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

Shiow-yang Wu (吳秀陽)

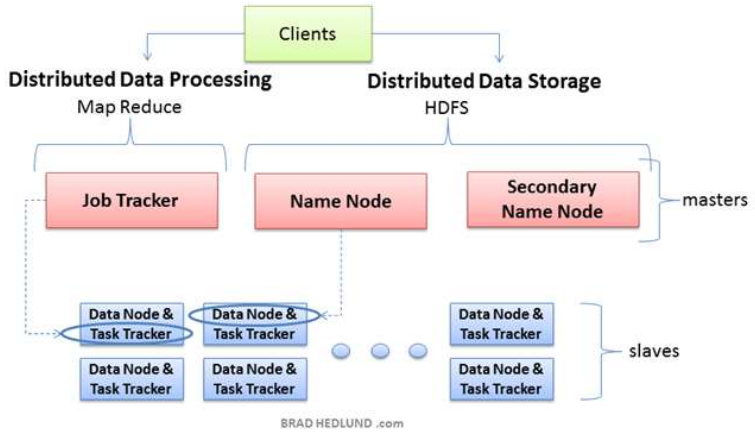
CSIE, NDHU, Taiwan, ROC

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

The HDFS Paper

- Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. 2010. The Hadoop Distributed File System. *In Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST) (MSST '10). IEEE Computer Society*

Hadoop Server Roles



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 was originally designed to 'just work'.
- Over the years, it has been improved into a solid big data storage system for Hadoop ecosystem.

Assumptions and Goals 1

- **Hardware Failure**
  - Hardware failure is the norm. Some component of HDFS is always non-functional.
  - Auto detection and quick recovery is a core goal.
- **Streaming Data Access**
  - Applications need streaming access to data.
  - Designed more for batch than interactive processing.
  - Emphasis is on high throughput rather than low latency
  - POSIX standard not entirely needed. Some areas can be traded to increase data throughput.

Assumptions and Goals 2

- **Large Data Sets**
  - Typical file is gigabytes to terabytes. HDFS is tuned to support large files.
  - Provide high aggregate data bandwidth
  - Scale to hundreds of nodes in a single cluster.
  - Support tens of millions of files in a single instance.
- **Simple Coherency Model**
  - Write-once-read-many access model, no change except for appends and truncates. (not arbitrary point)
  - Simplifies data coherency issues and enables high throughput.

Assumptions and Goals 3

- **Moving Computation is Cheaper than Moving Data**
  - This minimizes network congestion and increases the overall throughput.
  - Provides interfaces for applications to move themselves closer to where the data is located.
- **Portability Across Heterogeneous Hardware and Software Platforms**
  - Designed to be easily portable.
  - Facilitates widespread adoption of HDFS

Features of HDFS

Features of HDFS that make it good 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 read/write capacity scales fairly well with the number of DataNodes
- Space - need more 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
- Auto and quick recovery
- High throughput
- Suitable for applications with large data sets
- Streaming access to file system data
- Batch processing > interactive processing
- Can be built out of commodity hardware
- Support PB level storage space

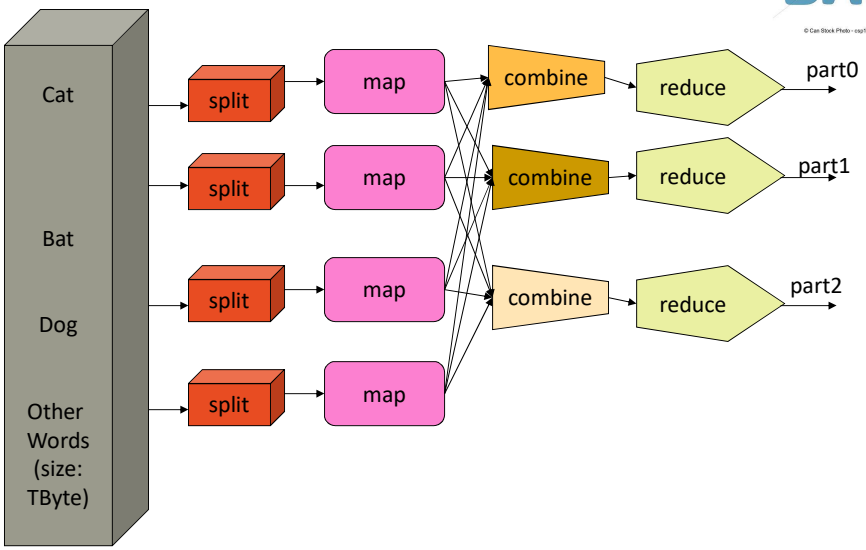
Fault Tolerance

- Failure is the norm rather than exception
- A HDFS instance may consist of thousands of server machines, each storing part of the data.
- Huge number of components and each component has non-trivial probability of failure.
- 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

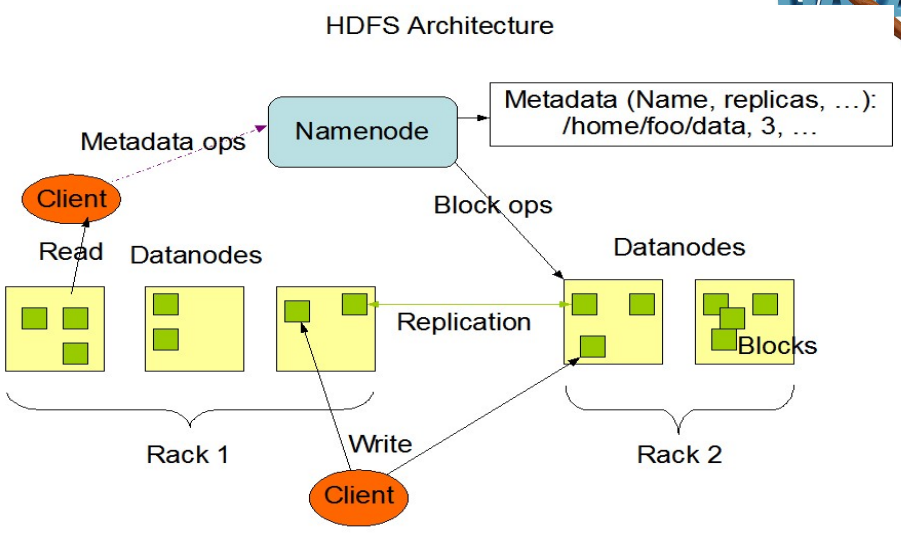
- Streaming data access
- Applications need streaming access to data
- Batch processing rather than interactive user access.
- Large data sets and files: gigabytes to terabytes size
- High aggregate data bandwidth
- Scale to hundreds of nodes in a cluster
- Tens of millions of files in a single instance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

MapReduce Revisited



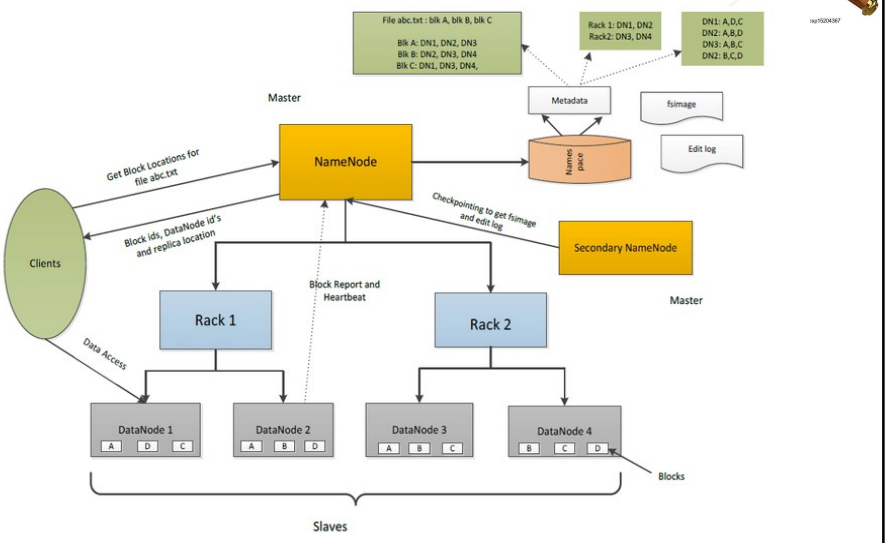
HDFS Architecture

HDFS Architecture



(https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html)

HDFS Architecture



HDFS Namespace

- Hierarchical organization of files and directories.
- In RAM
- NameNode maintains the namespace.
- Attributes: permissions, modification and access times, namespace and disk space quotas
- No hard links or soft links.
- An application can specify the number of replicas (replication factor) of a file.

Blocks

- HDFS blocks are either 64MB or 128MB
- Block size and replication factor are configurable per file.
- Large blocks minimize the cost of seeks
- Can benefit from any disks in the cluster
- Simplify the storage subsystem by reducing the metadata per file and fit well with replication.
- NameNode periodically receives Heartbeat and Blockreport messages from DataNodes.
- Heartbeat implies the DataNode is OK.
- Blockreport contains a list of all blocks on a DataNode.

NameNode & DataNodes

- Master/slave architecture
- The single **NameNode** is a master server that manages the file system namespace and regulates access to files.
- Many **DataNodes** (usually one per node in a cluster) that 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 which are stored in a set of DataNodes.
- **DataNodes**: serves open, close, rename, read, write requests, performs block creation, deletion, and replication upon instruction from NameNode.

NameNode

- A highly available server that manages the File System Namespace and controls access to files.
- The master that manages DataNodes(slave nodes)
- Maintains file system tree and metadata-persistently on two files: FsImage and EditLogs (more on this later)
- Stores locations of blocks-but not persistently
- **Metadata**: Namespace, DataNodes and replication information, list of blocks of each file
- User data never flows through the NameNode.

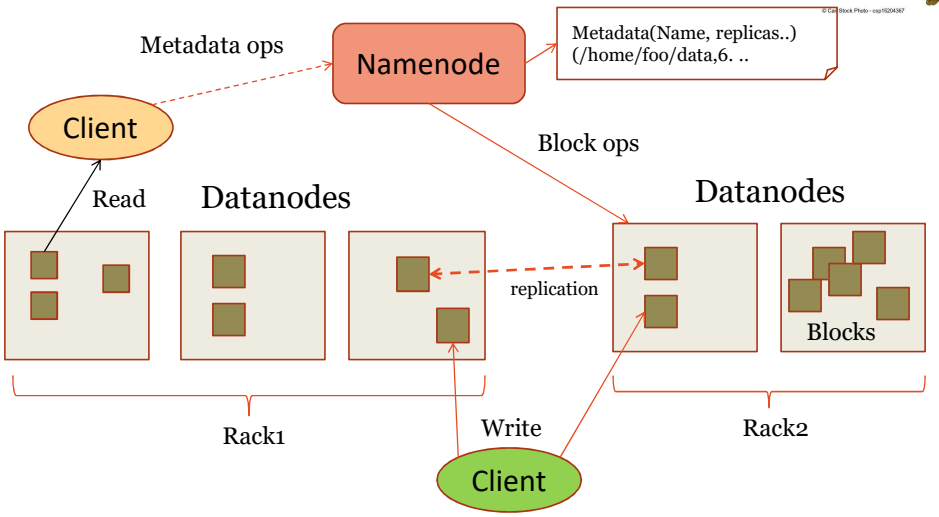
NameNode

- Two files associated with the metadata:
  - **FsImage:** contains the complete state of the file system namespace since the start of the NameNode.
  - **EditLogs:** contains all the recent modifications made to the file system w.r.t. the most recent FsImage.
- Records each change that takes place to the file system metadata.
- Regularly receives a Heartbeat and a block report from all the DataNodes.
- Keeps a record of all the blocks in HDFS and in which nodes these blocks are located.
- Takes care of the replication factor of all the blocks.
- In case of the DataNode failure, chooses new DataNodes for new replicas.

DataNodes

- Slave daemons or process which runs on each slave machine
- Workhorses of the file system
- Store(write) and retrieve(read) blocks for HDFS clients (actual data is stored on DataNodes)
- Send block reports and Hearbeat to NameNode, (every 3 seconds by default).
- Do not use data protection mechanisms like RAID…use replication

NameNode & DataNodes

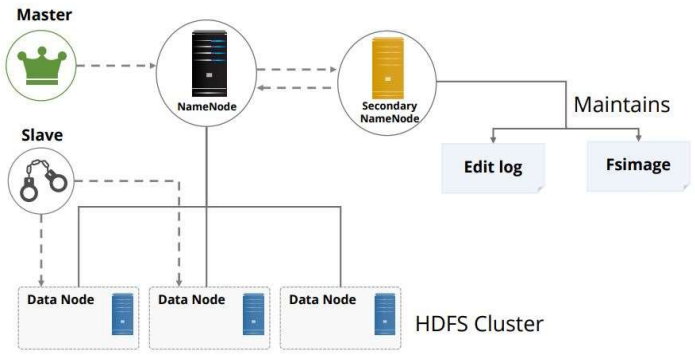


Secondary NameNode

- Works concurrently with the primary NameNode as a helper daemon.
- Not a **backup NameNode!!**
- Constantly reads all the file systems and metadata from the RAM of the NameNode and writes it into the hard disk or the file system
- Responsible for combining the EditLogs with FsImage from the NameNode
- Downloads the EditLogs from the NameNode at regular intervals and applies to FsImage
- New FsImage is copied back to the NameNode, which is used whenever the NameNode is started the next time

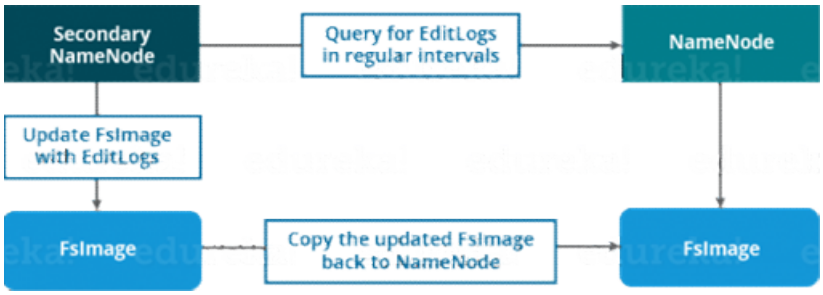
Secondary NameNode

- Secondary NameNode server maintains the edit log and namespace image information in sync with the NameNode server.



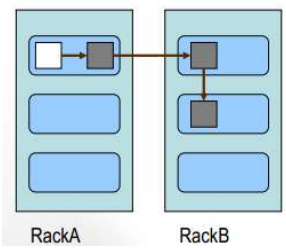
Secondary NameNode

- Secondary NameNode performs regular checkpoints in HDFS. Therefore, it is also called **CheckpointNode**.

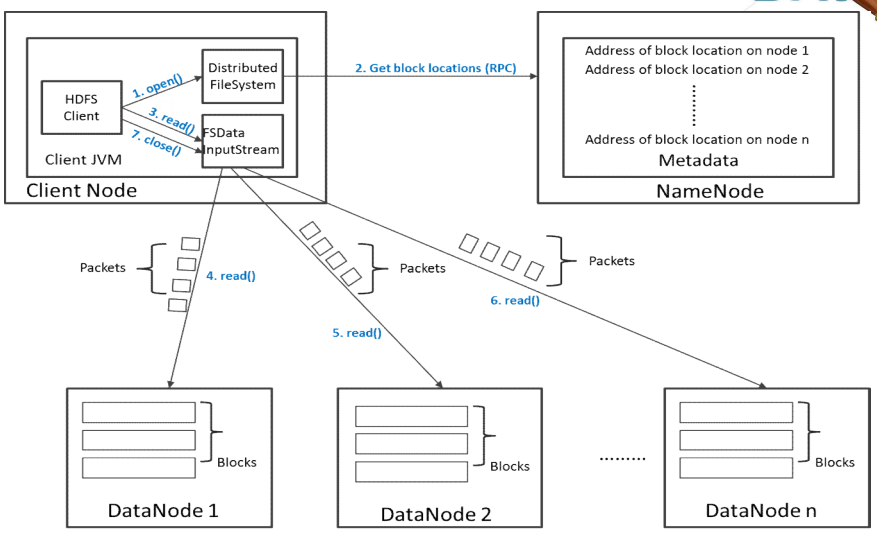


HDFS – Data Organization

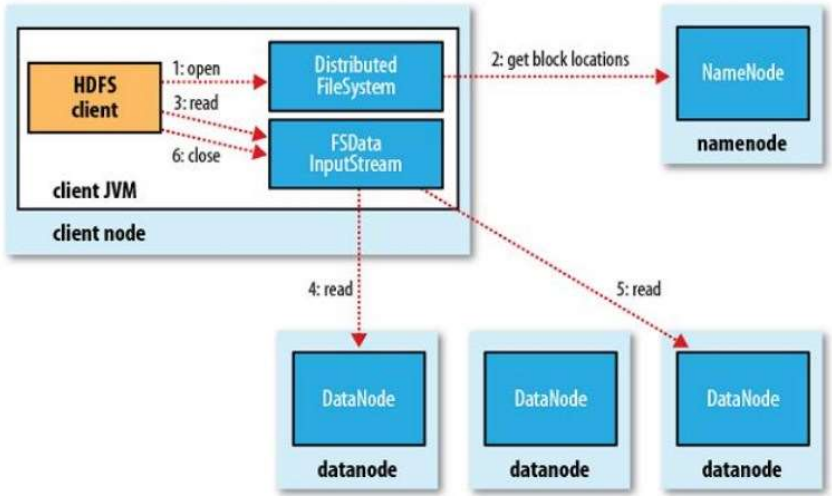
- Each file written into HDFS is split into data blocks
- Each block is stored on one or more nodes
- Each copy of the block is called a replica
- Block placement policy:
  - First replica is placed on the **local node**
  - Second replica is placed in a **different rack**
  - Third replica is placed in the same rack as the second replica
  - 4th and following replicas are placed randomly while keeping #replicas per rack below an upper limit



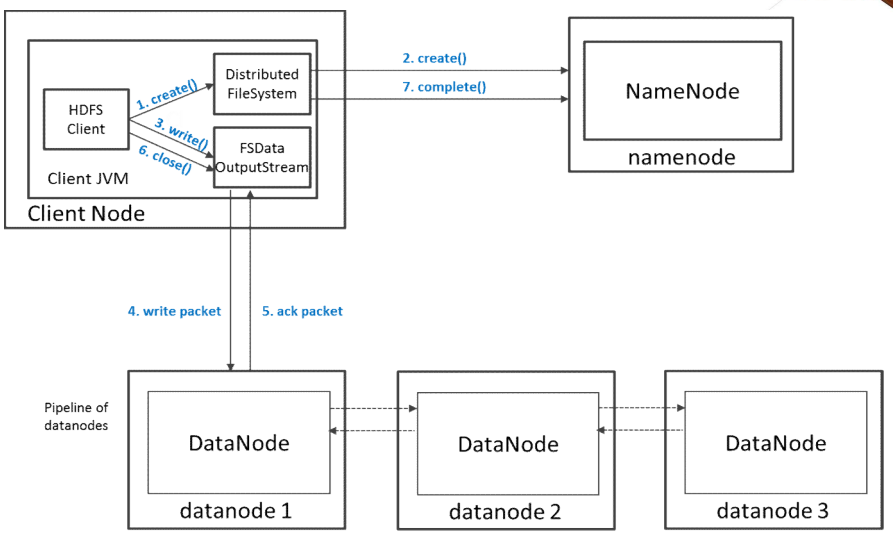
Read Operation in HDFS



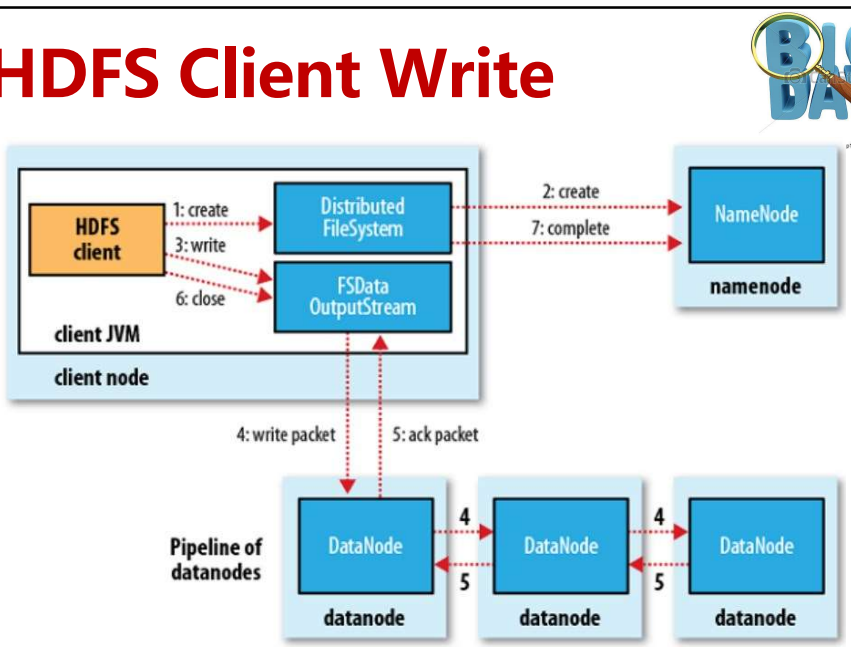
HDFS Client Read



Write Operation in HDFS



HDFS Client Write



HDFS Security

- Authentication to Hadoop
  - Simple – insecure way of using OS username to determine hadoop identity
  - Kerberos – a third party authentication mechanism using kerberos ticket
  - Set by `hadoop.security.authentication=simple|kerberos`
- File/directory permissions are same as 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 implementing 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 a 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 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 the local rack, one on a different rack and one on a different node of the same rack as the second.
- Remaining blocks are distributed evenly across remaining racks.

Replica Selection

- Replica selection for READ operation: HDFS tries to minimize the bandwidth consumption and latency.
- HDFS tries to satisfy a read request from a replica that is closest to the reader.
- 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, the NameNode exits Safemode.
- It then makes the list of blocks that still 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 Operations

- 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 Operations

- 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.

The Communication Protocol

- All HDFS communication protocols are layered on top of 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 failures
  - DataNode failures
  - Network partitions

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 forward any new IO requests to them.
- Any data registered to the failed DataNode is not available to the HDFS.
- The death of a DataNode may cause replication factor of some blocks to fall below their specified value.

Re-Replication

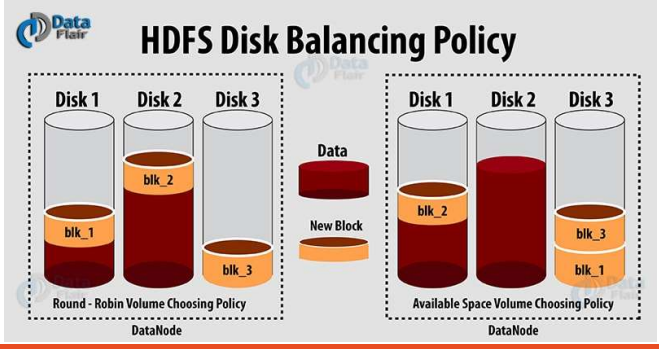
- The NameNode constantly tracks which blocks need to be replicated and initiates replication whenever necessary.
- The necessity for re-replication may arise due to:
  - A DataNode may become unavailable,
  - A replica may become corrupted,
  - A hard disk on a DataNode may fail, or
  - The replication 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.
- On a sudden high demand for a particular file, a scheme might dynamically create additional replicas and rebalance other data in the cluster.
- Data rebalancing is done by the balancer utility. (not implemented yet)

HDFS Disk Balancing

- When writing new blocks, DN chooses the target disk based on **volume-choosing policies**:
  - Round-Robin policy: spread new blocks across disks
  - Available space policy: writes data to disks with more free space (by percentage)



HDFS Disk Balancer

- Round-robin leads to unbalance in the long run.
- Available-space creates bottleneck on new disks.
- A tool named Disk Balancer in Hadoop 3 help us in balancing data on disks.
- A command line tool that distributes data evenly on all disks of a DataNode.
- It is mainly for Intro-DataNode balancing, i.e. **Disk Balancer** distributes data within the DN.
- Solve the new disk bottleneck for available-space policy.

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 not, the client can retrieve the block from another DataNode with a replica.

Metadata Disk Failure

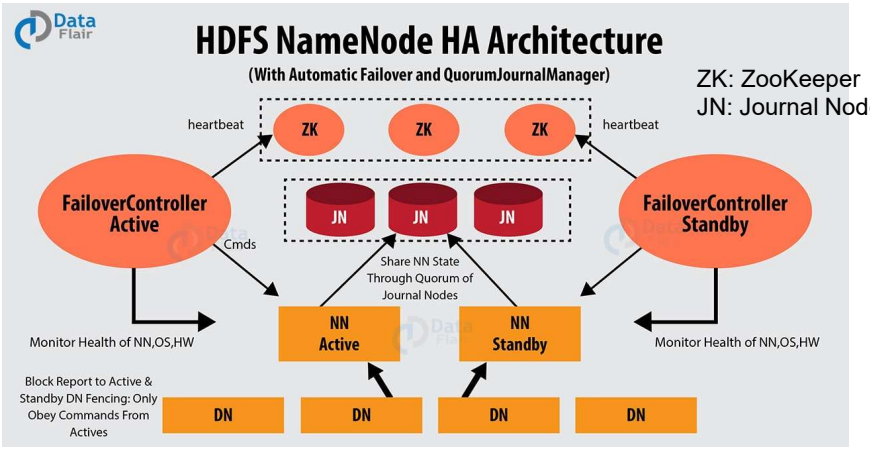
- FsImage and EditLog are central data structures of HDFS.
- A corruption of these files can cause a HDFS instance to be non-functional.
- For this reason, a Namenode can be configured to maintain multiple copies of the FsImage and EditLog.
- Multiple copies of the FsImage and EditLog files are updated synchronously.
- Degradation acceptable since HDFS applications are not metadata intensive.
- The NameNode could be a single point failure: automatic failover is not supported in Hadoop 1.
- Since Hadoop 2, can have redundant NameNodes for quick failover. (next slide)

Hadoop NameNode High Availability Architecture

- Hadoop 2.0 overcomes the **SPOF** of NameNode by providing support for many NameNodes.
- HDFS NameNode High Availability(HA) architecture provides the option of running two redundant NameNodes in the same cluster in an active/passive configuration with a hot standby.
  - **Active NameNode** – Handles all client operations in the cluster.
  - **Passive(Standby) NameNode** – A standby namenode, which has similar data as active NN. It acts as a slave, maintains enough state to provide a fast failover, if necessary.

NameNode HA Arch

- If Active NN fails, passive NN takes all the responsibility and cluster continues to work.



JournalNodes

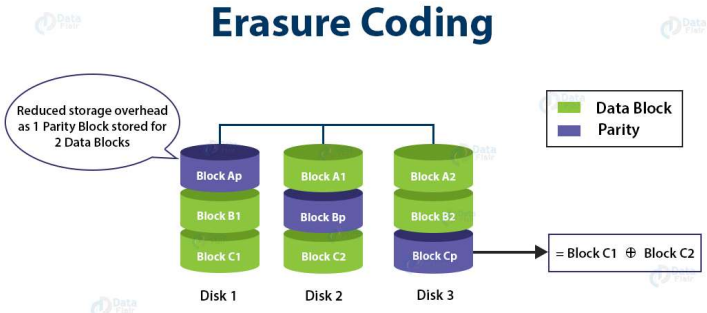
- HA use JournalNodes (JNs) to synchronize active and standby NameNodes.
- Active NN writes to each JN with changes (“edits”) to HDFS namespace metadata.
- During failover, the standby NN applies all edits from the JNs before promotion to active.
- Must be at least 3 JNs with the Quorum Journal Manager(QJM) as shared storage.

NameNode HA Arch

- Active and Standby NameNode are always in sync with each other, i.e. they have the same metadata.
- This permit to reinstate the Hadoop cluster to the same namespace state where it got crashed. And this will provide us to have fast failover.
- Must have only one NameNode active at a time.
- Fencing avoids such scenarios by ensuring that only one NameNode remains active at a particular time.

HDFS Erasure Coding

- As we mentioned earlier in new features of Hadoop 3, Erasure Coding reduces the storage overhead from 200% to 50%.
- This is primary done in HDFS.



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

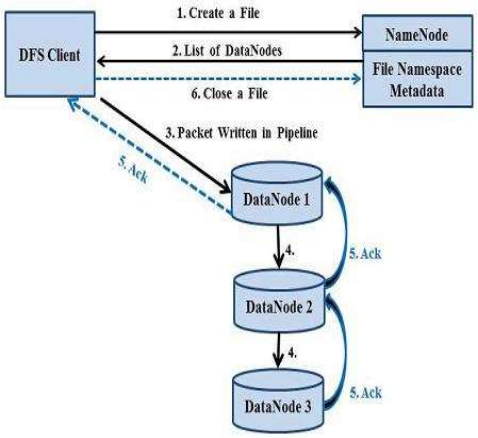
- A client request to create a file does not reach NameNode immediately.
- HDFS client caches the data into a temporary file. When the data reached a HDFS block size the client contacts the NameNode.
- NameNode inserts the filename into its hierarchy and allocates a data block for it.
- The NameNode 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 client receives response from NN, it flushes its block in small pieces (4K) to the first replica, that in turn copies it to the next replica and so on.
- Data is pipelined from DataNode to the next.



HDFS Federation

HDFS Federation

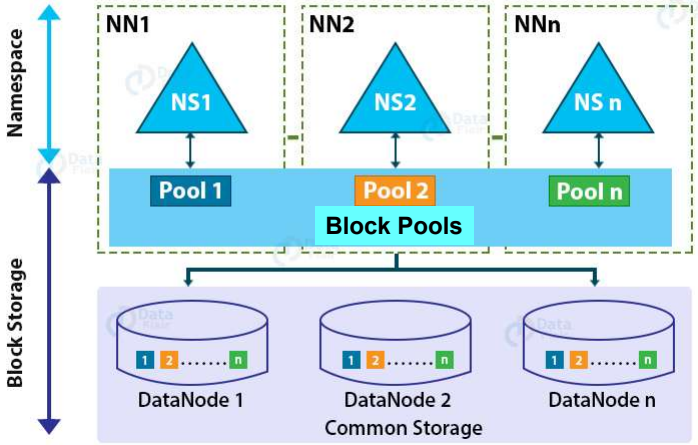
- The HDFS Federation architecture allow multiple NameNodes (NNs) and Namespaces.
- Each NN has its own **namespace** and **block pool** (independent of the other and no sync required)
- DataNodes are used as common storage for blocks by all the NameNodes.
- Each DN gets registered to ALL NNs for all block pools.
- DN periodically send heartbeats to ALL NNs.
- (next slide for arch.)

Benefits of HDFS Federation

- Namespace scalability
  - Can horizontally scale the namespace.
  - Good for large clusters.
- Performance
  - Improve performance since file system operations are not limited by the throughput of a single NN.
- Isolation
  - Multiple namespaces provide isolation.
- Future innovations

HDFS Federation Arch

HDFS Federation Architecture



API (Accessibility)

Application Programming Interface

- HDFS provides **Java API** for application to use.
- A **C language wrapper** for Java API and **REST API** is also available.
- **Python** access is also used in many applications.
- An HTTP browser can be used to browse the files of a HDFS instance.
- With NFS gateway, HDFS can be mounted as part of the client's local file system.

FS Shell, Admin and Browser Interface

- HDFS organizes its data into 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 a DFSAdmin interface available for HDFS administrator.
- Browser interface is also available to view the namespace.

Space Reclamation

- When a file is deleted, 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 which clears the blocks for use.

Interfaces to HDFS

- Java API (`DistributedFileSystem`)
- C wrapper (`libhdfs`)
- Python **`hdfs`** module (and many others)
- HTTP protocol
- WebDAV protocol
- WebHDFS REST API
- . . .
- Shell Commands

The Shell command line is still the simplest and most familiar one.

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

(https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HDFSCommands.html)

HDFS – User Commands (dfs)

List directory contents

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

Recursive display

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/
```

"human-readable"

Aggregate summary

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
```

Mountable HDFS–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(HDFS), a peta-scale file system to handle big data sets.
- What discussed: Architecture, data organization, protocols, APIs, basic implementation, etc.
- Missing elements: Advanced implementation details
- Look for “HDFS Internals”

(https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html)