HDFS – Hadoop Distributed File System

Hadoop: The Definitive Guide
4th edition
Tom White
O'Reilly

Chapter 3

Need for distributed file systems

- Large datasets
- Limited storage capacity of individual machines

DFS vs Local File System

- Stores data across a network of machines (partitioned)
- Uses network based programming
- Biggest challenge – Failure tolerance & Handling Data Loss

HDFS

- Stores ”very large files”
- Capable of handling streaming data
  - Write-once Read-many times principle
  - Time to read the whole dataset is more important than the latency in reading the first record
- Can be setup on commodity hardware

When HDFS is not suitable

- Low-latency data access
  - Not suitable for applications which need data access in milliseconds
  - HDFS delivers data at high throughput but the latency is high
    - HBase for low latency access
- Storing lots of small files
  - Name node stores the file system in its memory
    - Limit to number of files in filesystem
  - Each file/block/directory entry takes up to 150 bytes
  - Storing a single large file is preferable when compared to multiple small ones
- Multiple writers & arbitrary file modifications
  - HDFS supports only appending a dataset
  - Files have to be written only by a single user/entity

HDFS – Concepts

Data Blocks

- Minimum amount of data that can be read/written
  - Data - integral multiple of the disk block size
- Default size – 128 MB
- Large dataset is broken down into many data blocks and stored across many nodes in HDFS
- Large block size is always preferable – Less disk seek time
- User can access only the dataset as a whole (Block Level Abstraction)

- Block level commands

  % **hdfs fsck / -files -blocks**

  lists the blocks that make up each file in the filesystem

  % **hdfs –du /user/hadoop/dir1**

  shows the capacity, free and used space of the filesystem

- Fault tolerance – Replication
- Default replication factor - 3

Nodes

- HDFS cluster - 2 types of nodes
  - Name Node - Master node (Usually one Name node/cluster)
  - Data Node – Worker node

Name Node

- Doesn't store any data
- Manages the namespace of the HDFS in memory
- Namespace and metadata are organized as trees in memory
- This information is stored in 2 files
  - Namespace Image (fsimage) – the recent file system stored on disk (persistent)
  - Edit Logs – all the changes done to the fsimage stored on disk (persistent)
- Is aware of which data block belonging to which file is in which data node

Data Nodes

- Slaves (Multiple data nodes/cluster) - workhorses
- Stores all the data in HDFS
  - Store and retrieve blocks when told to (by clients or the namenode)
- Sends a list of blocks (block report) it is storing to Name Node periodically
- Adds/Deletes data blocks as instructed by the Name Nodes

Node Failure

- Name Node – Single point of failure
- If Name Node fails, the cluster becomes unusable
- Remedy 1 – Store the persistent state of the File System onto another node
- Remedy 2 – Use a Secondary Name Node
  - It takes over when the Name Node fails
  - Prevents the edit logs from becoming large by performing the edits on the persistent fsimage periodically
  - Lags behind the name node – In case of failure of NN, an absolute recovery is not possble

Block Caching

- Done at the data nodes (off-heap block cache)
- Frequently used blocks are cached
- Done to avoid high disk seek times & less running time for MR jobs
- Users/applications instructs Name Node on what files to cache & for how long by cache directive
- Cache pools are an administrative grouping for managing cache permissions and resource usage

HDFS Federation

- Single Name Node has limits on how much namespace it can hold in its memory
- HDFS Federation is released in Hadoop 2.x series
- Allows several name nodes to manage a part of the namespace
  - Example:
    - one name node might manage all the files rooted under */user*,
    - a second name node might handle files under */share*.

HDFS Federation

- In a federation each name node manages a namespace volume
- Namespace volume = Metadata + Block Pool
  - Block Pool - contains all the blocks for the files in the namespace
- Namespace volumes are independent (Failure of one Name node will not affect the other)
- Data nodes can store data from several namespace volumes

Recovery process upon Name Node failure (Hadoop 2.x)

- Introduces HDFS High Availability (HA)
- There are a pair of active name nodes in active-standby configuration
- If the active name node fails, the standby name node takes over
- Both these NN's will share edit logs stored in a highly available shared file system
- Block reports are sent to both these name nodes simultaneously
- Secondary name node is subsumed by stand-by name node

Choices for the highly available shared storage

- NFS (Network File System)
- Quorum Journal Manager (QJM)
  - Dedicated HDFS implementation to store edit logs in a highly shared storage
  - Most preferred choice
  - Runs a group of nodes (journal nodes)
  - Every edit should be written to the majority of the journal nodes
  - There are three journal nodes by default

Quorum Journal Manager (QJM)

- At any point, one of the NameNodes will be in Active state and the other will be in a Standby state.
- Active NameNode is responsible for all client operations in the cluster,
- Standby maintaining enough state to provide a fast failover.

Quorum Journal Manager (QJM)

- Problem - Standby node has to keep its state coordinated with the Active node
- Both nodes communicate with a group of separate daemons called ‘JournalNodes’ (JNs).
- When any namespace modification is performed by the Active node, it logs a record of the changes made, in the JournalNodes.
- Standby node reads the amended information from the JNs
- When Standby Node sees the changes, it then applies them to its own namespace.
- Hence, namespace state is fully synched before a failover occurs.
- For fast failover
  - DataNodes are configured with location of both NameNodes, and send block location information and heartbeats to both.

- If active namenode fails, the standby can take over very quickly (few tens of seconds)
  - it has the latest state available in memory:
    - both the latest edit log entries and an up-to-date block mapping.

- Name Node enters into safe mode when it is started after failure
- User will not be allowed to write into HDFS when a name node is in safe mode
- Exiting out of safe mode manually is not recommended as it will result in loss of some/entire namespace
- Name node will exit out of safe mode automatically after it receives enough block reports from the data nodes

Failover

- Failover controller
  - manages the transition from active to stand-by name node
  - Default failover controller – zookeeper implementation
  - light weight process running on both the name nodes
  - Makes sure that only one name node remains active at any given point of time

Graceful Failover

- Initiated manually by administrator
- Done for routine maintenance

Fencing

- Failover controller might be triggered when the active name node hasn't stopped (when the network is slow)
- Active name node thinks it is still active but the stand-by name node has taken over
- Fencing – HA implementation which prevents the previously active node from writing into the edit logs
- QJM – allows only one name node from writing into edit logs
- NFS Filer – Stronger fencing mechanisms must be manually implemented - not preferred

Fencing Mechanisms

- Revoking the previously active name node's access to shared storage
- Disabling network port by issuing remote commands
- Last resort – STONITH (Shoot The Other Node In The Head) – power down the previously active name node CS

Interfaces to HDFS

- CLI (Command Line) - Simplest and most commonly used interface to HDFS
- HTTP
- C
- NFS
- FUSE
- Java

HDFS Shell Commands

- mkdir
- ls
- lsr
- touchz
- appendToFile
- cat
- tail
- du
- dus
- count
- rm
- rmr

- cp
- mv
- copyFromLocal
- moveFromLocal
- put
- copyToLocal
- get
- moveToLocal
- getmerge
- stat
- test
- text
- expunge

- setrep
- chgrp
- chmod
- chown
- getfacl
- getfattr
- setfacl *
- setfattr *

[Source] https://hadoop.apache.org/docs/r2.6.3/hadoop-project-dist/hadoop-common/FileSystemShell.html
[*] https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/ExtendedAttributes.html

File Permissions in HDFS

- POSIX model
- Owners can grant rwx permissions to their files/folders to other HDFS users
- R – read the file, list contents of a folder
- W – to create new files/folders
- X – Ignored for a file & for folders – users can access their children

- Every file/folder in HDFS has Owner, Mode & Group
- Owner – User who owns the file
- Mode – Permissions for owners and other users the owner has shared his/her file with
- Group – user group
- By default Security in Hadoop is disabled
- Changing “dfs.permissions.enabled” property in hdfs-site.xml to true will activate the POSIX security model

Hadoop file systems

- HDFS is one of the abstract representation of the file systems in Hadoop
- List of file systems provided by Hadoop
  - Local file system - A filesystem for a locally connected disk with client-side checksums.
  - HDFS – Hadoop Distributed File System
  - WebHDFS - A filesystem providing authenticated read/ write access to HDFS over HTTP
  - Secure webHDFS – HTTPS version of WebHDFS
  - HAR - A filesystem layered on another filesystem for archiving files

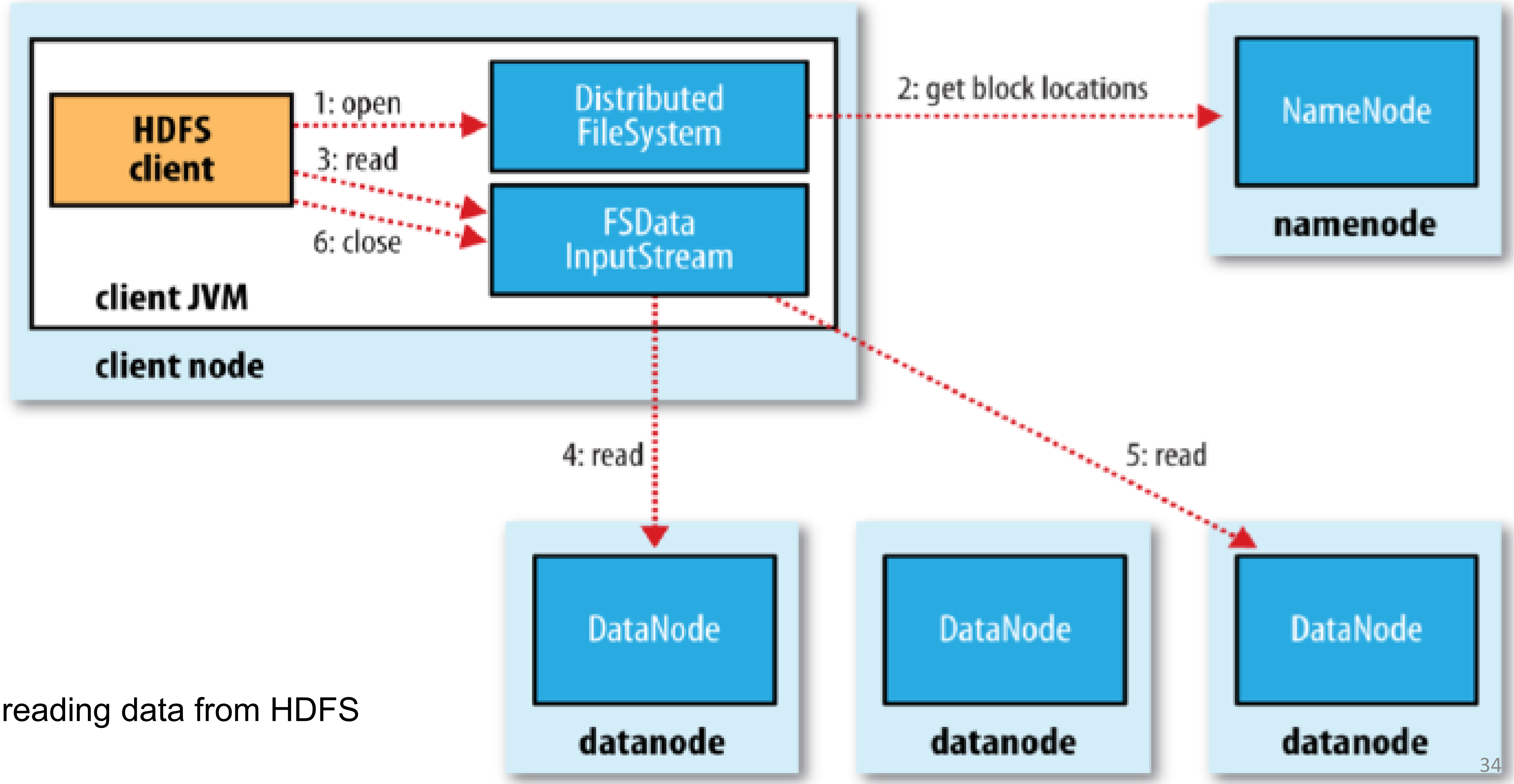
- Viewfs – file system for federated name nodes
- ftp – file system backed by a FTP server
- S3a - file system backed by a Amazon S3 server
- Wasb (azure) - file system backed by Microsoft Azure server
- Swift - file system backed by Openstack Swift

- Data stored in Hadoop file systems can be accessed by an URI
    - HDFS – hdfs://namenode:port/my/resource
    - WebHDFS – webhdfs://namenode:port/my/resource
    - Local – file://namenode:port/my/resource CS

Java Interface

- There are other intefaces – C, FUSE, HTTP, …
- HDFS exposes the HDFS with the “Hadoop FileSystem class”
- FileSystem API allows users to
  - Open a file
  - List contents of a folder
  - Delete a file
  - Create a file
  - Writing data into a file
  - Retrieving metadata about the files
- Refer to Chapter 3 for more detailed examples

Reading a file in HDFS



A client reading data from HDFS

Client reading data in HDFS

- Step 1: Client opens file it wishes to read by calling Open() on FileSystem Object
  - FileSystem Object- to access the computer file system and to manage file, folders and drives.
- Step 2: DFS gets data block locations from name node (a set of addresses of data nodes at once)
- For each block, the namenode returns the addresses of the datanodes that have a copy of that block.
- Datanodes are sorted according to their proximity to the client
- The DistributedFileSystem returns an FSDataInputStream (an input stream that supports file seeks) to the client for it to read data from.
- From the addresses of the datanodes, the client calls read() from FSInputStream class, to read the contents of the data blocks
- FSDataInputStream in turn wraps a DFSInputStream, which manages the datanode and namenode I/O.

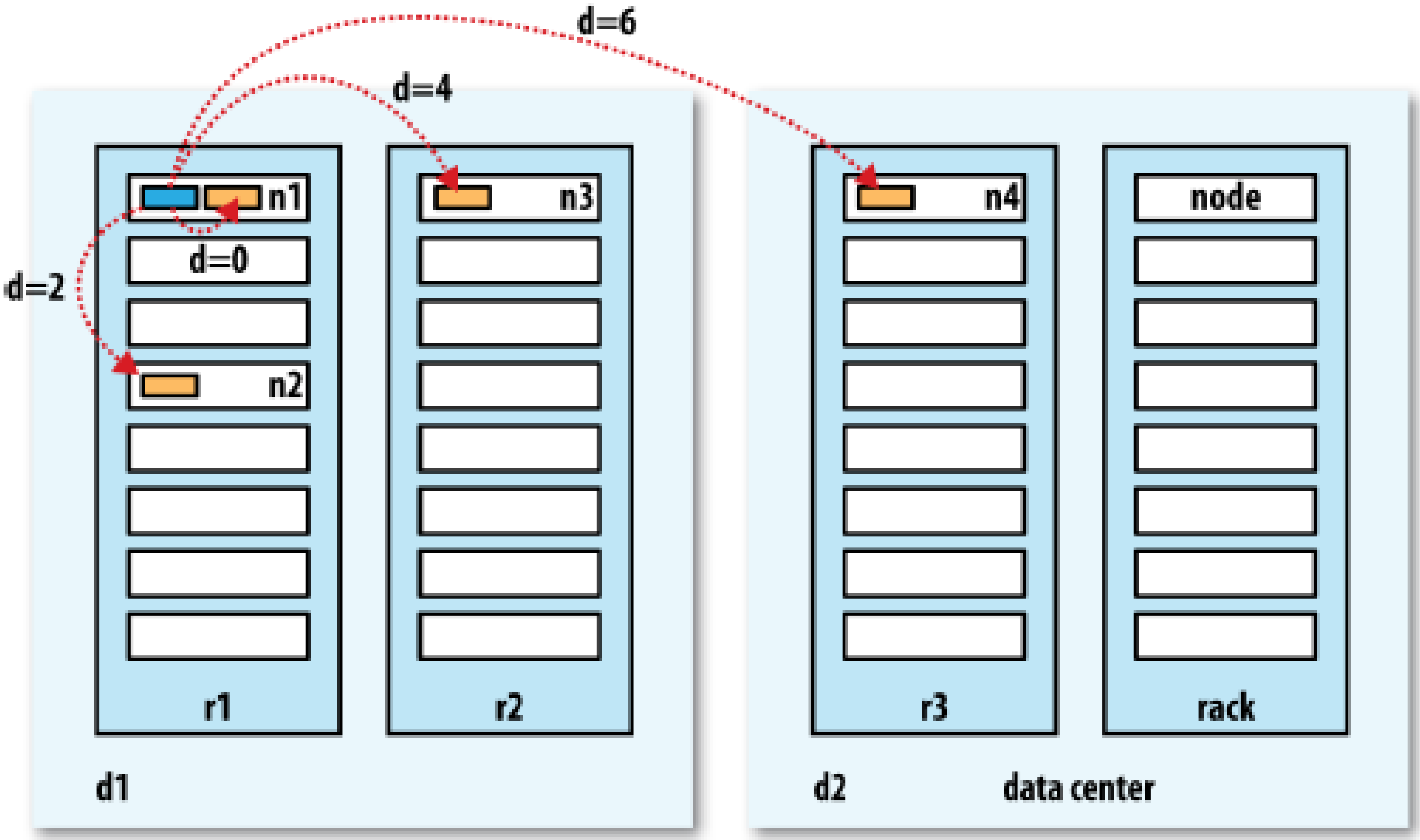
- Step 3: The client then calls read() on the stream
- Step 4: Data is streamed from the datanode back to the client, which calls read() repeatedly on the stream
- Step 5: When the end of the block is reached, DFSInputStream will close the connection to the datanode, then find the best datanode for the next block.
- Now the client gets locations of some more data nodes
- This process continues until all the data nodes are read
- Blocks are read in order, with the DFSInputStream opening new connections to datanodes as the client reads through the stream.
- It will also call the namenode to retrieve the datanode locations for the next batch of blocks as needed.
- Step 6: When the client has finished reading, it calls close() on the FSDataInputStream

Network Topology & Hadoop

- Network bandwidth is used as a measure of distance (scarce)
- How easy is it to measure bandwidth?
- Requires a quiet cluster
- Number of pairs of nodes increases as the number of nodes increases
- Hadoop represents nodes as trees.
- Distance between nodes = sum of common ancestors
- What can levels represent in this tree?

- Bandwidth availability
  - Processes on same node
  - Nodes on same rack
  - Nodes on different rack but same data center
  - Nodes on different data centers

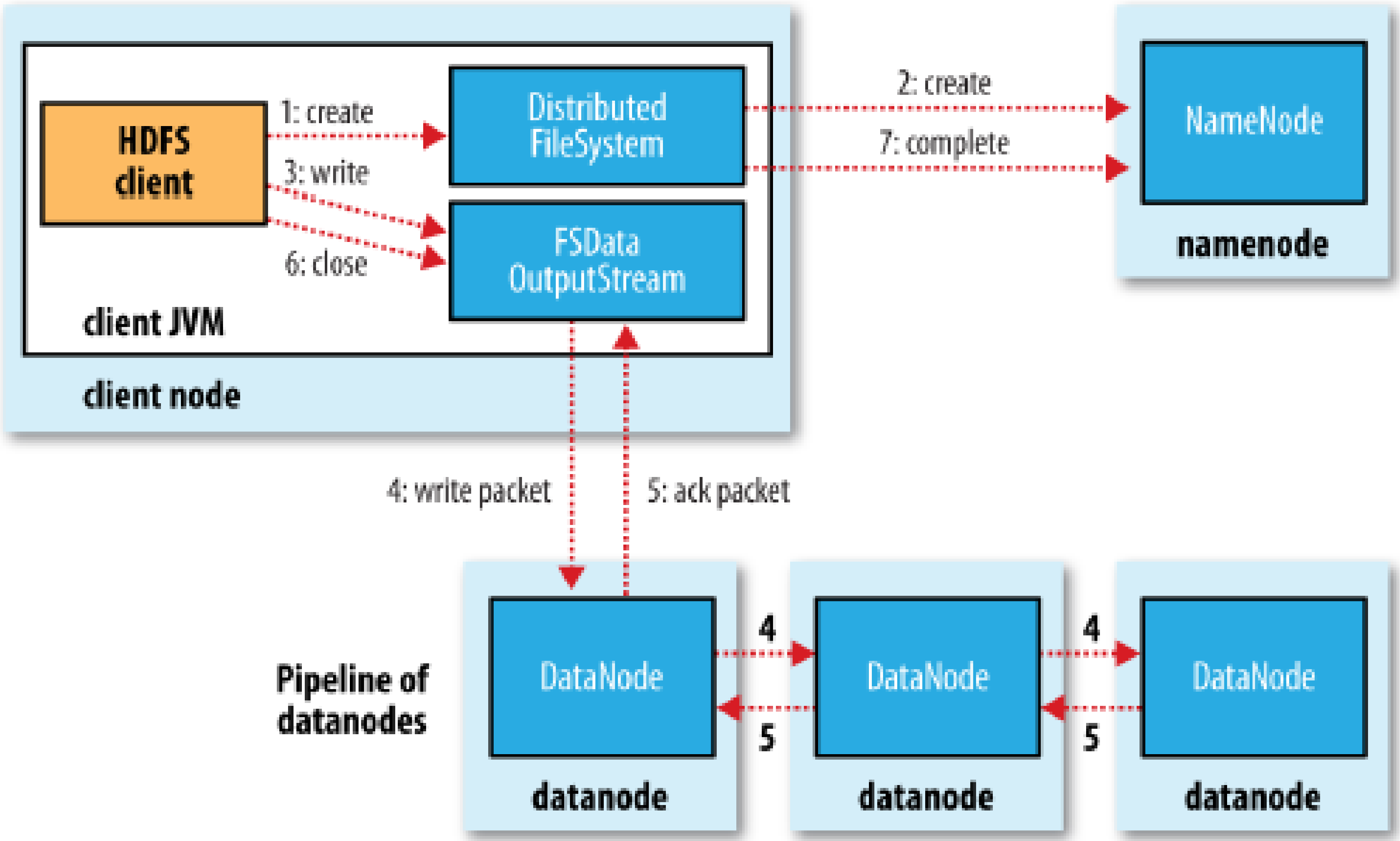
- Example, node *n1* on rack *r1* in data center *d1*.
- Represented as */d1/r1/n1*.
  - *distance(/d1/r1/n1, /d1/r1/n1)* = 0 (processes on the same node)
  - *distance(/d1/r1/n1, /d1/r1/n2)* = 2 (different nodes on the same rack)
  - *distance(/d1/r1/n1, /d1/r2/n3)* = 4 (nodes on different racks in the same data center)
  - *distance(/d1/r1/n1, /d2/r3/n4)* = 6 (nodes in different data centers)



Network distance in Hadoop

Writing a file to HDFS

- Step 1: HDFS client creates file by calling create() on DistributedFileSystem
- Step 2: DistributedFileSystem makes an RPC call to name node to create a file in HDFS namespace
- DistributedFileSystem returns an FSDataOutputStream for the client to start writing data to.
- FSDataOutputStream wraps a DFSOutputStream while returning object
- Step 3: DFSOutputStream splits large datasets into packets
- Data packets are written to internal queue called data queue



client writing data to HDFS

- Data queue is consumed by DataStreamer
- DataStreamer – asks name node to allocate new data blocks by picking list of data nodes
- Step 4: Data nodes forms a pipeline and packets are written in one node after another in the pipeline
  - assume replication level is three - three nodes in the pipeline
- Step 5: After writing data packets the data nodes sends ack packets so that the data block can be removed from data queue
- Step 6: After writing all data the HDFS client closes the data stream by calling close()
- Step 7: the file is complete

Replica Placement

- Placing replica on the same node
- Placing replica on different racks
- Placing replica on different data centers
- Bandwidth?
- Redundancy?

- First replica on same node as the client (if client runs on one of a node in Hadoop cluster). Client outside Hadoop cluster?
- Second replica on node in a different rack
- Third replica same rack & different node as the second replica
- Too many replicas on same rack?

Parallel Copying with Distcp

- Copying data from & to a HDFS
- Data is copied across several HDFS
- Data is copied by parallel processes

  % **hadoop distcp file1 file2**

  % **hadoop distcp dir1 dir2**