Namenode restarted
 - Merges FSImage and Transaction Log into a new FSImage in temporary directory
 - Uploads new FSImage to the Namenode
 - Transaction Log on Namenode is purged

Namenode high availability (HA)



HDFS command-line interface

List Files

<code>hdfs dfs -ls /</code>	List all the files/directories for the given hdfs destination path.
<code>hdfs dfs -ls -d /hadoop</code>	Directories are listed as plain files. In this case, this command will list the details of hadoop folder.
<code>hdfs dfs -ls -h /data</code>	Format file sizes in a human-readable fashion (eg 64.0m instead of 67108864).
<code>hdfs dfs -ls -R /hadoop</code>	Recursively list all files in hadoop directory and all subdirectories in hadoop directory.
<code>hdfs dfs -ls /hadoop/dat*</code>	List all the files matching the pattern. In this case, it will list all the files inside hadoop directory which starts with 'dat'.

Read/Write Files

<code>hdfs dfs -text /hadoop/derby.log</code>	HDFS Command that takes a source file and outputs the file in text format on the terminal. The allowed formats are zip and TextRecordInputStream.
<code>hdfs dfs -cat /hadoop/test</code>	This command will display the content of the HDFS file test on your stdout .
<code>hdfs dfs -appendToFile /home/ubuntu/test1 /hadoop/text2</code>	Appends the content of a local file test1 to a hdfs file test2.

Upload, download files

Upload/Download Files	
<code>hdfs dfs -put /home/ubuntu/sample /hadoop</code>	Copies the file from local file system to HDFS.
<code>hdfs dfs -put -f /home/ubuntu/sample /hadoop</code>	Copies the file from local file system to HDFS, and in case the local already exists in the given destination path, using -f option with put command will overwrite it.
<code>hdfs dfs -put -l /home/ubuntu/sample /hadoop</code>	Copies the file from local file system to HDFS. Allow DataNode to lazily persist the file to disk. Forces replication factor of 1.
<code>hdfs dfs -put -p /home/ubuntu/sample /hadoop</code>	Copies the file from local file system to HDFS. Passing -p preserves access and modification times, ownership and the mode.
<code>hdfs dfs -get /newfile /home/ubuntu/</code>	Copies the file from HDFS to local file system.
<code>hdfs dfs -get -p /newfile /home/ubuntu/</code>	Copies the file from HDFS to local file system. Passing -p preserves access and modification times, ownership and the mode.
<code>hdfs dfs -get /hadoop/*.txt /home/ubuntu/</code>	Copies all the files matching the pattern from local file system to HDFS.
<code>hdfs dfs -copyFromLocal /home/ubuntu/sample /hadoop</code>	Works similarly to the put command, except that the source is restricted to a local file reference.
<code>hdfs dfs -copyToLocal /newfile /home/ubuntu/</code>	Works similarly to the put command, except that the destination is restricted to a local file reference.
<code>hdfs dfs -moveFromLocal /home/ubuntu/sample /hadoop</code>	Works similarly to the put command, except that the source is deleted after it's copied.

File management

File Management	
<code>hdfs dfs -cp /hadoop/file1 /hadoop1</code>	Copies file from source to destination on HDFS. In this case, copying file1 from hadoop directory to hadoop1 directory.
<code>hdfs dfs -cp -p /hadoop/file1 /hadoop1</code>	Copies file from source to destination on HDFS. Passing <code>-p</code> preserves access and modification times, ownership and the mode.
<code>hdfs dfs -cp -f /hadoop/file1 /hadoop1</code>	Copies file from source to destination on HDFS. Passing <code>-f</code> overwrites the destination if it already exists.
<code>hdfs dfs -mv /hadoop/file1 /hadoop1</code>	Move files that match the specified file pattern <code><src></code> to a destination <code><dst></code> . When moving multiple files, the destination must be a directory.
<code>hdfs dfs -rm /hadoop/file1</code>	Deletes the file (sends it to the trash).
<code>hdfs dfs -rm -r /hadoop</code> <code>hdfs dfs -rm -R /hadoop</code> <code>hdfs dfs -rmr /hadoop</code>	Deletes the directory and any content under it recursively.
<code>hdfs dfs -rm -skipTrash /hadoop</code>	The <code>-skipTrash</code> option will bypass trash, if enabled, and delete the specified file(s) immediately.
<code>hdfs dfs -rm -f /hadoop</code>	If the file does not exist, do not display a diagnostic message or modify the exit status to reflect an error.
<code>hdfs dfs -rmdir /hadoop1</code>	Delete a directory.
<code>hdfs dfs -mkdir /hadoop2</code>	Create a directory in specified HDFS location.
<code>hdfs dfs -mkdir -f /hadoop2</code>	Create a directory in specified HDFS location. This command does not fail even if the directory already exists.
<code>hdfs dfs -touchz /hadoop3</code>	Creates a file of zero length at <code><path></code> with current time as the timestamp of that <code><path></code> .

Ownership and validation

Ownership and Validation	
hdfs dfs -checksum /hadoop/file1	Dump checksum information for files that match the file pattern <src> to stdout.
hdfs dfs -chmod 755 /hadoop/file1	Changes permissions of the file.
hdfs dfs -chmod -R 755 /hadoop	Changes permissions of the files recursively.
hdfs dfs -chown ubuntu:ubuntu /hadoop	Changes owner of the file. 1st ubuntu in the command is owner and 2nd one is group.
hdfs dfs -chown -R ubuntu:ubuntu /hadoop	Changes owner of the files recursively.
hdfs dfs -chgrp ubuntu /hadoop	Changes group association of the file.
hdfs dfs -chgrp -R ubuntu /hadoop	Changes group association of the files recursively.
Filesystem	
hdfs dfs -df /hadoop	Shows the capacity, free and used space of the filesystem.
hdfs dfs -df -h /hadoop	Shows the capacity, free and used space of the filesystem. -h parameter Formats the sizes of files in a human-readable fashion.
hdfs dfs -du /hadoop/file	Show the amount of space, in bytes, used by the files that match the specified file pattern.
hdfs dfs -du -s /hadoop/file	Rather than showing the size of each individual file that matches the pattern, shows the total (summary) size.
hdfs dfs -du -h /hadoop/file	Show the amount of space, in bytes, used by the files that match the specified file pattern. Formats the sizes of files in a human-readable fashion.

Administration

Administration	
hdfs balancer -threshold 30	Runs a cluster balancing utility. Percentage of disk capacity. This overwrites the default threshold.
hadoop version	To check the version of Hadoop.
hdfs fsck /	It checks the health of the Hadoop file system.
hdfs dfsadmin -safemode leave	The command to turn off the safemode of NameNode.
hdfs dfsadmin -refreshNodes	Re-read the hosts and exclude files to update the set of Datanodes that are allowed to connect to the Namenode and those that should be decommissioned or recommissioned.
hdfs namenode -format	Formats the NameNode.

HDFS Name node UI

Hadoop

Overview

Datanodes

Datanode Volume Failures

Snapshot

Startup Progress

Utilities ▾

Overview 'hd01:8020' (active)

Started:	Thu Mar 14 11:01:37 +0700 2019
Version:	3.1.1.3.1.0.0-78, re4f82af51faec922b4804d0232a637422ec29e64
Compiled:	Thu Dec 06 20:34:00 +0700 2018 by jenkins from (HEAD detached at e4f82af)
Cluster ID:	CID-a1dea38d-b6cf-44e4-b5e8-3bde87ffad35
Block Pool ID:	BP-1412866890-10.10.137.41-1544787807355

Summary

Security is off.

Safemode is off.

165,739 files and directories, 99,858 blocks (99,858 replicated blocks, 0 erasure coded block groups) = 265,597 total filesystem object(s).

Heap Memory used 250.76 MB of 1011.25 MB Heap Memory. Max Heap Memory is 1011.25 MB.

HDFS Name node UI (2)

In operation

Show 25 entries

Search:


Node	Http Address	Last contact	Last Block Report	Capacity	Blocks	Block pool used	Version
✓hd01:50010 (10.10.137.41:50010)	http://hd01:50075	1s	251m	296.83 GB <div><div></div></div>	99858	166.56 GB (56.11%)	3.1.1.3.1.0.0-78
✓hd02:50010 (10.10.137.42:50010)	http://hd02:50075	0s	57m	296.83 GB <div><div></div></div>	99858	166.57 GB (56.12%)	3.1.1.3.1.0.0-78
✓hd03:50010 (10.10.137.43:50010)	http://hd03:50075	2s	197m	296.83 GB <div><div></div></div>	99858	166.57 GB (56.12%)	3.1.1.3.1.0.0-78

Showing 1 to 3 of 3 entries

Previous 1 Next





Hadoop Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities ▾

Browse Directory

/ Go!   

Show 25 entries

Search:

<input type="checkbox"/>	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
<input type="checkbox"/>	drwxrwxrwt	yarn	hadoop	0 B	Jun 01 17:52	0	0 B	app-logs	
<input type="checkbox"/>	drwxr-xr-x	hdfs	hdfs	0 B	Dec 18 2018	0	0 B	apps	
<input type="checkbox"/>	drwxr-xr-x	yarn	hadoop	0 B	Dec 14 2018	0	0 B	ats	
<input type="checkbox"/>	drwxr-xr-x	hdfs	hdfs	0 B	Dec 14 2018	0	0 B	atsv2	



Other HDFS interfaces

- Java API
- Thrift API
- Fuse
- WebDAV

HDFS data format

Text

Sequence file

Avro

Parquet

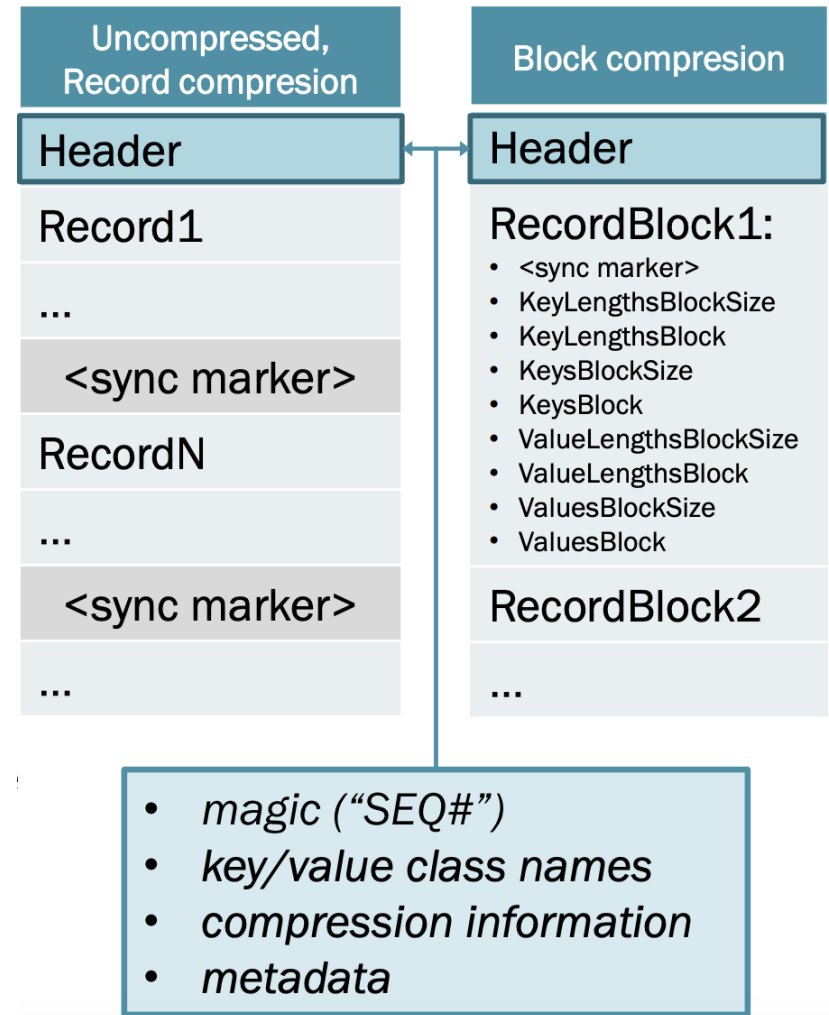
Optimized Row Columnar (ORC)

Text file

- CSV, TSV, Json records
- Convenient format to use to exchange between applications or scripts
- Human readable and parsable
- Do not support block compression
- Not as efficient to query
- Good for the beginning, but not good enough for real life.

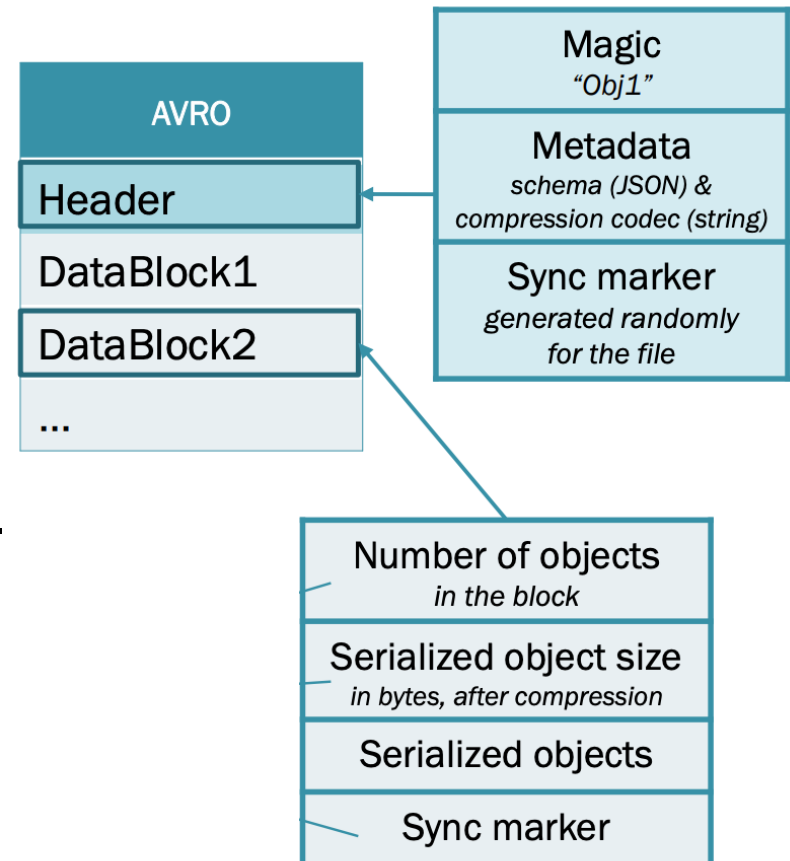
Sequence file

- Provides a persistent data structure for binary key-value pairs
- Commonly used to transfer data between Map Reduce jobs
- Can be used as an archive to pack small files in Hadoop
- Row-based
- Compression
 - Support splitting even when the data is compressed
- Splittable



Avro

- Row based
- Supports (object) compression and splitting
- Flexible data scheme
 - Schema (JSON) included to the file
- Data types
 - primitive: null, boolean, int, long, ...
 - complex: records, arrays, maps, ...
- Binary and JSON data serialization
- Data corruption detection

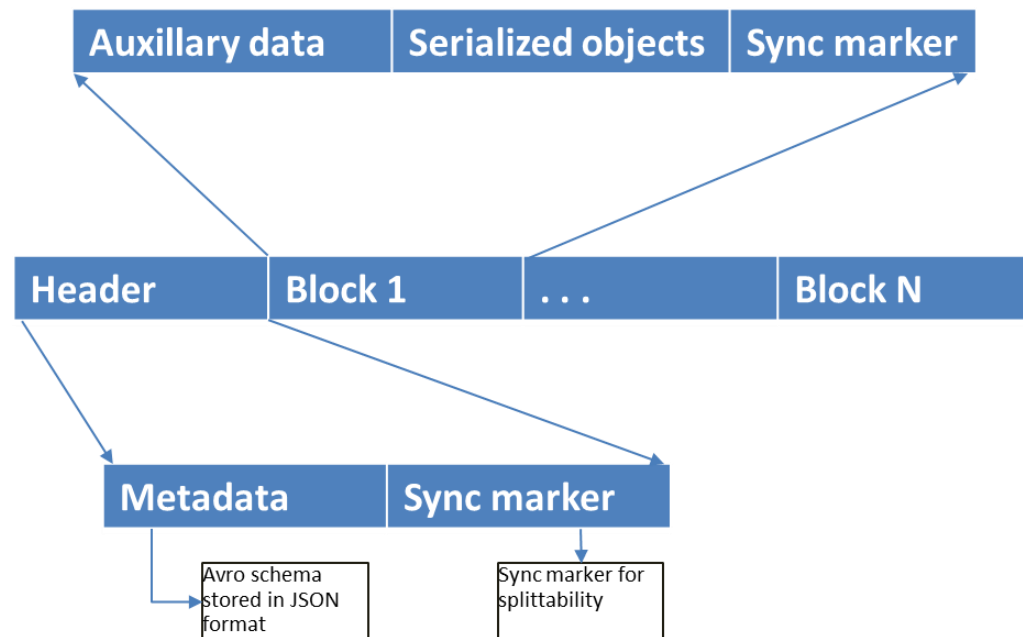


Avro – File structure and example

Sample AVRO schema in JSON format

```
{
  "type" : "record",
  "name" : "tweets",
  "fields" : [ {
    "name" : "username",
    "type" : "string",
  }, {
    "name" : "tweet",
    "type" : "string",
  }, {
    "name" : "timestamp",
    "type" : "long",
  } ],
  "doc:" : "schema for storing tweets"
}
```

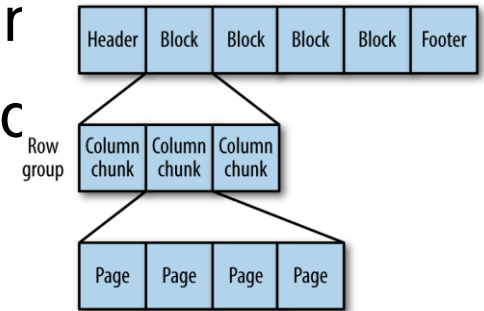
Avro file structure



Parquet

- Column-oriented binary file format
- Efficient in terms of disk I/O when specific columns need to be queried
- Supports (page) compression and splitting
- Supports nested columns (Dremel encoding)

Internal structure of parquet file



Nested schema

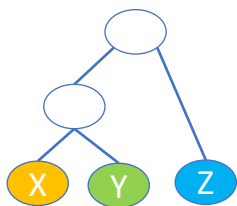


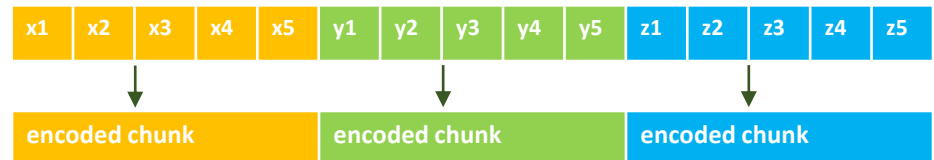
Table representation

X	Y	Z
x1	y1	z1
x2	y2	z2
x3	y3	z3
x4	y4	z4
x5	y5	z5

Row format

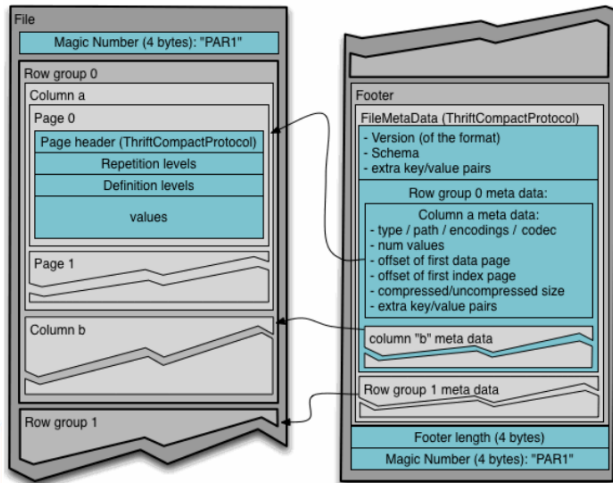
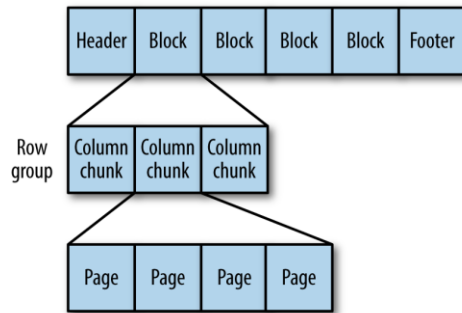


Columnar format



Parquet file structure & configuration

Internal structure of parquet file



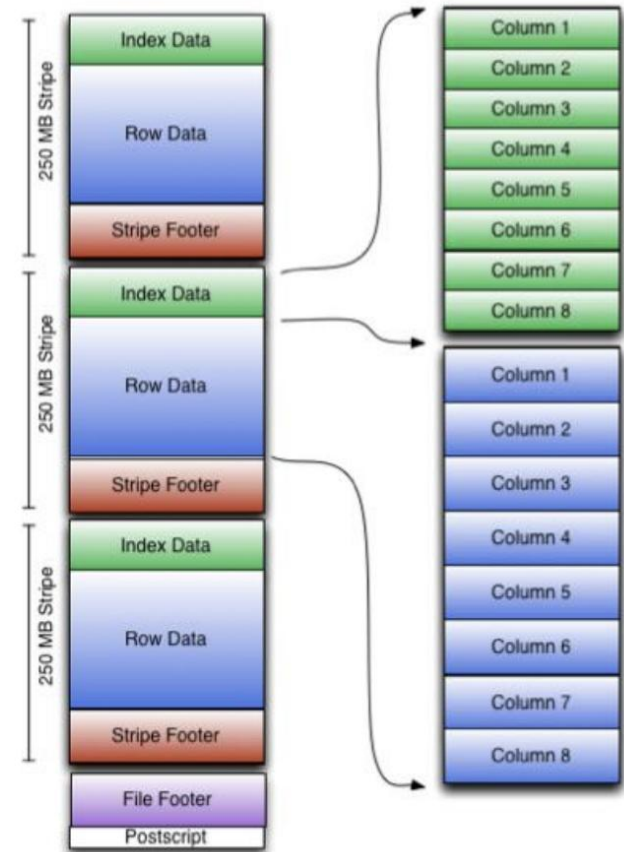
Configurable parquet parameters

Property name	Default value	Description
parquet.block.size	128 MB	The size in bytes of a block (row group).
parquet.page.size	1MB	The size in bytes of a page.
parquet.dictionary.page.size	1MB	The maximum allowed size in bytes of a dictionary before falling back to plain encoding for a page.
parquet.enable.dictionary	true	Whether to use dictionary encoding.
parquet.compression	UNCOMPRESSED	The type of compression: UNCOMPRESSED, SNAPPY, GZIP & LZO

Parquet is optimized for high compression and high scan efficiency

Optimized row columnar (ORC)

- RCFile
 - Every column is compressed individually within the row group
- ORC File
 - Block-mode compression
 - Data type support
 - Ordered data store (within one stripe)
- Stores collections of rows and within the collection the row data is stored in columnar format
- Introduces a lightweight indexing that enables skipping of irrelevant blocks of rows
- Splittable: allows parallel processing of row collections
- Indices with column-level aggregated values (min, max, sum and count)





25 YEARS ANNIVERSARY
SOICT

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Thank you
for your
attention!!!



soict.hust.edu.vn/



fb.com/groups/soict

