Big Data
- HDFS and I/O

Dr. Matt Zhang
ITU

Outline

- HDFS: Hadoop Distributed File System
- Hadoop Security: Hadoop Security model
- Hadoop IO: Shuffling Optimization in Hadoop MR

HDFS: Hadoop Distributed File System Overview

HDFS: Overview

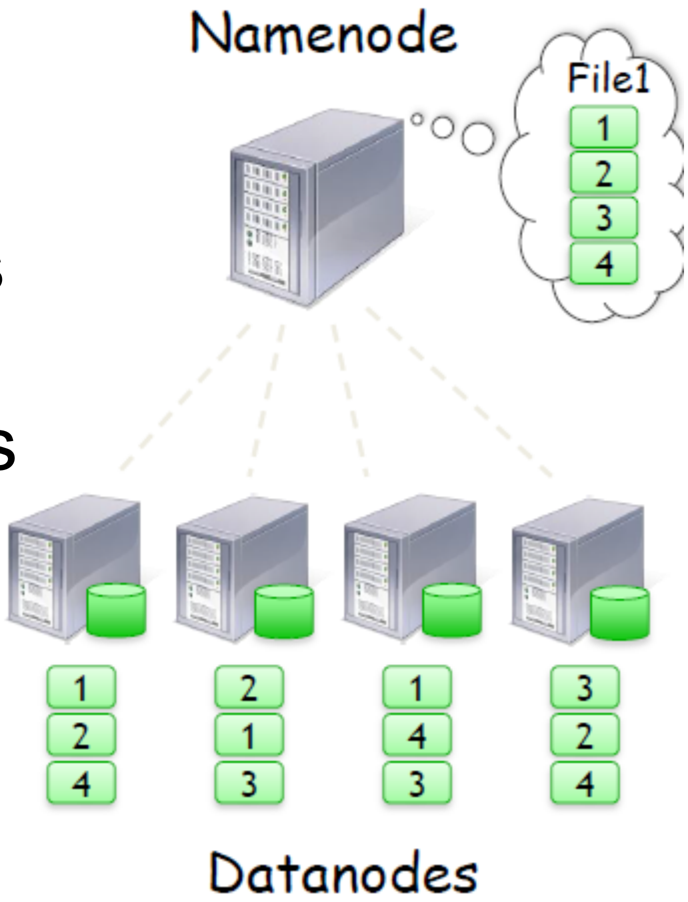
- Based on Google GFS
- Provides redundant storage for massive amounts of data
- Data is distributed to all nodes in the cluster
- Optimized for MR processing

HDFS: Design Assumptions

- High component failure rates
  - Nodes are generally commodity machines
- Modest number of huge files
  - A few million
  - Typically multi-gigabyte files
- Write-once read-many
- Large streaming reads
  - No random access

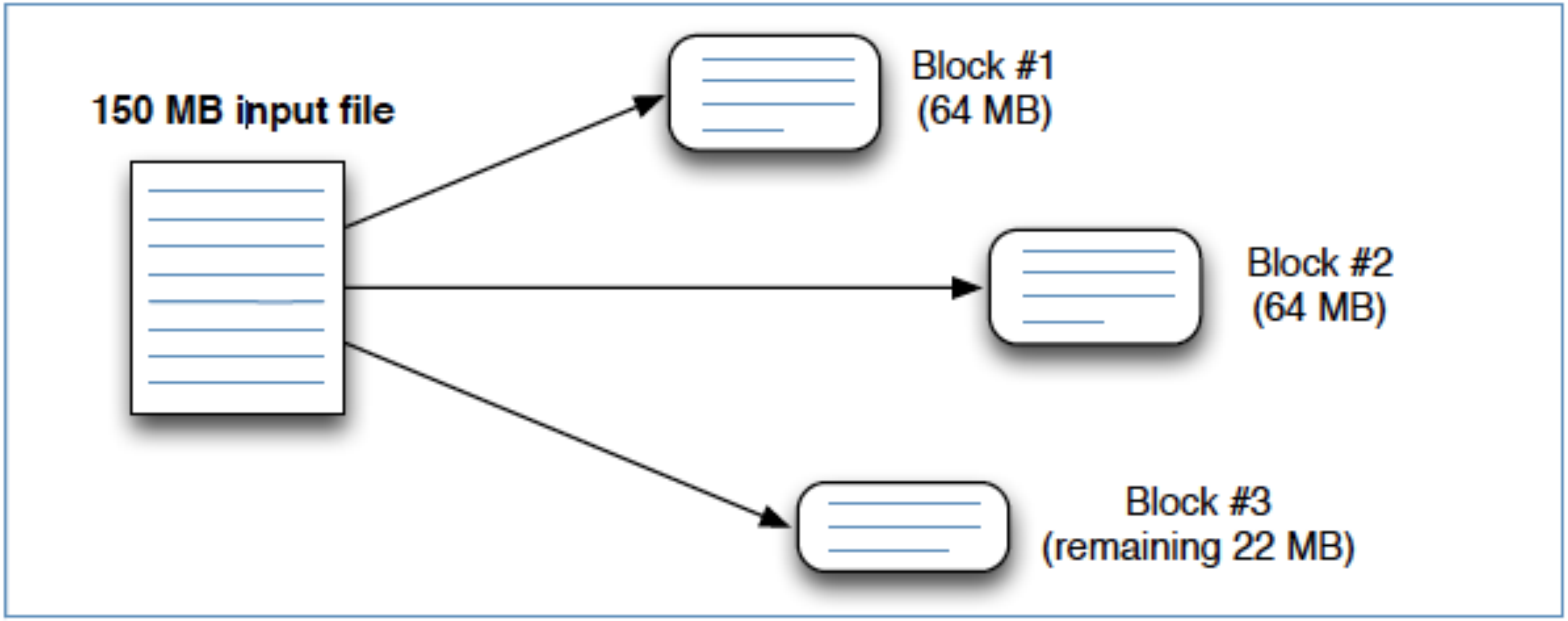
HDFS: Design

- Single PetaByte file system
  - Files split into 64 MB blocks (shards)
  - Blocks are replicated across several
    data nodes (x3 default)
- Single namenode stores metadata
  (file names, block locations, etc.)



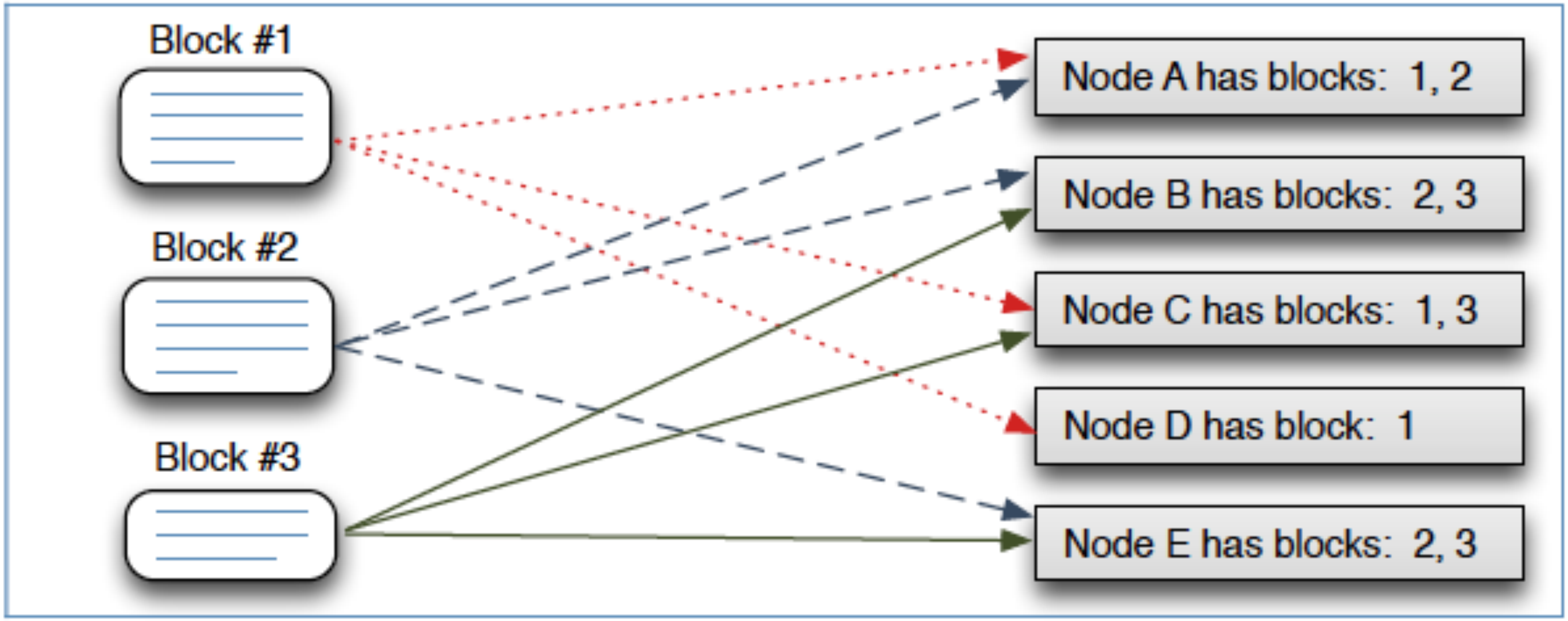
HDFS Blocks

- File is split into 64 MB blocks when adding to HDFS



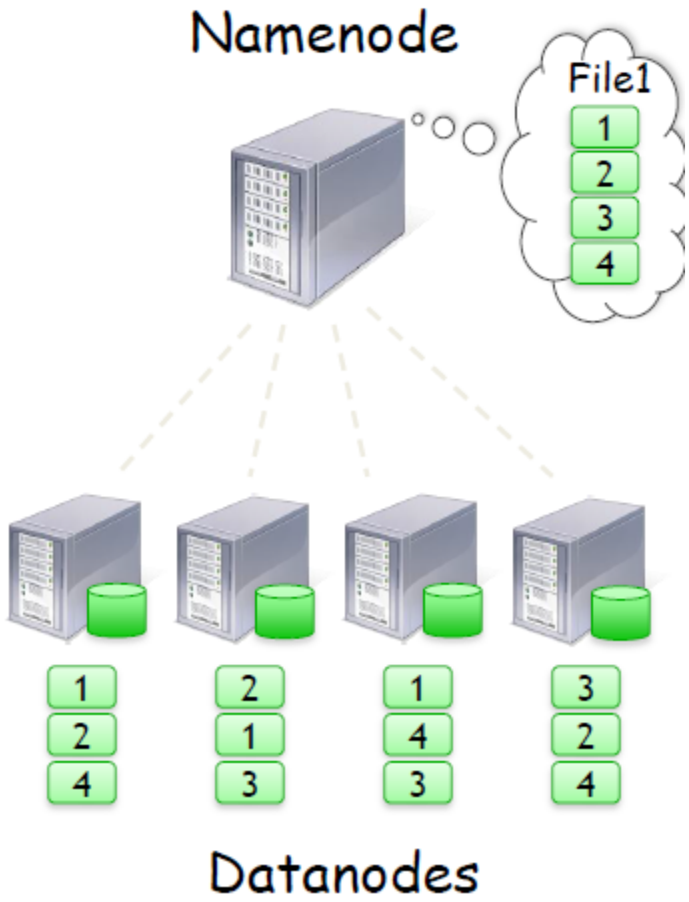
HDFS Replication

- Blocks are replicated to nodes throughout the cluster
- Benefits on reliability and performance



HDFS: Design

- Optimized for large files, sequential reads
- Files are append-only (no modification)
- Robust to failures, no need for backup
- Multiple sources for any one piece of data



HDFS: Design

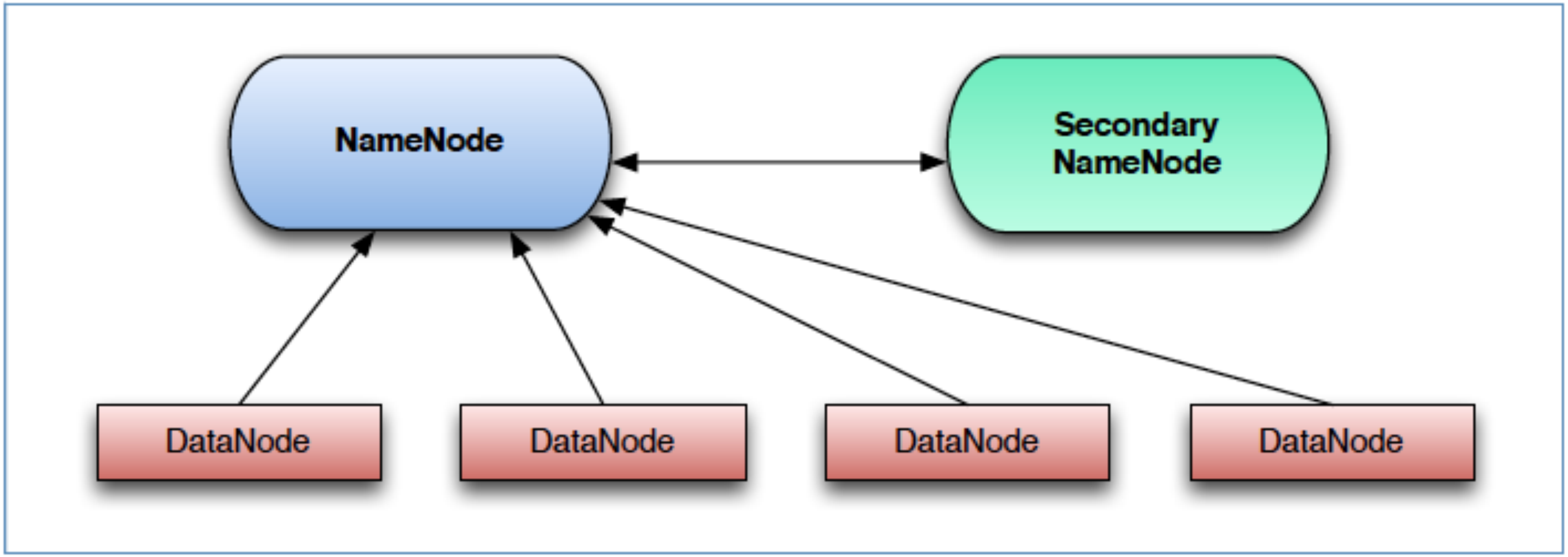
- Single Namespace for the entire cluster
- Data Coherency
  - Write-once-read-many access model
  - Client can only append to existing files

HDFS Features

- High performance
- Fault tolerance
- Simple Centralized management
- Security
  - Two levels to choose
- Optimized for MR processing
  - Data locality
- Scalability

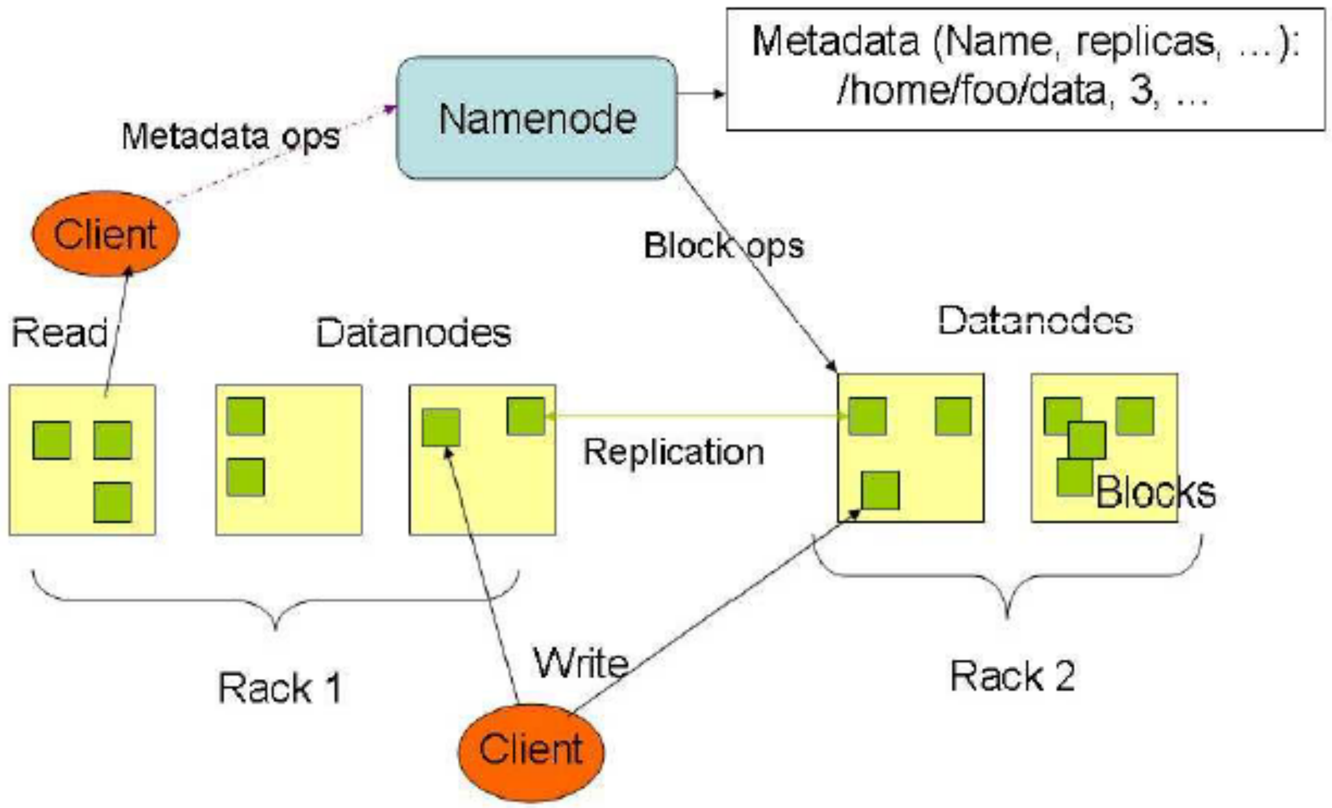
Classical Architecture

- Three daemons
  - NameNode (master)
  - Secondary NameNode (master)
  - DataNode (slave)



HDFS: Operations

- HDFS Architecture:



NameNode

- Stores all metadata
  - File locations in HDFS
  - File ownership/permissions
  - Name and location of blocks
- Metadata stored on disk
  - Named *fsimage*
  - Read when NameNode daemon starts up
- Store metadata change in memory
  - Also write to log file on disk named *edits*

Metadata and Transaction Log

- 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, etc.

- A Transaction (Edit) Log:
  - Records file creation, file deletion, etc.

DataNode

- A Block Server:
  - Stores actual contents of the file in the local file system (e.g., ext3)
    - Blocks named *blk_xxxxxxx*
  - Stores meta-data of a block (e.g., CRC)
  - Nothing about the file this block belongs to
    - Stored in NameNode's medadata
- A DataNode daemon on each node
  - Controls access to blocks
  - Communicate with NameNode

DataNode

- Block Report:
  - Periodically sends a report of all existing blocks to the NameNode
- Facilitates Pipelining of Data:
  - Forwards data to other specified DataNodes

Block Placement

- Current Strategy:
  - One replica on local node
  - Second replica on a remote rack
  - Third replica is randomly placed
- Different blocks from a given file in one rack are placed close to each other with highest bandwidth.
- Clients read from nearest replica
- Would like to make this policy pluggable

Secondary NameNode

- Performs memory-intensive administrative functions for NameNode
  - Not a failover NameNode
  - Periodically combines a prior snapshot of the file system metadata and edit log into a new snapshot
  - New snapshot is sent back to NameNode
- Run it on a separate machine
  - Needs as much RAM as the NameNode

Metadata Snapshot and Edit log

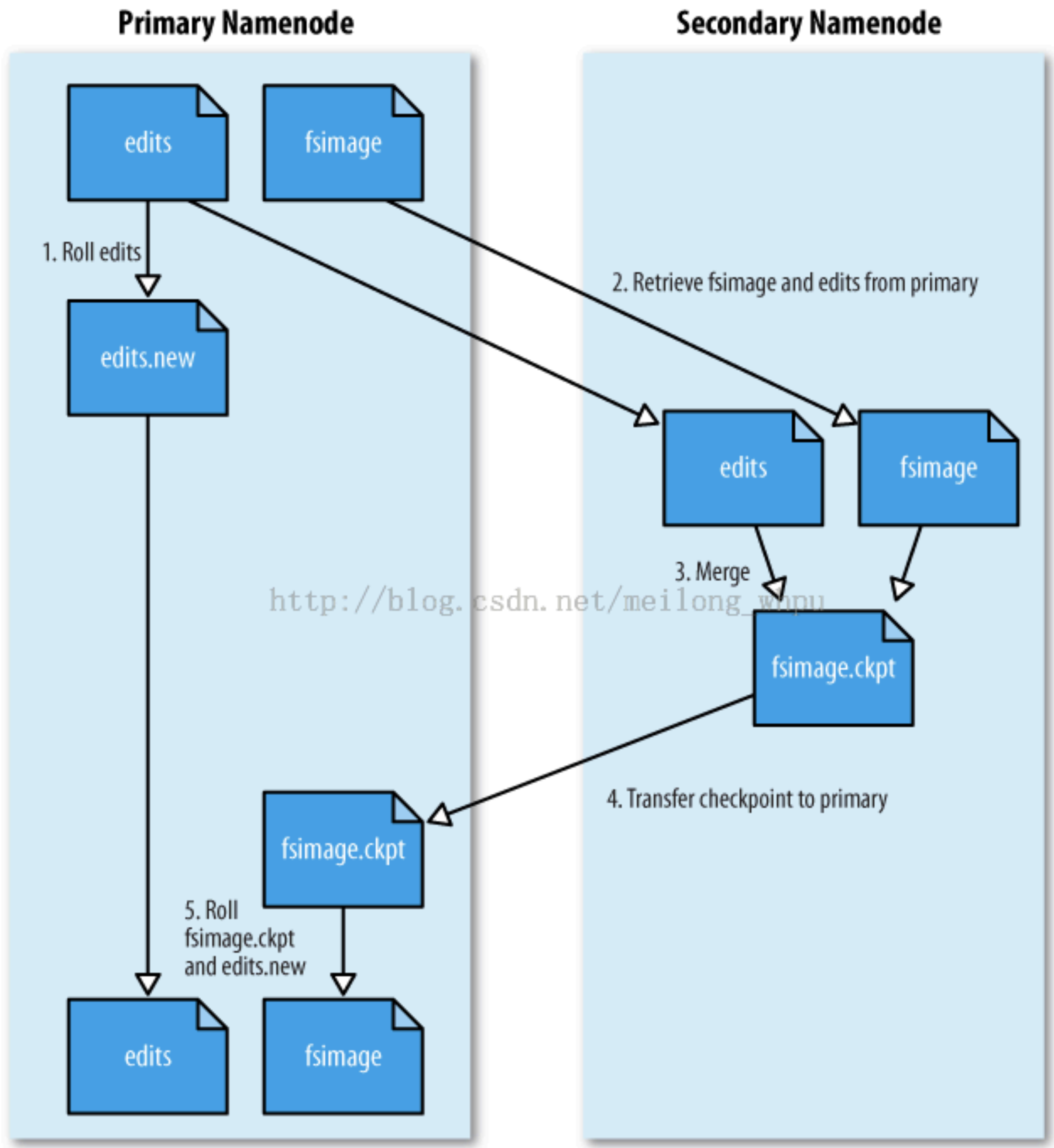
- *fsimage* file contains a metadata snapshot
  - Not updated at every write (too slow)
- On HDFS write
  - Recorded in NameNode's edit log: the *edits* file
  - NameNode's in-memory representation of the file system metadata is also updated
- May apply all changes in *edits* during a NameNode restart
  - File can grow huge
  - Would take a long time

Checkpointing the Metadata

- Secondary NameNode periodically checkpoints the NameNode's in memory file system data
  - Tells the NameNode to roll its *edits* file
  - Retrieves *fsimage* and *edits* from the NameNode
  - Loads *fsimage* into memory and applies the changes from the *edits* file
  - Creates a new, consolidated *fsimage* file
  - Sends the new *fsimage* file back to the NameNode
  - NameNode replaces old *fsimage* with the new one
    - Replaces the old *edits* file with the new one created in step 1
    - Updates the *fstime* file to record the checkpoint time

Checkpointing

- Occurs once an hour
  - or if the *edits* file grows larger than 64MB



Single Point of Failure

- Each Hadoop Cluster has a single NameNode
  - Secondary NameNode is not a failover NameNode
  - It is SPOF
- In practice, not a major issue
  - HDFS may be unavailable during failure
  - Little risk of data loss

High Availability

- High Availability leveraging Replication
- NameNode detects DataNode failure
  - Chooses new DataNodes for new replica
  - Balances disk usage
  - Balances communication traffic to DataNodes

Data Correctness

- Use checksum 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

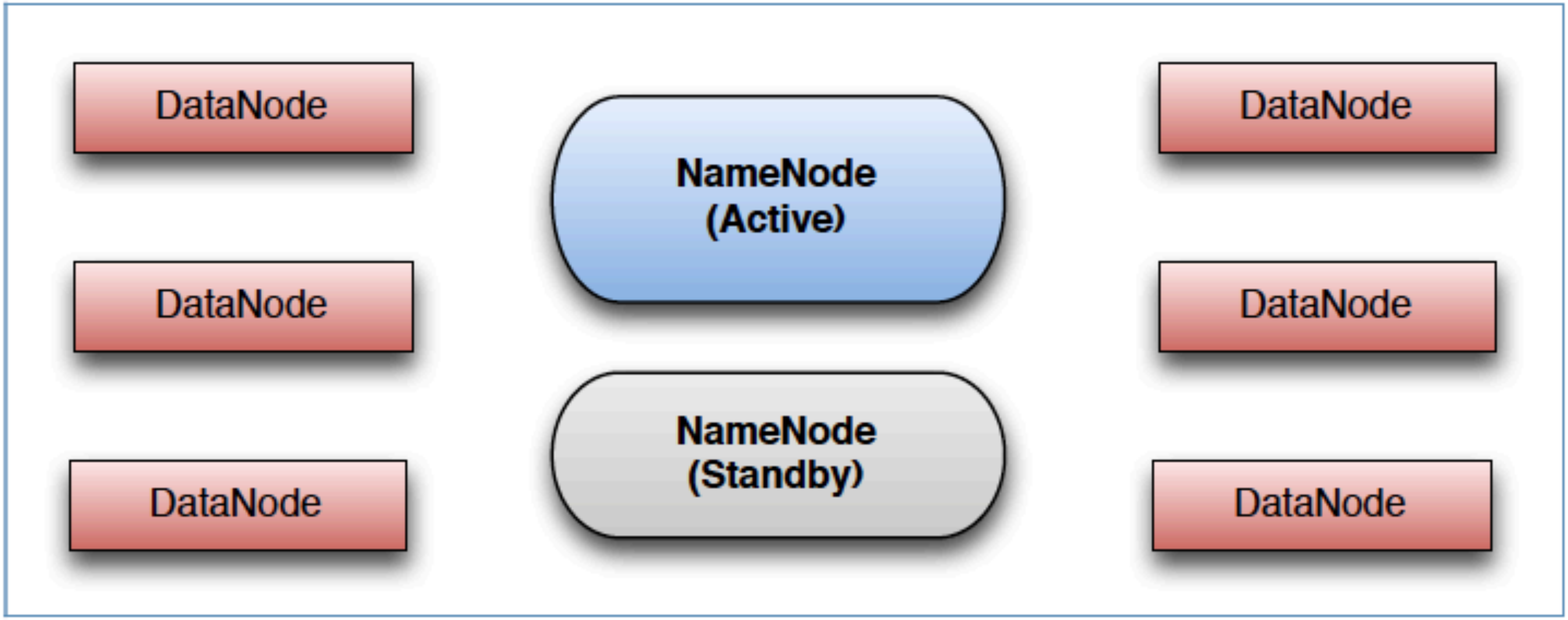
NameNode Failure

- A single point of failure
- Transaction/Edit Log is stored in multiple directories
  - A directory on the local file system
  - A directory on a remote file system (NFS/CIFS)
- Need to develop a real HA solution for the NameNode!

HA for Hadoop 2.x

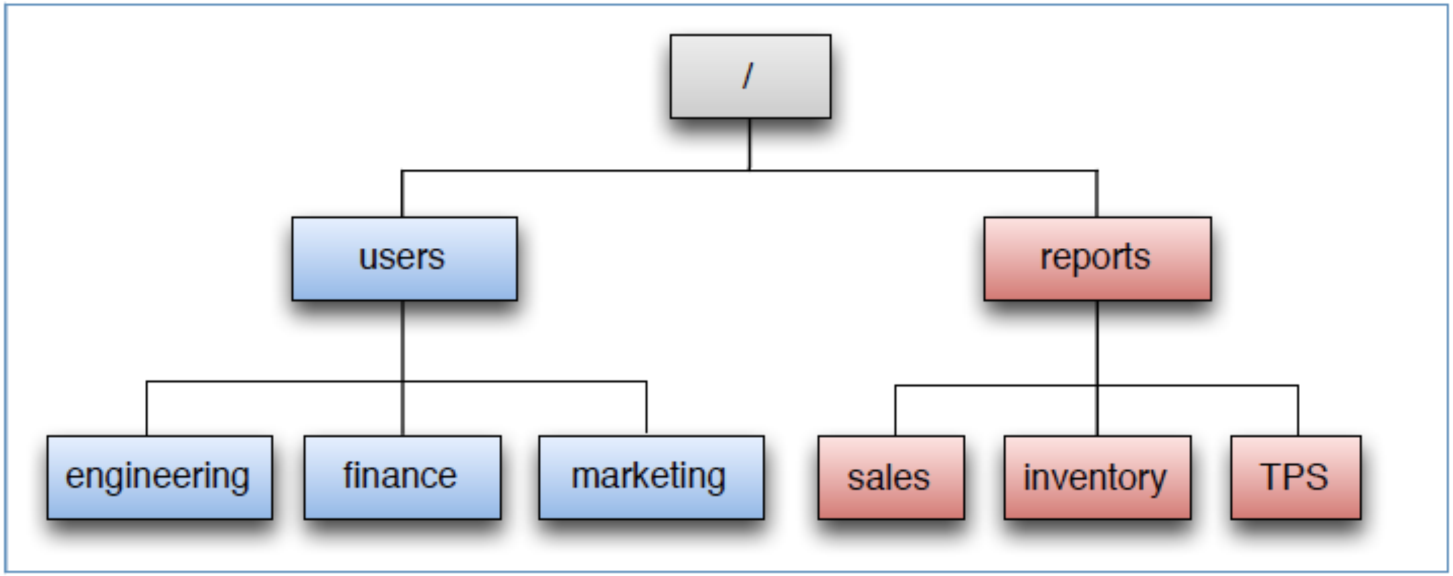
- Initially designed on Apache 0.23 branch
- Address the NameNode SPOF
- Two NameNodes
  - Standby takes over when active fails
  - Standby also does checkpointing (Secondary no longer needed)

HA


DataNode
NameNode
(Active)
DataNode
DataNode
DataNode
NameNode
(Standby)
DataNode
DataNode

HDFS Federation

- Federation improves the scalability of HDFS
  - Allows for multiple independent NameNodes
  - Each NameNode manages a namespace volume
  - Client-side mount tables define the overall view
  - NN1 manages */users* and NN2 manages */reports*



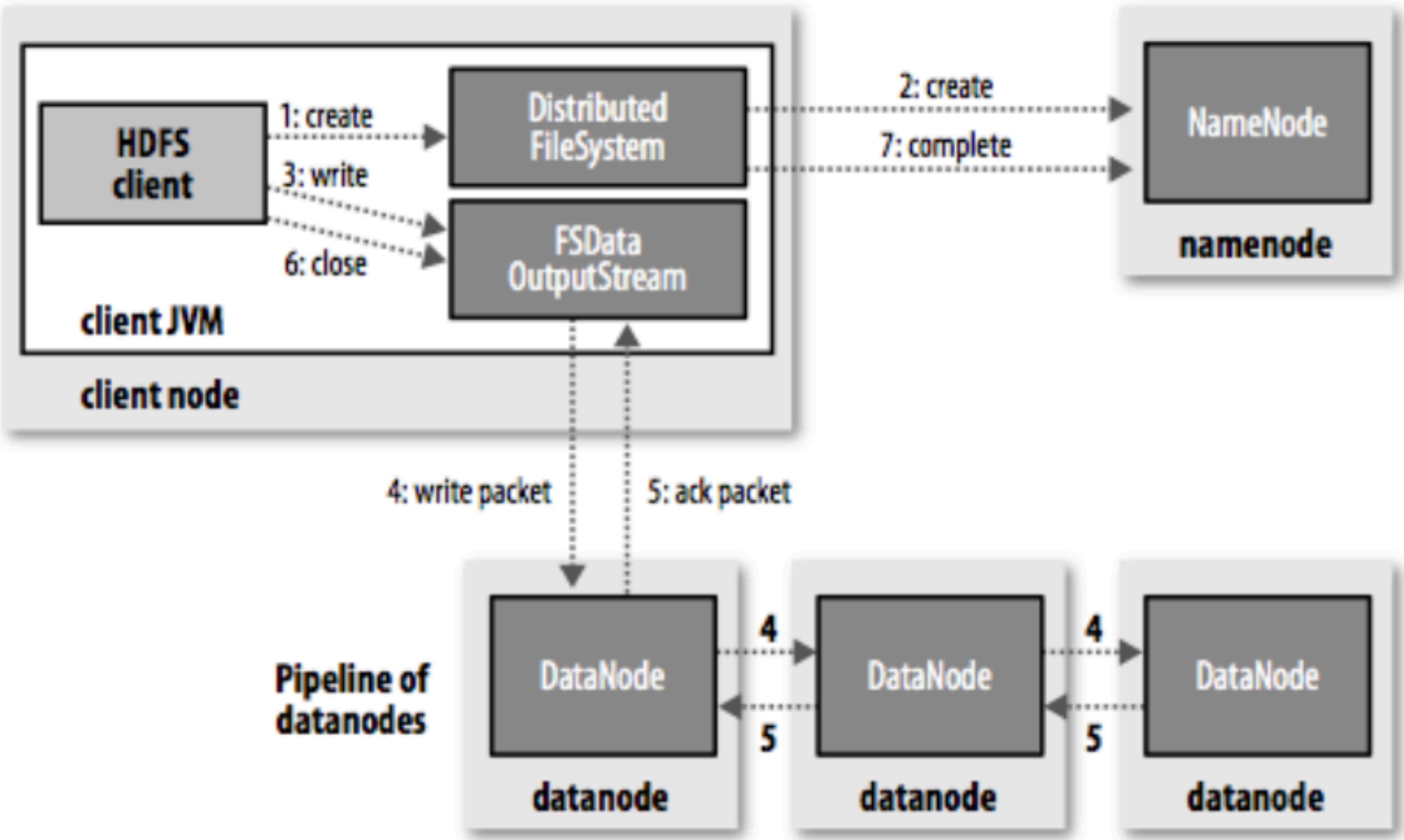
Data Pipelining

- 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 DataNode in the pipeline
- When all replicas are written, the client moves on to write the next block in the file.

Rebalancer

- Goal: the percentage of disk utilization 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

Anatomy of a File Write



Anatomy of a File Write

- Client connects to the NameNode
- NameNode places an entry for the file in its metadata, returns the block name and list of DataNodes to the client
- Client connects to the first DataNode and starts sending data
- As data is received by the first DataNode, it connects to the second and starts sending data
- Second DataNode similarly connects to the third
- ack packets from the pipeline are sent back to the client
- Client reports to the NameNode when the block is written

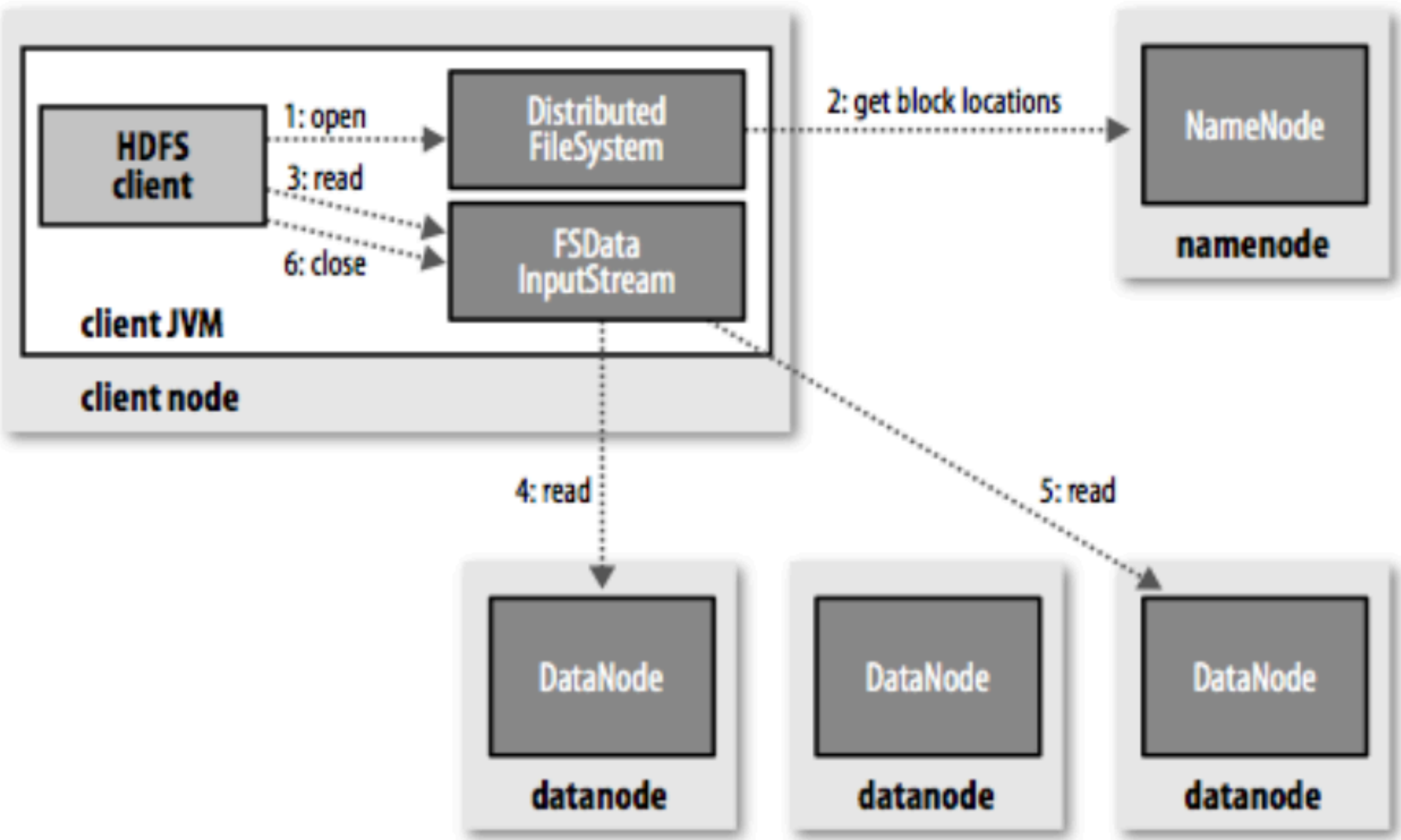
DataNode Failure in Pipeline

- If a DataNode in the pipeline fails
  - The pipeline is closed
  - A new pipeline is opened with the two good nodes
  - The data continues to be written to the two good nodes in the pipeline
  - The NameNode will realize that the block is under-replicated, and will re-replicate it to another DataNode
- As the blocks are written, a checksum is also calculated and written
  - Used to ensure the integrity of the data when it is later read

Rack-aware

- Hadoop understands the concept of 'rack awareness'
  - The idea of where nodes are located, relative to one another
  - Helps the JobTracker to assign tasks to nodes closest to the data
  - Helps the NameNode determine the 'closest' block to a client during reads
- HDFS replicates data blocks on nodes on different racks
  - Provides extra data security in case of catastrophic hardware failure
- Rack-awareness is determined by a user-defined script

Anatomy of File Read



Anatomy of File Read

- Client connects to the NameNode
- NameNode returns the name and locations of the first few blocks of the file
  - Block locations are returned closest first
- Client connects to the first of the DataNodes, and reads the block
  - If the DataNode fails during the read, the client will seamlessly connect to the next one in the list to read the block

Data Corruption

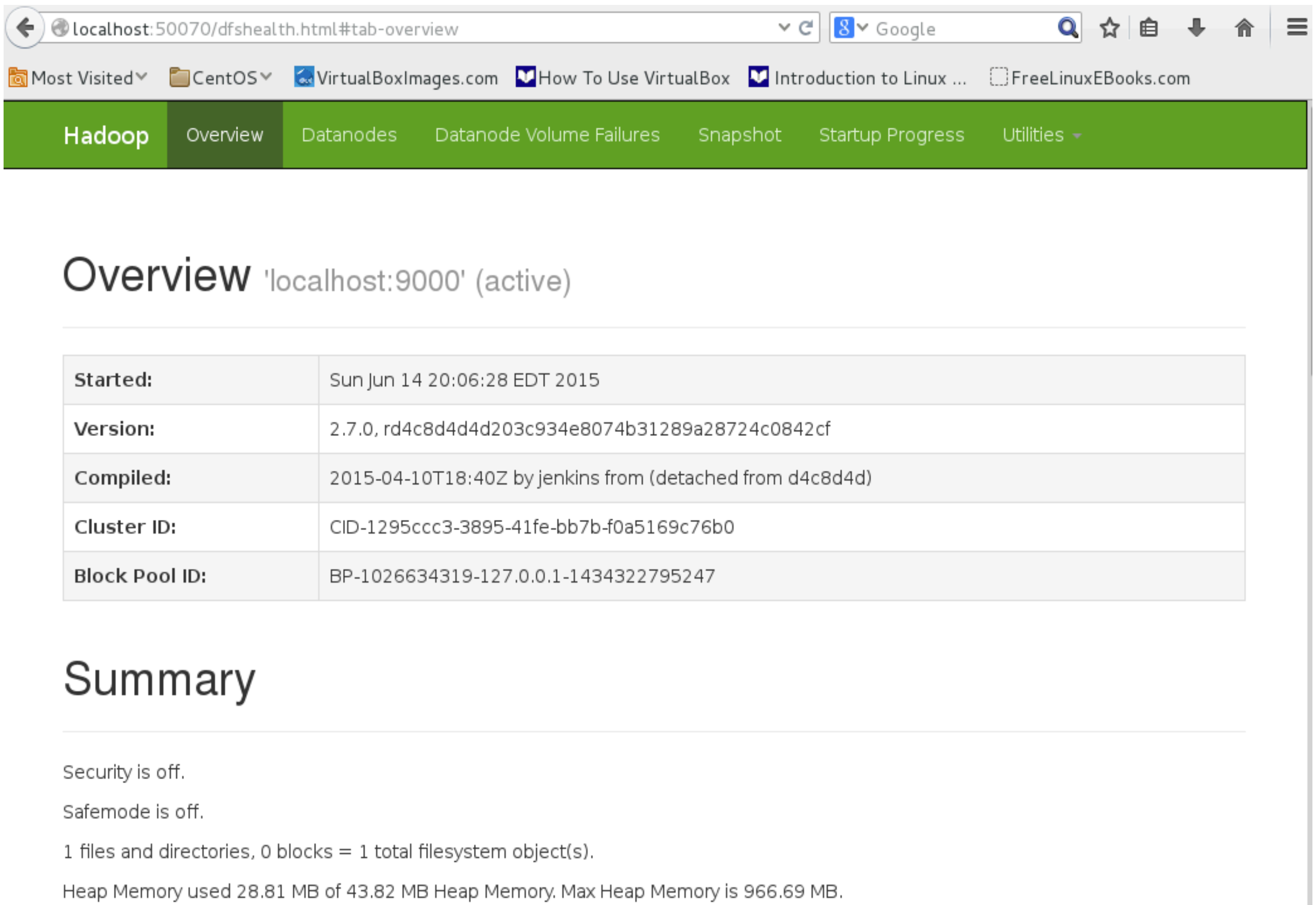
- As the DataNode is reading the block, it also calculates the checksum
- 'Live' checksum is compared to the checksum created when the block was stored
- If they differ, the client reads from the next DataNode in the list
  - The NameNode is informed that a corrupted version of the block has been found
  - The NameNode will then re-replicate that block elsewhere
- The DataNode verifies the checksums for blocks on a regular basis to avoid 'bit rot'
  - Default is every three weeks after the block was created

Reliablity and Recovery

- DataNodes send heartbeats-to the NameNode
  - Every three seconds
- After a period without any heartbeats, a DataNode is assumed to be lost
  - NameNode determines which blocks were on the lost node
  - NameNode finds other DataNodes with copies of these blocks
    - These DataNodes are instructed to copy the blocks to other nodes
  - Three-fold replication is actively maintained

NameNode Web UI

- NameNode exposes its Web UI on port 50070



Overview 'localhost:9000' (active)

Started:	Sun Jun 14 20:06:28 EDT 2015
Version:	2.7.0, rd4c8d4d4d203c934e8074b31289a28724c0842cf
Compiled:	2015-04-10T18:40Z by jenkins from (detached from d4c8d4d)
Cluster ID:	CID-1295ccc3-3895-41fe-bb7b-f0a5169c76b0
Block Pool ID:	BP-1026634319-127.0.0.1-1434322795247

Summary

Security is off.

Safemode is off.

1 files and directories, 0 blocks = 1 total filesystem object(s).

Heap Memory used 28.81 MB of 43.82 MB Heap Memory. Max Heap Memory is 966.69 MB.

HDFS: Commands Shell

- Execute:
  - Echo alias hfs = ‘hadoop fs ’ >> .bashrc
  - Source .bachrc

- Common Hadoop FS shell commands:
- hfs // See available commands
- hfs -help // more command details

HDFS: Commands Shell

- hfs -ls [<path>] // List files
- hfs -cp <src> <dst> // Copy files
- hfs -mkdir <path> // Create directory
- hfs -rm <path> // remove a file
- hfs -chmod <path> // Modify // permissions
- hfs -chown <path> // Modify owner

HDFS: Commands Shell

- Remote access commands:
  - hfs -cat <src> // Cat content to stdout
  - hfs -copyFromLocal <localsrc> <dst> // Copy stuff
  - hfs -copyToLocal <src> <localdst> // Copy stuff

The file system is browsable.

Moving/Copying

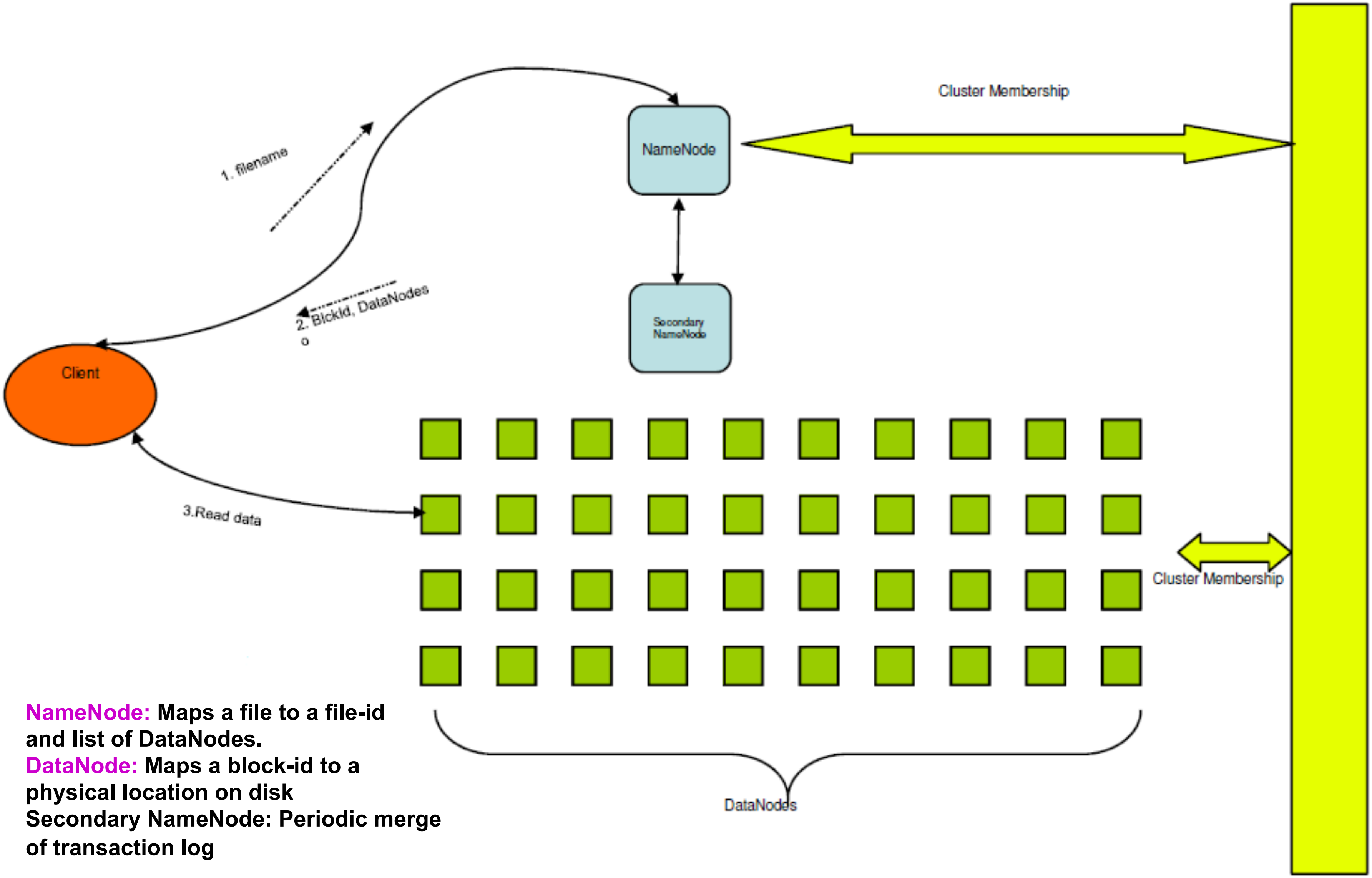
- Moving/copying file between Local FS and Hadoop HDFS:
  - Pulling files out from my Linux home directory to HDFS:
    - **hadoop** fs **-copyFromLocal** /home/$USERNAME/out ~/myfiles_out
    - **hadoop** fs **-moveFromLocal** /home/$USERNAME/out hdfs://hd.itu.edu:8020/user/myname/myfiles_out
    - **hadoop** fs **-put** /home/$USERNAME/out ~/myfiles_out

Moving/Copying

- Pulling files out from HDFS into my Linux home directory:
  - **hadoop** fs **-copyToLocal** /user/$USERNAME/out ~/myfiles_out
  - **hadoop** fs **-moveToLocal** /user/$USERNAME/out /home/dliang/out/myfiles_out
  - **hadoop** fs **-get** /user/$USERNAME/out ~/myfiles_out

Moving/Copying

- Moving files between file systems is not permitted
  - **hadoop** fs **-mv** **URI [URI…] <dest>**
  - **hadoop** fs **-mv** hdfs://hadoop.itu.edu:8020/user/myname/myf1 hdfs://hadoop.itu.edu:8020/user/myname/myf2 hdfs://hadoop.itu.edu:8020/user/myname/dir



NameNode: Maps a file to a file-id and list of DataNodes.
DataNode: Maps a block-id to a physical location on disk
Secondary NameNode: Periodic merge of transaction log

Hadoop Security: Hadoop Security Model

HDFS Security

- Files in HDFS have an owner, a group, and permissions
  - Very similar to Unix file permissions
- File permissions are read (r), write (w) and execute (x) for each of owner, group, and other
  - x is ignored for files
  - For directories, x means that its children can be accessed
- HDFS permissions are designed to stop good people doing foolish things
  - Not to stop bad people doing bad things!
  - HDFS believes you are who you tell it you are

Hadoop Security

- Laws governing data privacy
  - Particularly important for healthcare and finance industries
- Export control regulations for defense information
- Protection of proprietary research data
- Company policies
  - Different teams in a company have different needs
- Setting up multiple clusters is a common solution
  - One cluster may contain protected data, another cluster does not

Security Terms

- Security
  - Computer security is a very broad topic
  - Access control is the area most relevant to Hadoop
  - We'll therefore focus on authentication and authorization
- Authentication
  - Confirming the identity of a participant
  - Typically done by checking credentials (username/ password)
- Authorization
  - Determining whether participant is allowed to perform an action
  - Typically done by checking an access control list

Types of Security

- Support for HDFS file ownership and permissions
  - Provides only modest protection
  - User/group authentication is easily subverted (client-side)
  - Mainly intended to guard against accidental deletions/ overwrites
- Enhanced security with Kerberos
  - Provides strong authentication of both clients and servers
  - Tasks can be run under a job submitter's own account
  - This enhanced security is optional (disabled by default)

Security Design Considerations

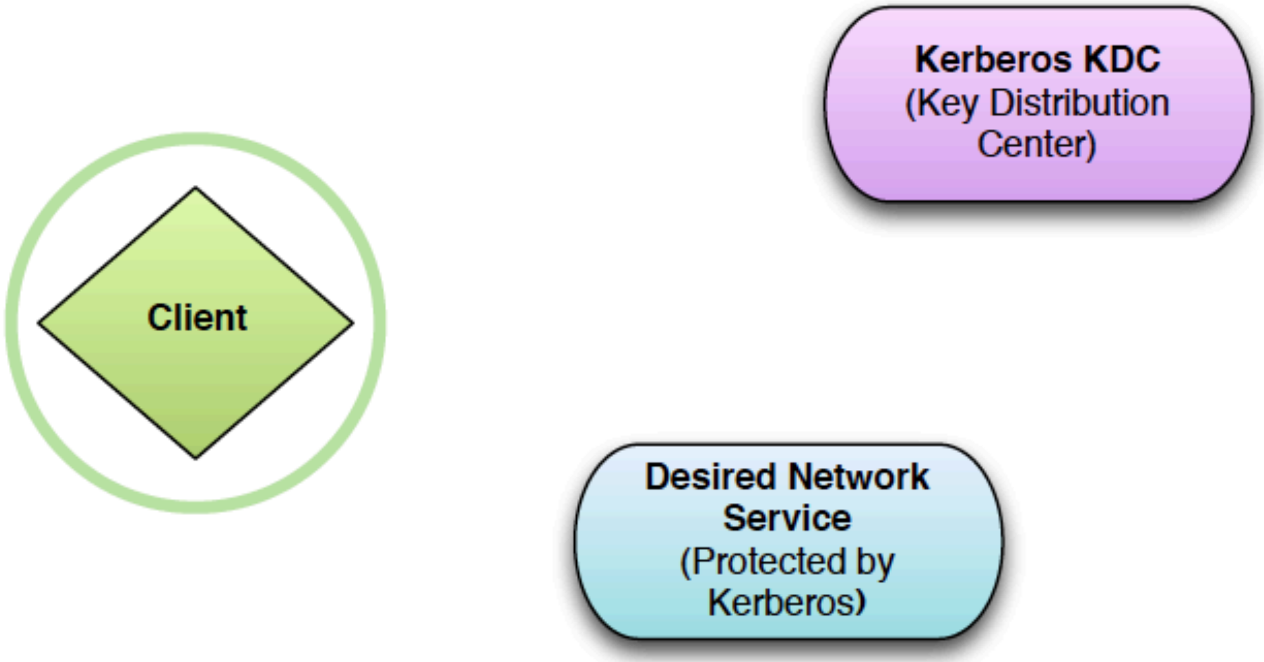
- Hadoop security does not provide
  - Encryption for data transmitted on the wire
  - Encryption for data stored on disk
- The security of a cluster is enhanced by isolation
  - It should ideally be on its own network
  - Access to nodes/network should be limited for untrusted users

Kerberos

- Kerberos involves messages exchanged among three parties
  - Client
  - The server providing a desired network service
  - The Kerberos Key Distribution Center (KDC)

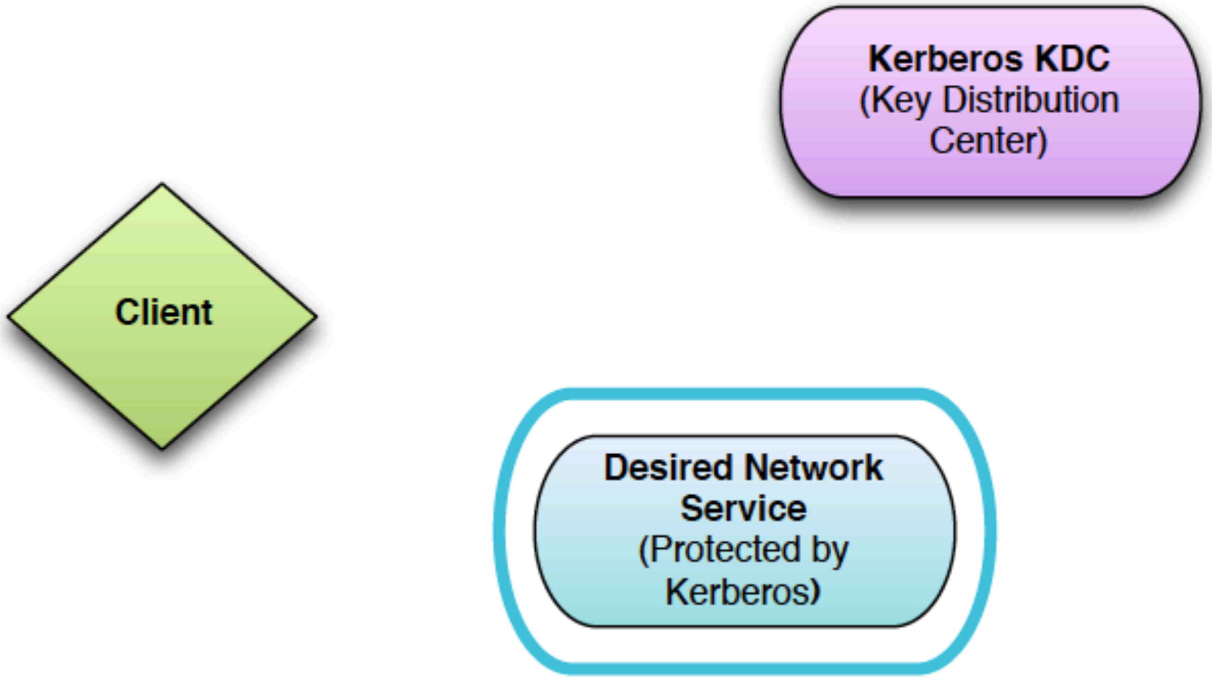
Kerberos

- Client is a software which request access to a service
  - *hadoop fs*



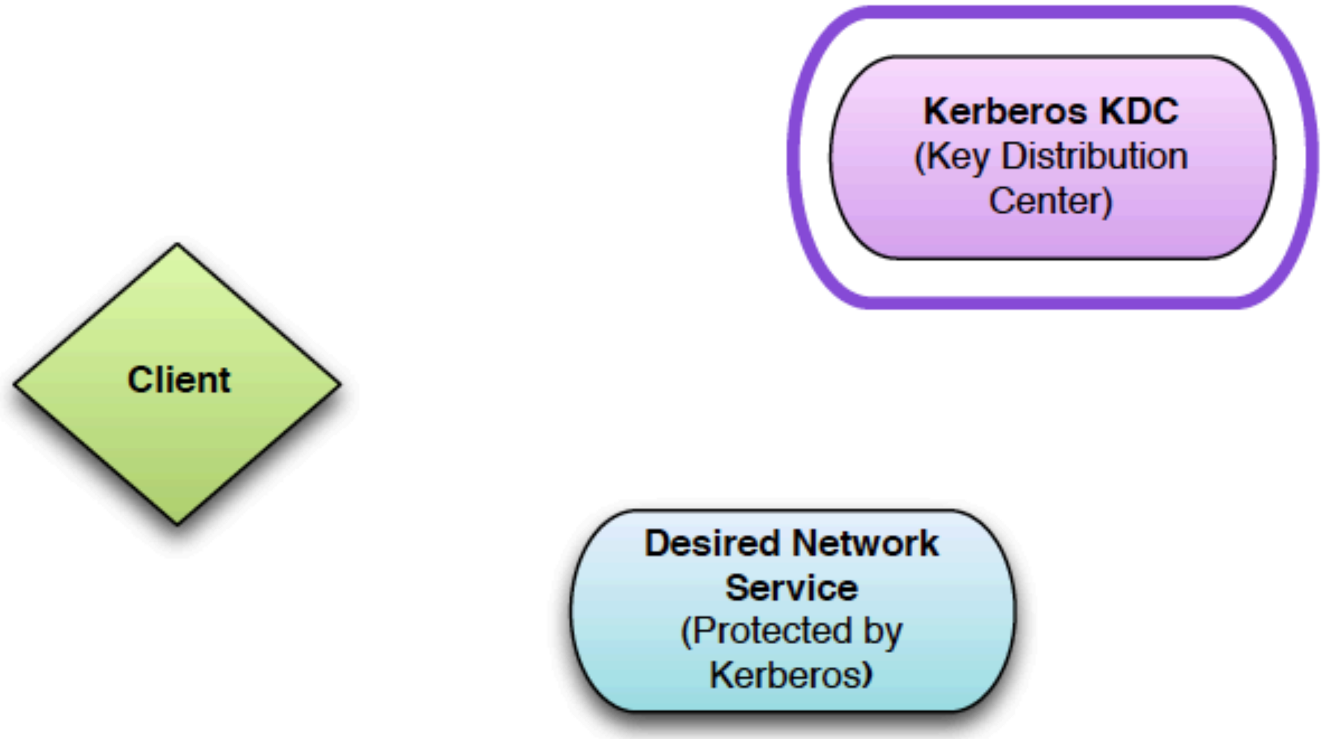
Kerberos

- Service is a Hadoop service daemon
  - NameNode, JobTracker, etc



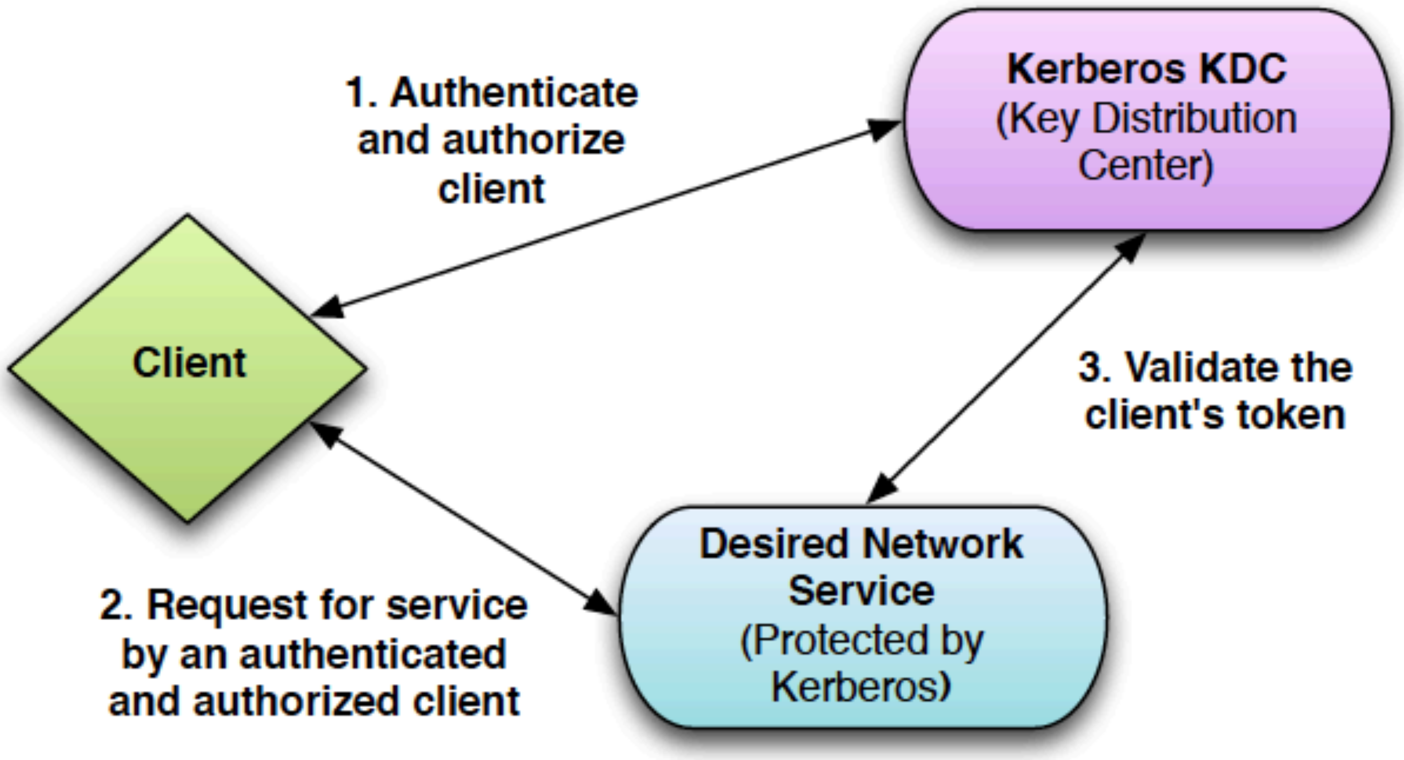
Kerberos

- KDC authenticates and authorizes a client
- Not part of Hadoop



General Kerberos Concept

- Service don't directly authenticate client



General Kerberos Concept

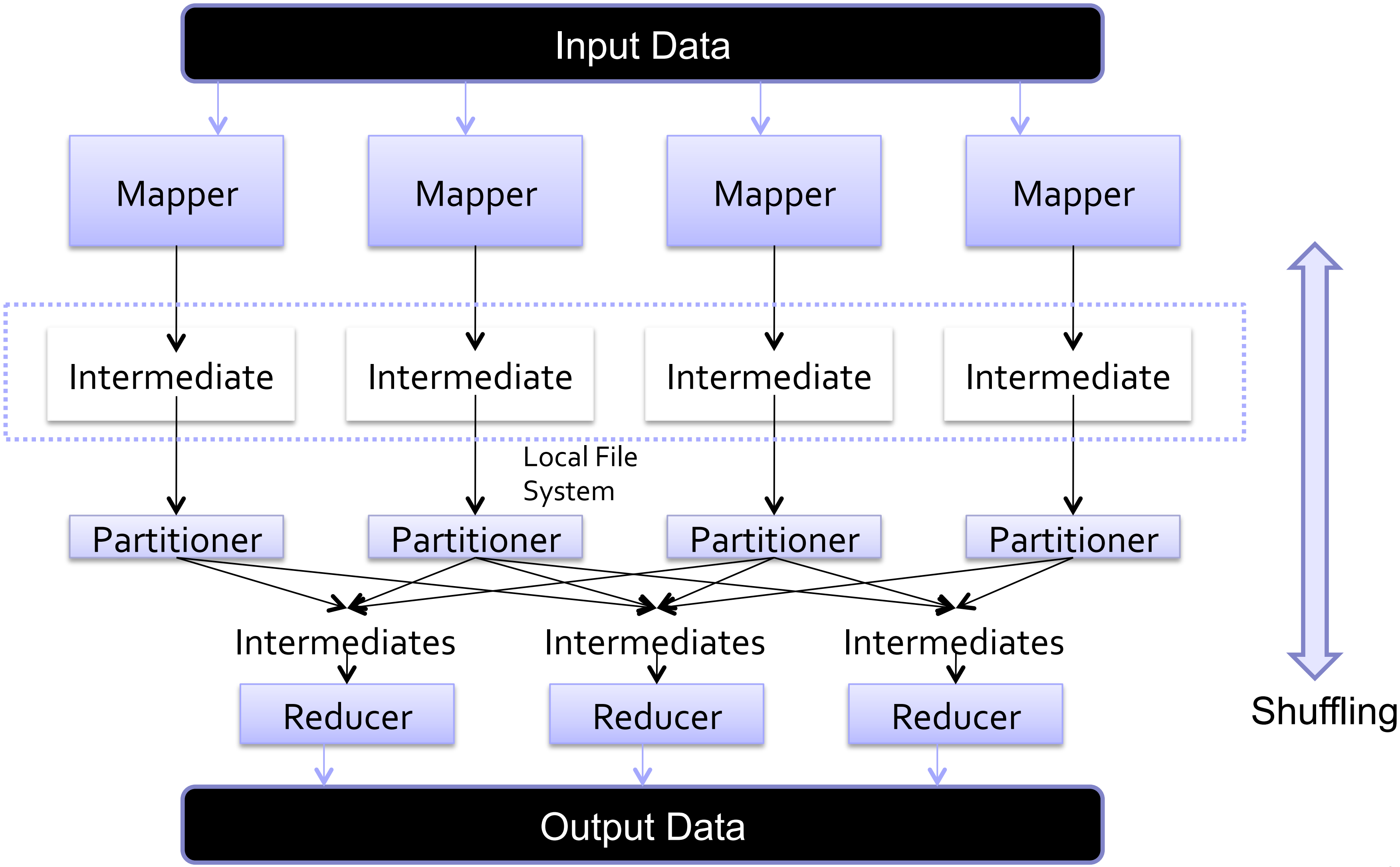
- Authenticated status is cached
  - You don’t need to explicitly submit credentials with each request
- Passwords are not sent across network
  - Instead, passwords are used to compute encryption keys
  - The Kerberos protocol uses encryption extensively
- Timestamps are an essential part of Kerberos
  - Make sure you synchronize system clocks (NTP)
- It’s important that reverse lookups work correctly

Hadoop Security

- Hadoop's security has had authorization for sometime
  - The ability to allow people to do some things but not others
  - Example: file permissions
- Authentication has historically been relatively weak
  - Authorization requires you to first identify the user
  - Hadoop's default mechanism for doing this is easily defeated
  - Intended to prevent stupid mistakes made by honest people
- Hadoop can now optionally enforce strong authentication
  - Via integration with Kerberos

Hadoop IO: Shuffling Optimization in Hadoop MR

Hadoop IO: Shuffling in Hadoop MR



Shuffling Problem

- Mapper needs to write to local file system, then reducer needs to read from the file system – 2 I/Os.
- Main observation is shuffling hurts response time
- No reducer can start processing till all mapper are completed.

Shuffling Optimization

- Put MapOutputFile into memory instead of local disk
- Two-level intermediate file storage
  - In-memory
  - Local disk (traditional) – needed for recovery
- The main idea is to pass the data from mapper output to reducer input through memory and IPC and in parallel still do the disk IO for recovery.

Shuffling Optimization

- Absence of failure, reducers can start much earlier as IPC is much faster than 2 I/Os
- Mellonox (IBM partner) optimization is similar to the above approach but uses RDMA instead of IPC to pass the data between mapper and reducer.

Summary

- We covered Hadoop component HDFS
  - Overview
  - NameNode and DataNode
  - HDFS commands
- We discussed the Hadoop shuffling issue and covered different flavors of optimizing this issue