Big Data Storage 2: Hadoop Distributed File System (HDFS)

Shiow-yang Wu (吳秀陽)

CSIE, NDHU, Taiwan, ROC

Lecture material is mostly home-grown, partly taken with permission and courtesy from Professor Shih-Wei Liao of NTU.

What's HDFS

- HDFS is a distributed file system that is fault tolerant, scalable and extremely easy to expand.
- HDFS is the primary distributed storage for Hadoop applications.
- HDFS provides interfaces for applications to move themselves closer to data.
- HDFS is designed to 'just work', however a working knowledge helps in diagnostics and improvements.

Features of HDFS

Features of HDFS that make it ideal for distributed systems:

- Failure tolerant - data is duplicated across multiple DataNodes to protect against machine failures. The default is a replication factor of 3 (every block is stored on three machines).
- Scalability - data transfers happen directly with the DataNodes so your read/write capacity scales fairly well with the number of DataNodes
- Space - need more disk space? Just add more DataNodes and re-balance
- Industry standard - Other distributed applications are built on top of HDFS (HBase, Map-Reduce)

HDFS is designed to process large data sets with write-once-read-many semantics, it is not for low latency access

Key Features: HDFS

- Highly fault-tolerant
- High throughput
- Suitable for applications with large data sets
- Streaming access to file system data
- Can be built out of commodity hardware

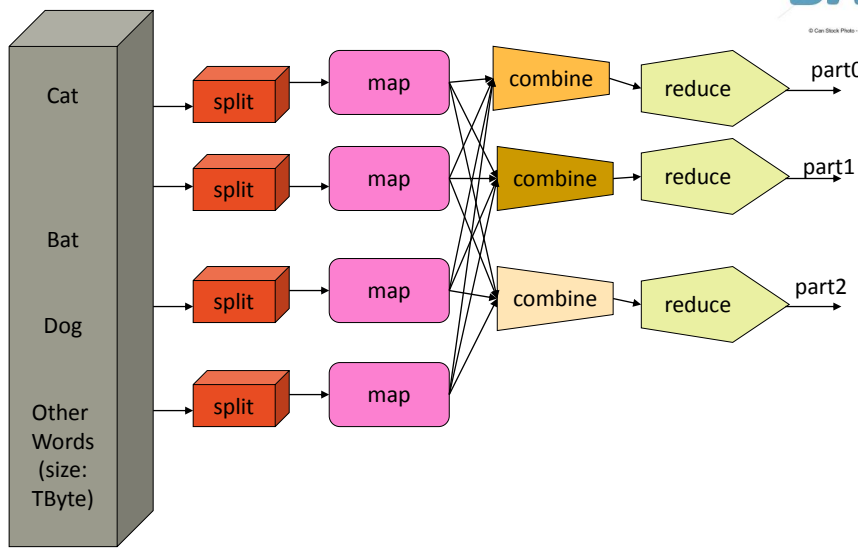
Fault Tolerance

- Failure is the norm rather than exception
- A HDFS instance may consist of thousands of server machines, each storing part of the file system's data.
- Since we have huge number of components and that each component has non-trivial probability of failure means that there is always some component that is non-functional.
- Detection of faults and quick, automatic recovery from them is a core architectural goal of HDFS.

Data Characteristics

- Streaming data access
- Applications need streaming access to data
- Batch processing rather than interactive user access.
- Large data sets and files: gigabytes to terabytes size
- High aggregate data bandwidth
- Scale to hundreds of nodes in a cluster
- Tens of millions of files in a single instance
- Write-once-read-many: a file once created, written and closed need not be changed – this assumption simplifies coherency
- A map-reduce application or web-crawler application fits perfectly with this model.

MapReduce Revisited

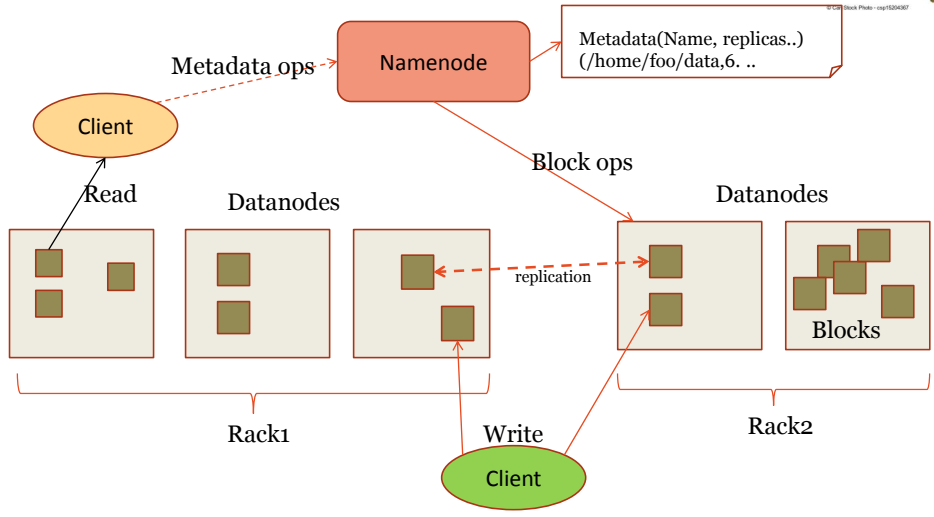


Architecture

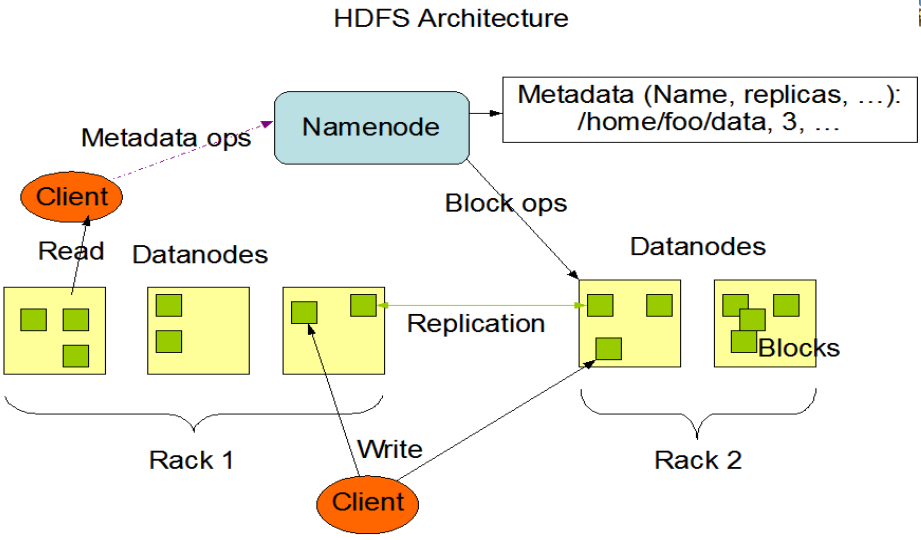
Namenode and Datanodes

- Master/slave architecture
- HDFS cluster consists of a single **Namenode**, a master server that manages the file system namespace and regulates access to files by clients.
- There are a number of **DataNodes** usually one per node in a cluster.
- The DataNodes manage storage attached to the nodes that they run on.
- HDFS exposes a file system namespace and allows user data to be stored in files.
- A file is split into one or more blocks and set of blocks are stored in DataNodes.
- DataNodes: serves read, write requests, performs block creation, deletion, and replication upon instruction from Namenode.

HDFS Architecture

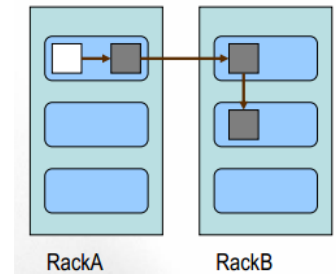


HDFS Architecture

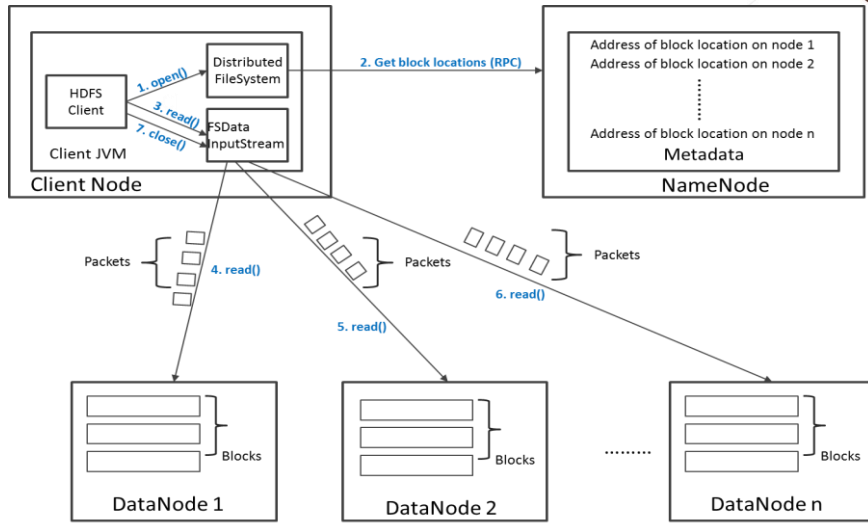


HDFS – Data Organization

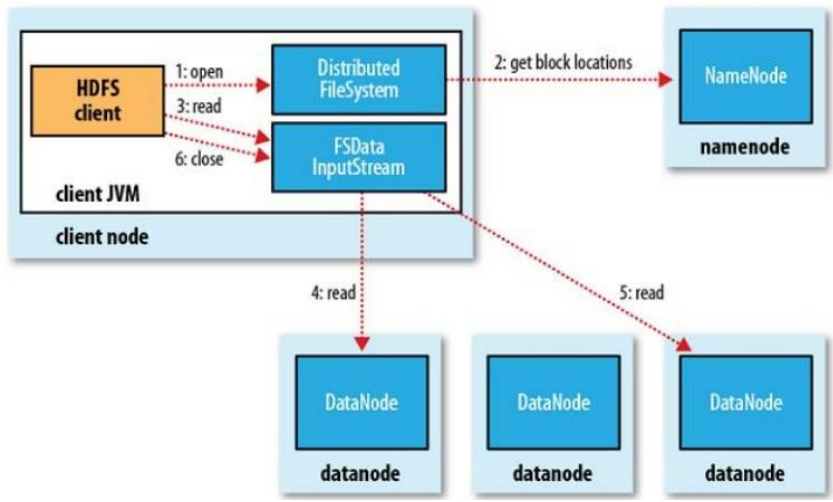
- Each file written into HDFS is split into data blocks
- Each block is stored on one or more nodes
- Each copy of the block is called replica
- Block placement policy
- First replica is placed on the local node
  - Second replica is placed in a different rack
  - Third replica is placed in the same rack as the second replica



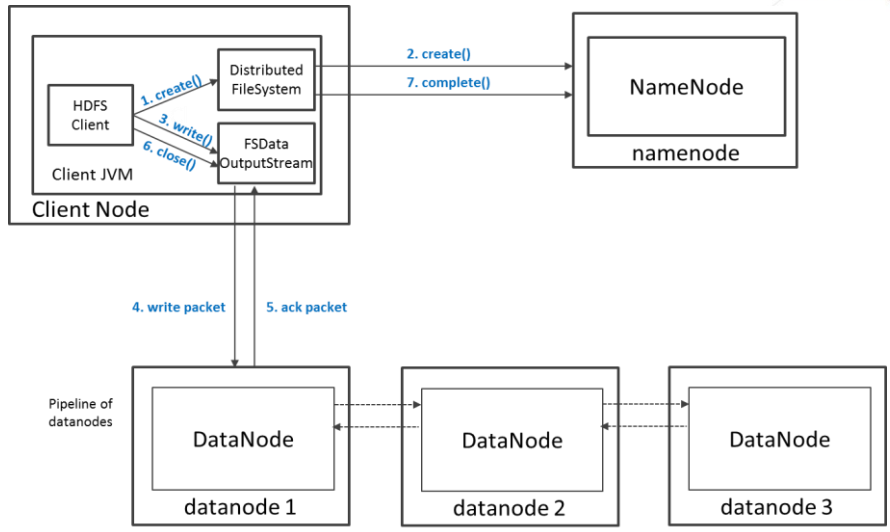
Read Operation in HDFS



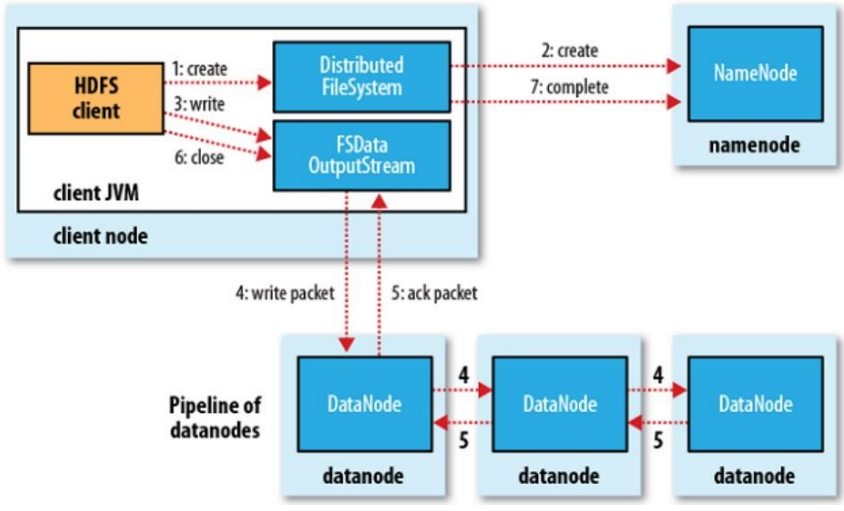
HDFS Client Read



Write Operation in HDFS



HDFS Client Write



HDFS Security

- Authentication to Hadoop
  - Simple – insecure way of using OS username to determine hadoop identity
  - Kerberos – authentication using kerberos ticket
  - Set by `hadoop.security.authentication=simple|kerberos`
- File and Directory permissions are same like in POSIX
  - read (r), write (w), and execute (x) permissions
  - also has an owner, group and mode
  - enabled by default `(dfs.permissions.enabled=true)`
- ACLs are used for implemention permissions that differ from natural hierarchy of users and groups
  - enabled by `dfs.namenode.acls.enabled=true`

File System Namespace

- Hierarchical file system with directories and files
- Create, remove, move, rename etc.
- **Namenode** maintains the file system
- Any meta information changes to the file system are recorded by the Namenode.
- An application can specify the number of replicas of the file needed: replication factor of the file. This information is stored in the Namenode.

Data Replication

- HDFS is designed to store very large files across machines in a large cluster.
- Each file is a sequence of **blocks**.
- All blocks in the file except the last are of the same size.
- Blocks are replicated for fault tolerance.
- Block size and replicas are configurable per file.
- The Namenode receives a Heartbeat and a BlockReport from each DataNode in the cluster.
- BlockReport contains all the blocks on a Datanode.

Replica Placement

- The placement of the replicas is critical to HDFS reliability and performance.
- Optimizing replica placement distinguishes HDFS from other distributed file systems.
- Rack-aware replica placement:
  - Goal: improve reliability, availability and network bandwidth utilization
  - Research topic
- Many racks, communication between racks are through switches.
- Network bandwidth between machines on the same rack is greater than those in different racks.

Replica Placement

- Namenode determines the rack id for each DataNode.
- Replicas are typically placed on unique racks
  - Simple but non-optimal
  - Writes are expensive
  - Replication factor is 3
  - Another research topic?
- Replicas are placed: one on a node in a local rack, one on a different node in the local rack and one on a node in a different rack.
- 1/3 of the replica on a node, 2/3 on a rack and 1/3 distributed evenly across remaining racks.

Replica Selection

- Replica selection for READ operation: HDFS tries to minimize the bandwidth consumption and latency.
- If there is a replica on the Reader node then that is preferred.
- HDFS cluster may span multiple data centers: replica in the local data center is preferred over the remote one.

Safemode Startup

- On startup Namenode enters **Safemode**.
- Replication of data blocks do not occur in Safemode.
- Each DataNode checks in with Heartbeat and BlockReport.
- Namenode verifies that each block has acceptable number of replicas
- After a configurable percentage of safely replicated blocks check in with the Namenode, Namenode exits Safemode.
- It then makes the list of blocks that need to be replicated.
- Namenode then proceeds to replicate these blocks to other Datanodes.

Filesystem Metadata

- The HDFS namespace is stored by Namenode.
- Namenode uses a transaction log called the EditLog to record every change that occurs to the filesystem meta data.
  - For example, creating a new file.
  - Change replication factor of a file
  - EditLog is stored in the Namenode's local filesystem
- Entire filesystem namespace including mapping of blocks to files and file system properties is stored in a file FsImage. Stored in Namenode's local filesystem.

Namenode

- Keeps image of entire file system namespace and file Blockmap in memory.
- 4GB of local RAM is sufficient to support the above data structures that represent the huge number of files and directories.
- When the Namenode starts up it gets the FsImage and Editlog from its local file system, update FsImage with EditLog information and then stores a copy of the FsImage on the filesytstem as a checkpoint.
- Periodic checkpointing is done. So that the system can recover back to the last checkpointed state in case of a crash.

Datanode

- A Datanode stores data in files in its local file system.
- Datanode has no knowledge about HDFS filesystem
- It stores each block of HDFS data in a separate file.
- Datanode does not create all files in the same directory.
- It uses heuristics to determine optimal number of files per directory and creates directories appropriately:
  - Research issue?
- When the filesystem starts up it generates a list of all HDFS blocks and send this report to Namenode: Blockreport.

Protocol

The Communication Protocol

- All HDFS communication protocols are layered on top of the TCP/IP protocol
- A client establishes a connection to a configurable TCP port on the Namenode machine. It talks ClientProtocol with the Namenode.
- The Datanodes talk to the Namenode using Datanode protocol.
- RPC abstraction wraps both ClientProtocol and Datanode protocol.
- Namenode is simply a server and never initiates a request; it only responds to RPC requests issued by DataNodes or clients.

Robustness

Objectives

- Primary objective of HDFS is to store data reliably in the presence of failures.
- Three common failures are:
  - Namenode failure
  - Datanode failure
  - network partition

DataNode Failure and Heartbeat

- A network partition can cause a subset of Datanodes to lose connectivity with the Namenode.
- Namenode detects this condition by the absence of a Heartbeat message.
- Namenode marks Datanodes without Hearbeat and does not send any IO requests to them.
- Any data registered to the failed Datanode is not available to the HDFS.
- Also the death of a Datanode may cause replication factor of some of the blocks to fall below their specified value.

Re-replication

- The necessity for re-replication may arise due to:
  - A Datanode may become unavailable,
  - A replica may become corrupted,
  - A hard disk on a Datanode may fail, or
  - The replication factor on the block may be increased.

Cluster Rebalancing

- HDFS architecture is compatible with data rebalancing schemes.
- A scheme might move data from one Datanode to another if the free space on a Datanode falls below a certain threshold.
- In the event of a sudden high demand for a particular file, a scheme might dynamically create additional replicas and rebalance other data in the cluster.
- These types of data rebalancing are not yet implemented: research issue.

Data Integrity

- Consider a situation: a block of data fetched from Datanode arrives corrupted.
- This corruption may occur because of faults in a storage device, network faults, or buggy software.
- A HDFS client creates the checksum of every block of its file and stores it in hidden files in the HDFS namespace.
- When a clients retrieves the contents of file, it verifies that the corresponding checksums match.
- If does not match, the client can retrieve the block from a replica.

Metadata Disk Failure

- FsImage and EditLog are central data structures of HDFS.
- A corruption of these files can cause a HDFS instance to be non-functional.
- For this reason, a Namenode can be configured to maintain multiple copies of the FsImage and EditLog.
- Multiple copies of the FsImage and EditLog files are updated synchronously.
- Meta-data is not data-intensive.
- The Namenode could be single point failure: automatic failover is NOT supported! Another research topic.

Data Organization

Data Blocks

- HDFS support write-once-read-many with reads at streaming speeds.
- A typical block size is 64MB (or even 128 MB).
- A file is chopped into 64MB chunks and stored.

Staging

- A client request to create a file does not reach Namenode immediately.
- HDFS client caches the data into a temporary file. When the data reached a HDFS block size the client contacts the Namenode.
- Namenode inserts the filename into its hierarchy and allocates a data block for it.
- The Namenode responds to the client with the identity of the Datanode and the destination of the replicas (Datanodes) for the block.
- Then the client flushes it from its local memory.

Staging (contd.)

- The client sends a message that the file is closed.
- Namenode proceeds to commit the file for creation operation into the persistent store.
- If the Namenode dies before file is closed, the file is lost.
- This client side caching is required to avoid network congestion; also it has precedence in AFS (Andrew file system).

Replication Pipelining

- When the client receives response from Namenode, it flushes its block in small pieces (4K) to the first replica, that in turn copies it to the next replica and so on.
- Thus data is pipelined from Datanode to the next.

API (Accessibility)

Application Programming Interface

- HDFS provides **Java API** for application to use.
- **Python** access is also used in many applications.
- A C language wrapper for Java API is also available.
- A HTTP browser can be used to browse the files of a HDFS instance.

FS Shell, Admin and Browser Interface

- HDFS organizes its data in files and directories.
- It provides a command line interface called the FS shell that lets the user interact with data in the HDFS.
- The syntax of the commands is similar to bash and csh.
- Example: to create a directory /foodir

```
hdfs dfs –mkdir /foodir
```

- There is also DFSAdmin interface available
- Browser interface is also available to view the namespace.

Space Reclamation

- When a file is deleted by a client, HDFS renames file to a file in the /trash directory for a configurable amount of time.
- A client can request for an undelete in this allowed time.
- After the specified time the file is deleted and the space is reclaimed.
- When the replication factor is reduced, the Namenode selects excess replicas that can be deleted.
- Next heartbeat transfers this information to the Datanode that clears the blocks for use.

Interfaces to HDFS

- Java API (`DistributedFileSystem`)
- C wrapper (`libhdfs`)
- HTTP protocol
- WebDAV protocol
- Shell Commands

However the command line is one of the simplest and most familiar

HDFS – Shell Commands

There are two types of shell commands

User Commands

- `hdfs dfs` – runs filesystem commands on the HDFS
- `hdfs fsck` – runs a HDFS filesystem checking command

Administration Commands

- `hdfs dfsadmin` – runs HDFS administration commands

HDFS – User Commands (dfs)

List directory contents

```
hdfs dfs -ls
hdfs dfs -ls /
hdfs dfs -ls -R /var
```

Display the disk space used by files

```
hdfs dfs -du -h /
hdfs dfs -du /hbase/data/hbase/namespace/
hdfs dfs -du -h /hbase/data/hbase/namespace/
hdfs dfs -du -s /hbase/data/hbase/namespace/
```


HDFS – User Commands (dfs)

Copy data to HDFS

```
hdfs dfs -mkdir tdata
hdfs dfs -ls
hdfs dfs -copyFromLocal tutorials/data/geneva.csv tdata
hdfs dfs -ls -R
```

Copy the file back to local filesystem

```
cd tutorials/data/
hdfs dfs -copyToLocal tdata/geneva.csv geneva.csv.hdfs
md5sum geneva.csv geneva.csv.hdfs
```

HDFS – User Commands (acls)

List acl for a file

```
hdfs dfs -getfacl tdata/geneva.csv
```

List the file statistics – (%r – replication factor)

```
hdfs dfs -stat "%r" tdata/geneva.csv
```

Write to hdfs reading from stdin

```
echo "blah blah blah" | hdfs dfs -put - tdataset/tfile.txt
hdfs dfs -ls -R
hdfs dfs -cat tdataset/tfile.txt
```


HDFS – User Commands (fsck)

Removing a file

```
hdfs dfs -rm tdataset/tfile.txt
hdfs dfs -ls -R
```

List the blocks of a file and their locations

```
hdfs fsck /user/cloudera/tdata/geneva.csv -
files -blocks -locations
```

Print missing blocks and the files they belong to

```
hdfs fsck / -list-corruptfileblocks
```

HDFS – Adminstration Commands

Comprehensive status report of HDFS cluster

```
hdfs dfsadmin -report
```

Prints a tree of racks and their nodes

```
hdfs dfsadmin -printTopology
```

Get the information for a given datanode (like ping)

```
hdfs dfsadmin -getDatanodeInfo
localhost:50020
```


HDFS – Advanced Commands

Get a list of namenodes in the Hadoop cluster

```
hdfs getconf -namenodes
```

Dump the NameNode fsimage to XML file

```
cd /var/lib/hadoop-hdfs/cache/hdfs/dfs/name/current
hdfs oiv -i fsimage_0000000000000003388 -o
/tmp/fsimage.xml -p XML
```

The general command line syntax is

hdfs command [genericOptions] [commandOptions]

Other Interfaces to HDFS

HTTP Interface

```
http://quickstart.cloudera:50070
```

MountableHDFS – FUSE

```
mkdir /home/cloudera/hdfs
sudo hadoop-fuse-dfs dfs://quickstart.cloudera:8020
/home/cloudera/hdfs
```

Once mounted all operations on HDFS can be performed using standard Unix utilities such as 'ls', 'cd', 'cp', 'mkdir', 'find', 'grep',

Summary

- We discussed the features of the Hadoop Distributed File System, a peta-scale file system to handle big-data sets.
- What discussed: Architecture, Protocol, API, etc.
- Missing element: Implementation
  - The Hadoop file system (internals)
  - An implementation of an instance of the HDFS (for use by applications such as web crawlers).