Data 603 – Big Data Platforms

Lecture 2
Distributed File Systems

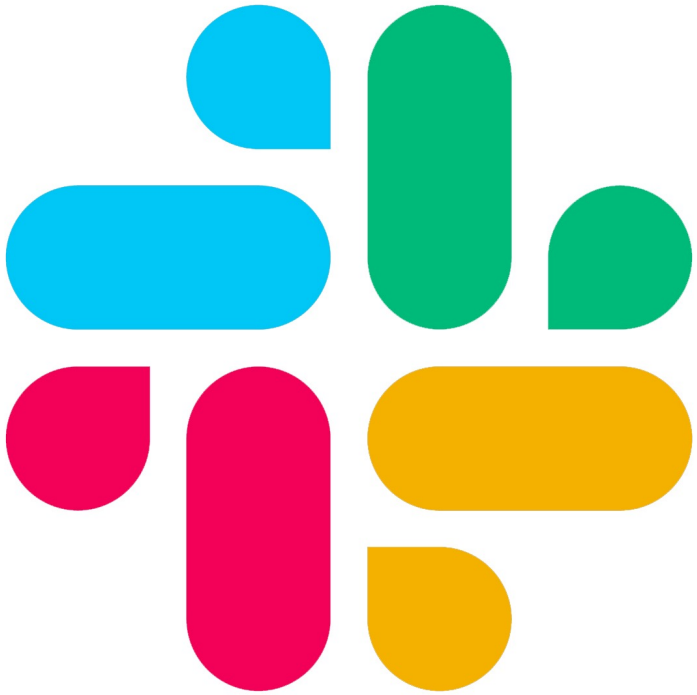
Lecture 2 Outline

- Homework Review
- Slack Channel for the class
- What is Docker
- Installing PySpark in Docker
- File System Concepts
- Distributed File Systems
- Hadoop Distributed File System (HDFS)
- Examples of Hadoop clients: Hbase, Accumulo, Cassandra
- Hadoop Ecosystem
- Project and Paper Topics

Homework Review

- Git / pull requests
- Python/Jupyter approach
- Bash approach

Slack



Slack Channel: data603-sp22-enkeboll
(my user: @enkeboll)

PySpark Local Install

https://vimeo.com/507778388

alternatively, just use Docker:

```
docker pull jupyter/pyspark-notebook
```

Filesystem

Filesystem - Definition

"A filesystem is the methods and data structures that an operating system uses to keep track of files on a disk or partition; that is, the way the files are organized on the disk."

Reference: https://tldp.org/LDP/sag/html/filesystems.html

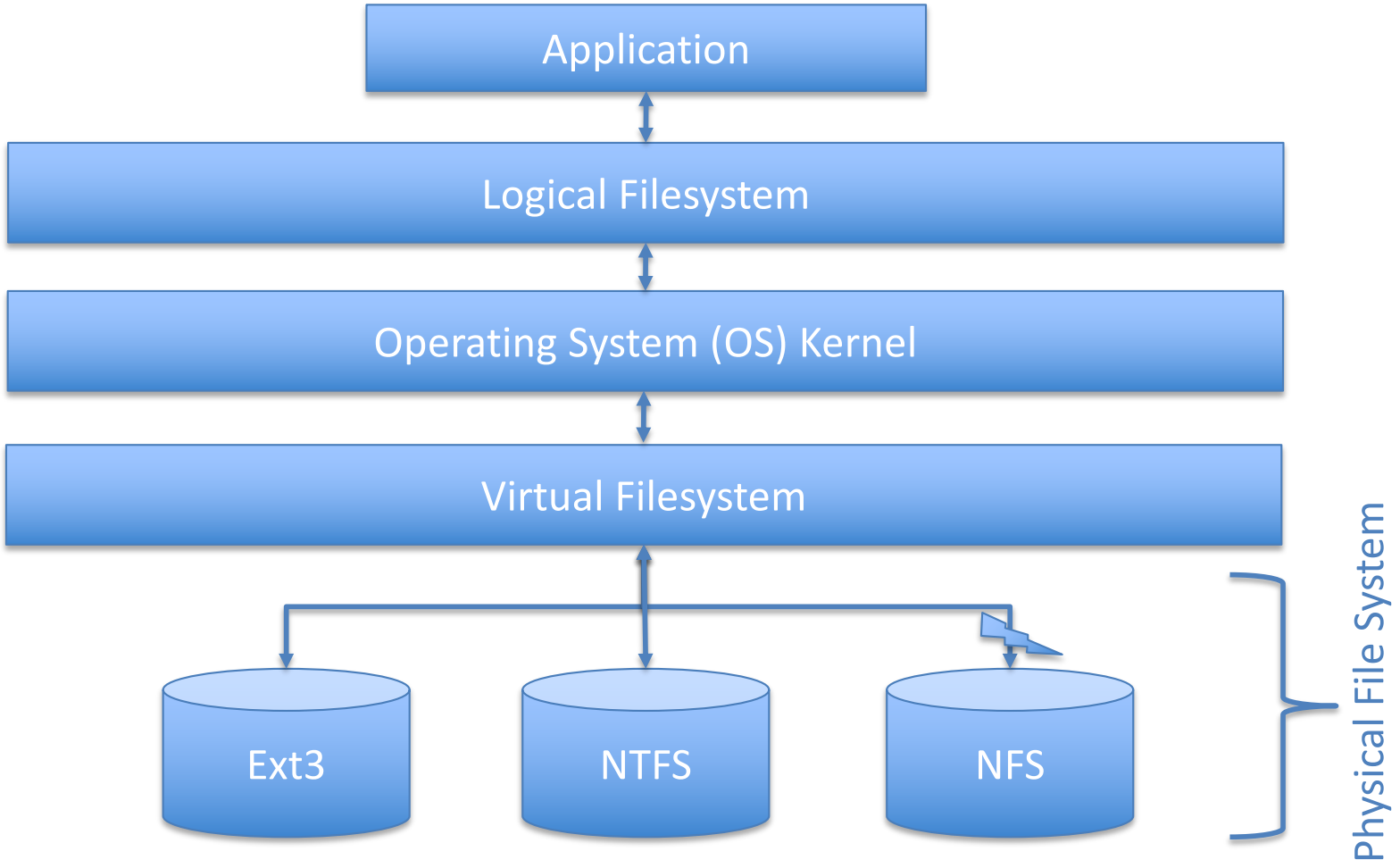
- A subsystem of the operating system that provides protection via access control (security), storage, retrieval and naming of files.
- Controls how data is stored and retrieved
- The structure and logic rules used to manage the groups of data and their names

Reference: https://en.wikipedia.org/wiki/File_system

Filesystem - Architecture

- Logical File System
  - Abstraction layer, provides interaction with user applications via APIs (READ, OPEN, CLOSE)
- Virtual File System
  - Allows client applications to access different types of concrete file systems in a uniform way. E.g. access to local and network storage devices without the client application needing to have different behavior.
- Physical File System
  - Operations of the storage device including reading, writing, buffering, memory management via device drivers

Filesystem


Application
Logical Filesystem
Operating System (OS) Kernel
Virtual Filesystem
Ext3
NTFS
NFS
Physical File System

Filesystem - Concepts

- Space Management
  - Organizing files and directories
  - Keeping physical location within the media where the file objects are located
- File names
  - Human readable name
  - Identify storage location of the files
- Directory Names
  - Logical collection of files
  - Flat or hierarchical structure
- Metadata
  - Length of the data, number of blocks/byte count, time created/modified, user id, group id, access permissions (r/w/x), attributes, etc.

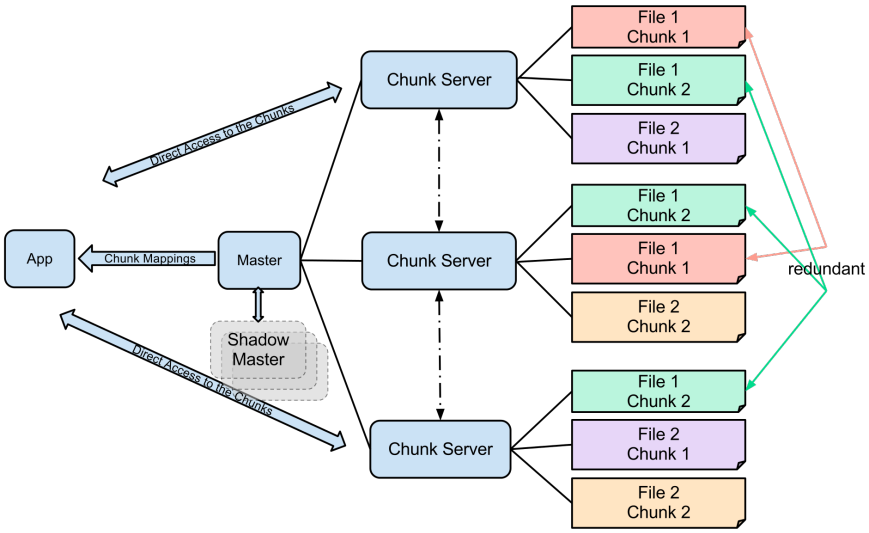
Distributed Filesystems

Distributed Filesystems

- Filesystems that manage the storage across a network
- Do not share block level access to the same storage but use a network protocol.
- Allows files to be accessed using the same interfaces as local files
  - Mounting/unmounting, listing directories, read/write at byte boundaries, system's native permission model

References:

- https://en.wikipedia.org/wiki/Clustered_file_system#Distributed_file_systems
- https://en.wikipedia.org/wiki/Google_File_System



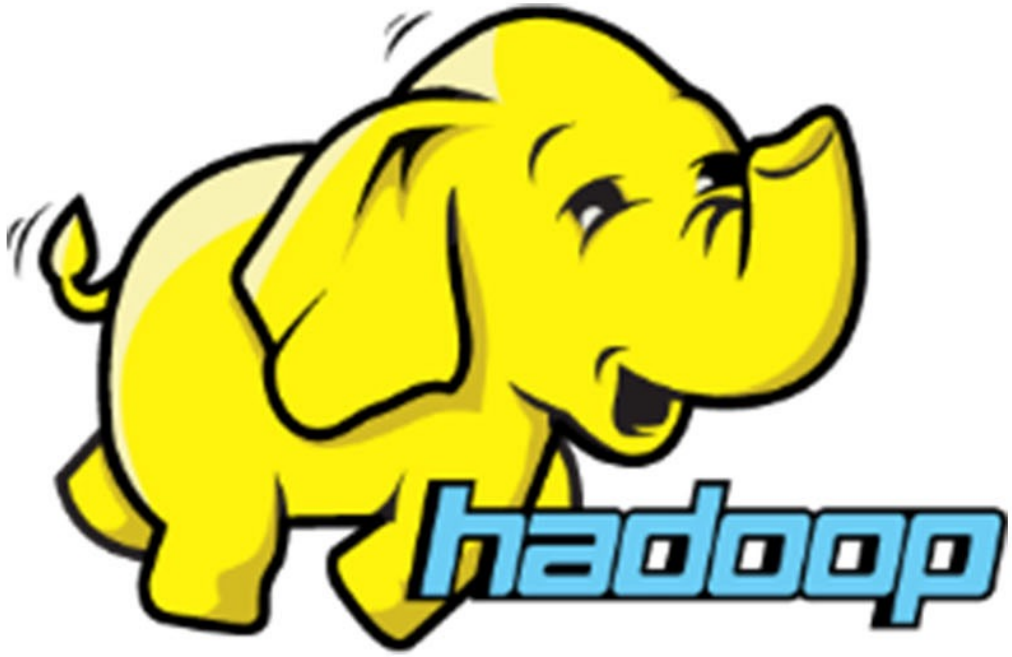
Distributed Filesystems

- **Access Transparency**: Clients can use the distributed filesystems just like they would use the local filesystems
- **Concurrency**: A modification by one process is coherently presented to other processes reading the file.
- **Scalability**: File system should be able serve varying degree of loads incurred (small to large traffic)
- **Replication**: Files are replicated across different nodes for the scalability
- **Fault Tolerant**: Ability to self-heal during hardware failures

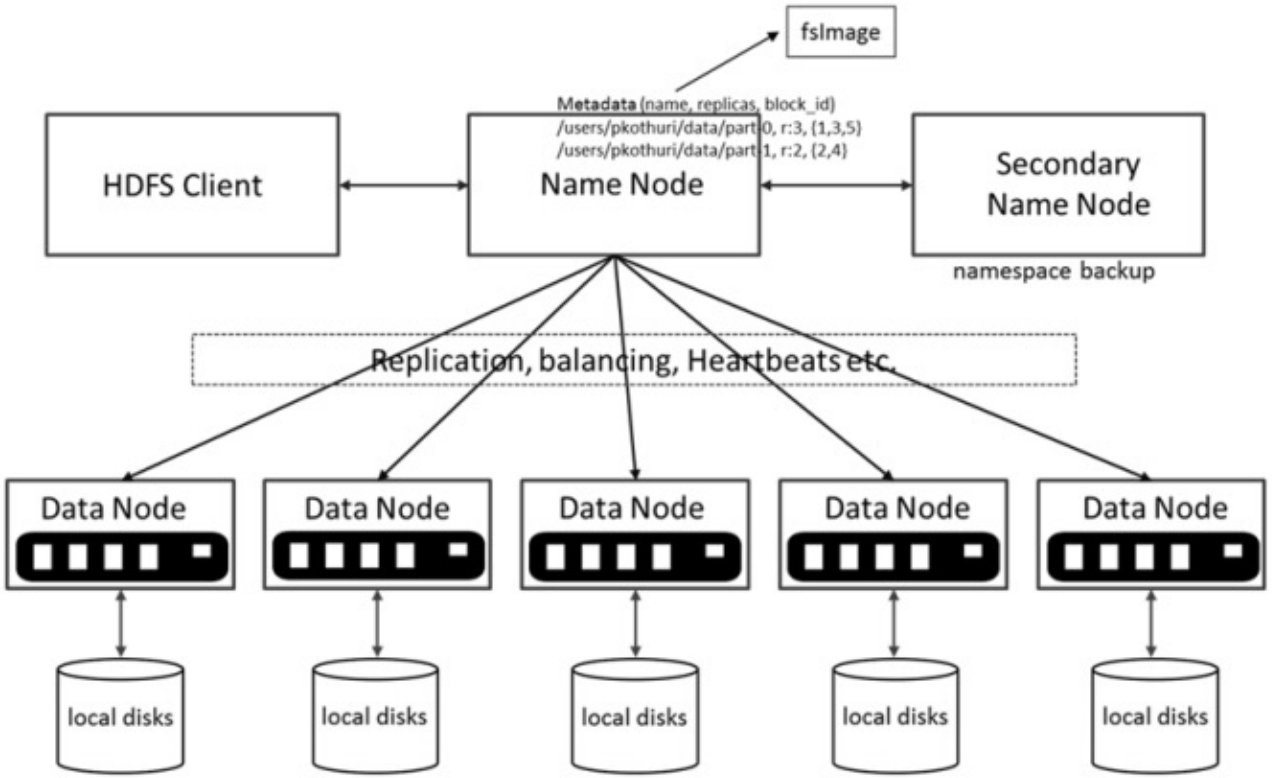
Reference:

- https://en.wikipedia.org/wiki/Clustered_file_system#Distributed_file_systems

Hadoop Distributed File System



Hadoop Distributed File System



Source: https://dzone.com/articles/an-introduction-to-hdfs

Hadoop Distributed File System

- Hadoop Distributed File System (HDFS) is designed to run on **commodity hardware.**
  - Runs on low-cost hardware
- Provides **Fault Tolerance**
  - Quick automated detection and recovery from hardware/software failures.
- **High throughput access** to applications with need for large data sets.
  - **Tuned to support large files** (gigabytes to terabytes in size).
  - High aggregate data bandwidth and scale to hundreds of nodes in a single cluster, supporting millions of files in a single instance.
  - The design **emphasis is on high throughput of data access**.
    - **Not a good choice for low latency data access**

HDFS – Design Choices

Simple Coherency Model

- **Write-once-read-many access model** for files
- A file once created written, and closed need not be changed.
  - Simplifies data coherency issues
  - Enables high throughput data access
- Good use case for MapReduce (topic for lecture 3) applications

Moving Computation, not the data

- Having the data closer to the compute resources to boost the computational efficiency. Read requests are satisfied from a replica (covered later) that is closest to the reader.
- Minimizing the network traffic, and increasing the throughput

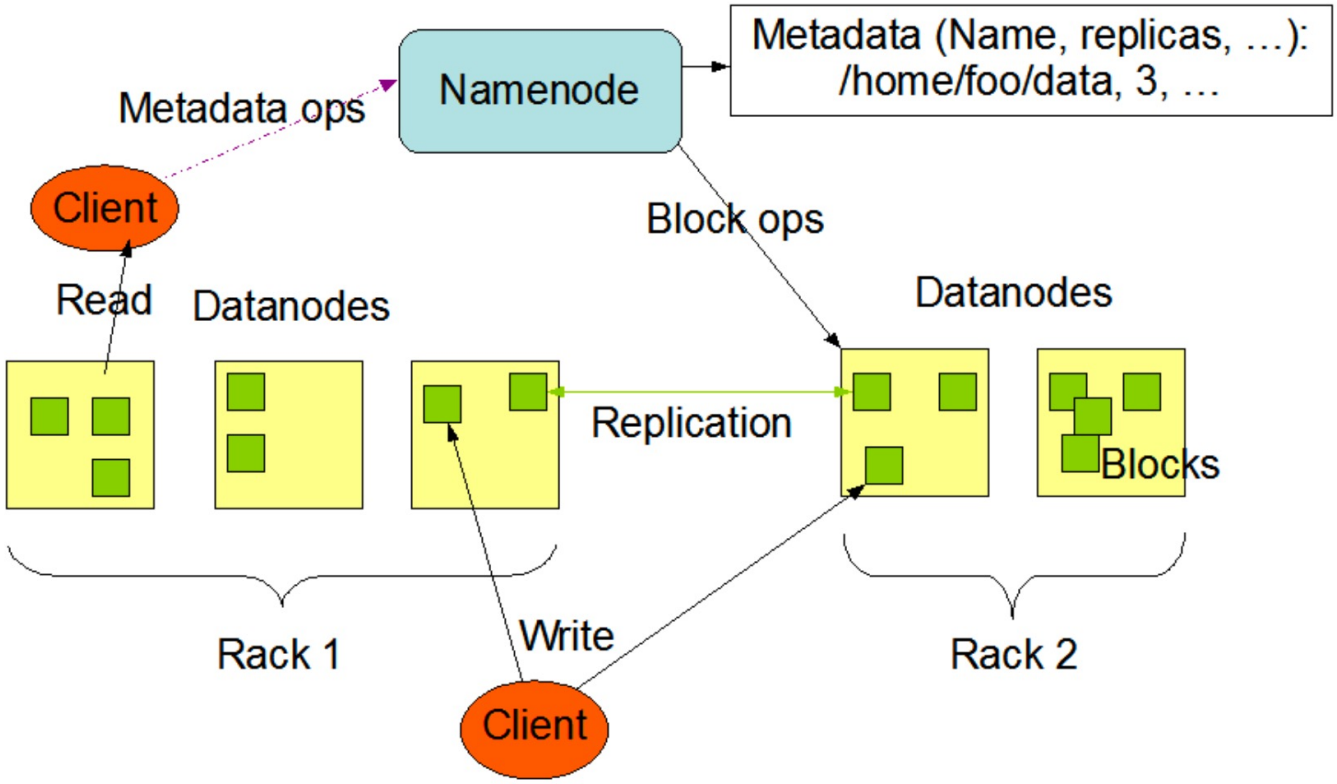
Cross Platform Portability

- Increasing the adaption of HDFS

HDFS – Implementation

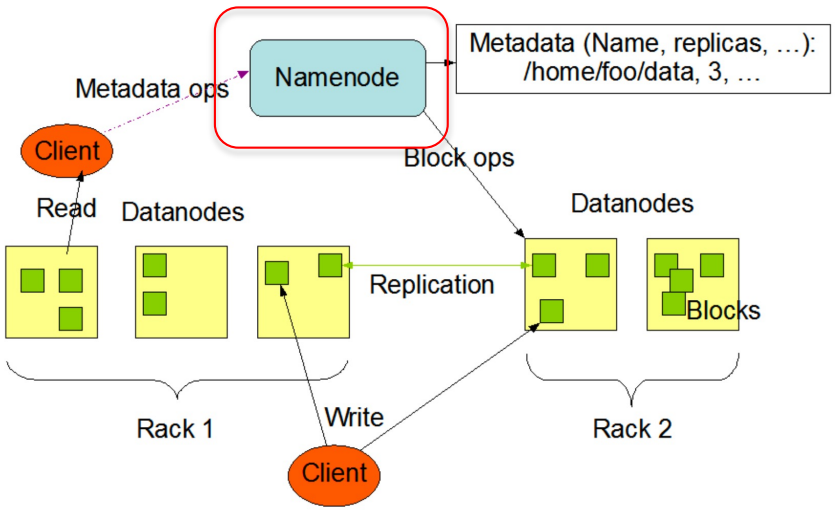
- HDFS is **implemented using Java language** - cross platform portability
- A typical deployment has a dedicated machine running the namenode.
  - Simplifies the architecture
  - **The namenode is the arbitrator and repository for all HDFS metadata.**
  - **User data never flows through the namenode**
- Other instances each run a datanode
- It is possible to run multiple datanodes on one machine, but it is not practical.

HDFS – Architecture



Hadoop employs a leader/follower architecture

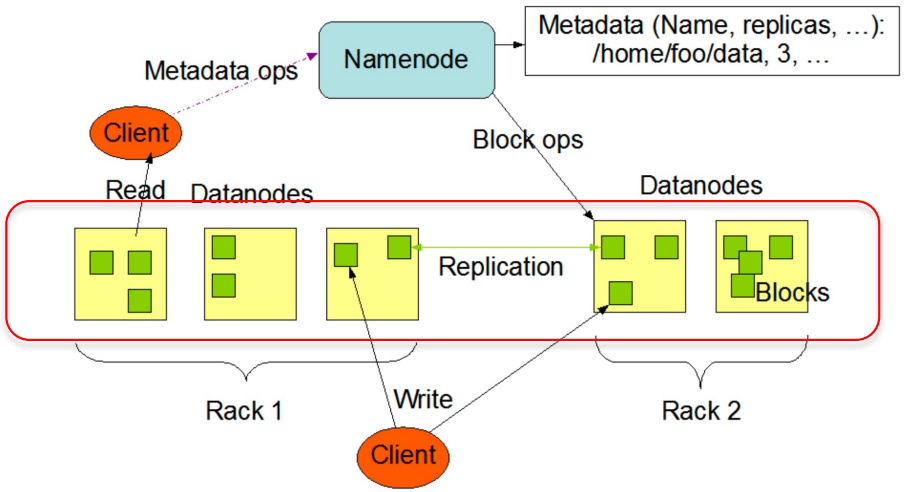
HDFS – Architecture



HDFS cluster consists of a single **namenode:**

- Leader server managing the file system namespace and regulates access to the files by clients.
- Executes file system namespace operations (open, close, rename) on files and directories
- Determines the mapping of blocks to datanodes
- Manages the filesystem tree and the metadata for all the files and directories in the tree.
- The filesystem is not functional without the namenode.

HDFS – Architecture



Datanodes

- Workers of the filesystem.
- Usually one per node in the cluster.
- Manages storage attached to the nodes they run on.
- A file is split into one or more blocks and they are stored in a set of datanodes
- Responsible for serving read and write requests from the file system's clients.
- Perform block creation, deletion, and replication upon instruction from the namenode.

HDFS – Namespace

- **HDFS supports a traditional hierarchical file organization**
  - Tree like directories can be created
  - Other files and directories can be stored in a directory
- HDFS allows the administrator to set quotas for the number of names used (number of file and directory names in the tree rooted at that directory) and the amount of space (a hard limit on the number of bytes used by files in the tree rooted at that directory) used for individual directories.

HDFS – Permissions

- HDFS implements (most of) POSIX model for the file and directory access.
  - Each file and directory is assigned an owner and a group
  - Separate permission for the owner, group users and other users
    - E.g. 755, rwxr-xr-x
    - Note: HDFS files cannot be executed, so 'execute' permissions are ignored.

Reference:

- https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsPermissionsGuide.html

HDFS – Data Replication

- Files are stored as **sequences of blocks**.
- All blocks in a file except the last block are the same size (128 MB by default).
- The blocks of a file are replicated for fault tolerance.
- The block size and replication factor are configurable per file.
  - # of replicas can be specified.
  - The replication factor can be specified at file creation time and can be changed later.
- Files in HDFS are write-once and have one writer at any time
- All decisions regarding the replication of blocks are made by the namenode.
  - *Hearbeats* and *Blockreports* (a list of all blocks on a datanode) are received from each of the datanodes in the cluster.

HDFS – Data Replication (cont.)

- At startup time, each datanode determines the rack ID belongs to and sends its rack ID to the namenode upon registration. Rack IDs are determined using pluggable modules providing rack ID detection APIs.
- HDFS placement policy for the replication factor three
  - One replica on one node in the local rack
  - Another on a different node in the local rack
  - Last on a different node in a different rack
- Goal: reduce the latency caused by network bandwidth between machines in different rack, at the same time provide the high availability (survive node and rack failures)

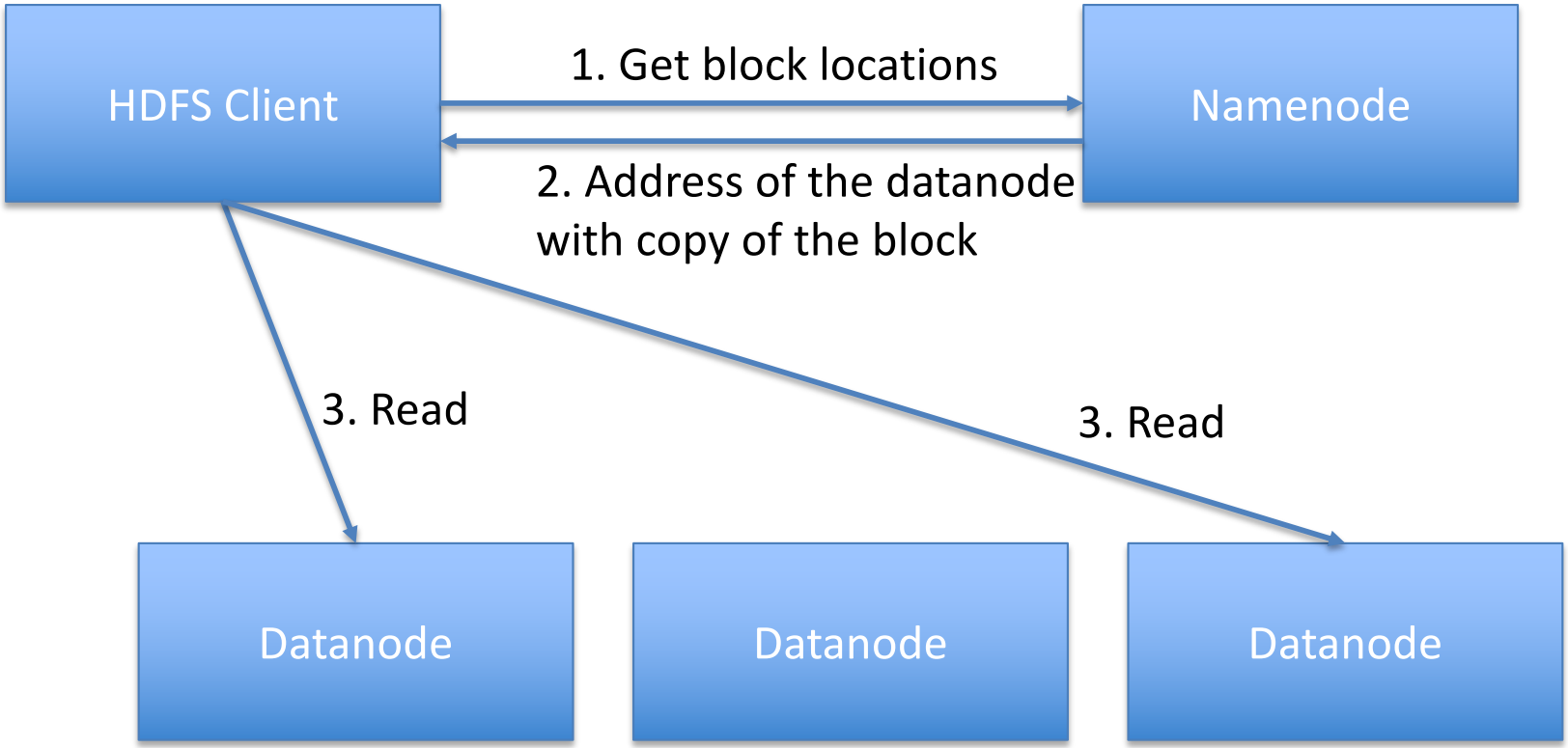
HDFS – Metadata

- HDFS namespace is stored by the namenode.
- A transaction log called the *EditLog* is used to persist the records of each changes to the file system *metadata*.
  - *EditLog* is stored within the namenode's local host OS file system.
  - The entire file system namespace, including the mapping of blocks to files and file system properties, is stored in a file called the *FsImage*.
- The namenode keeps an image of the entire file system namespace and file *Blockmap* in memory.
  - Since the namenode holds filesystem metadata in memory, the number of files in a filesystem is limited by the amount of memory on the namenode; not good for a lot of small files.
- HDFS federation (since 2.x release) allows adding more namenodes, each to manage a portion of the filesystem namespace

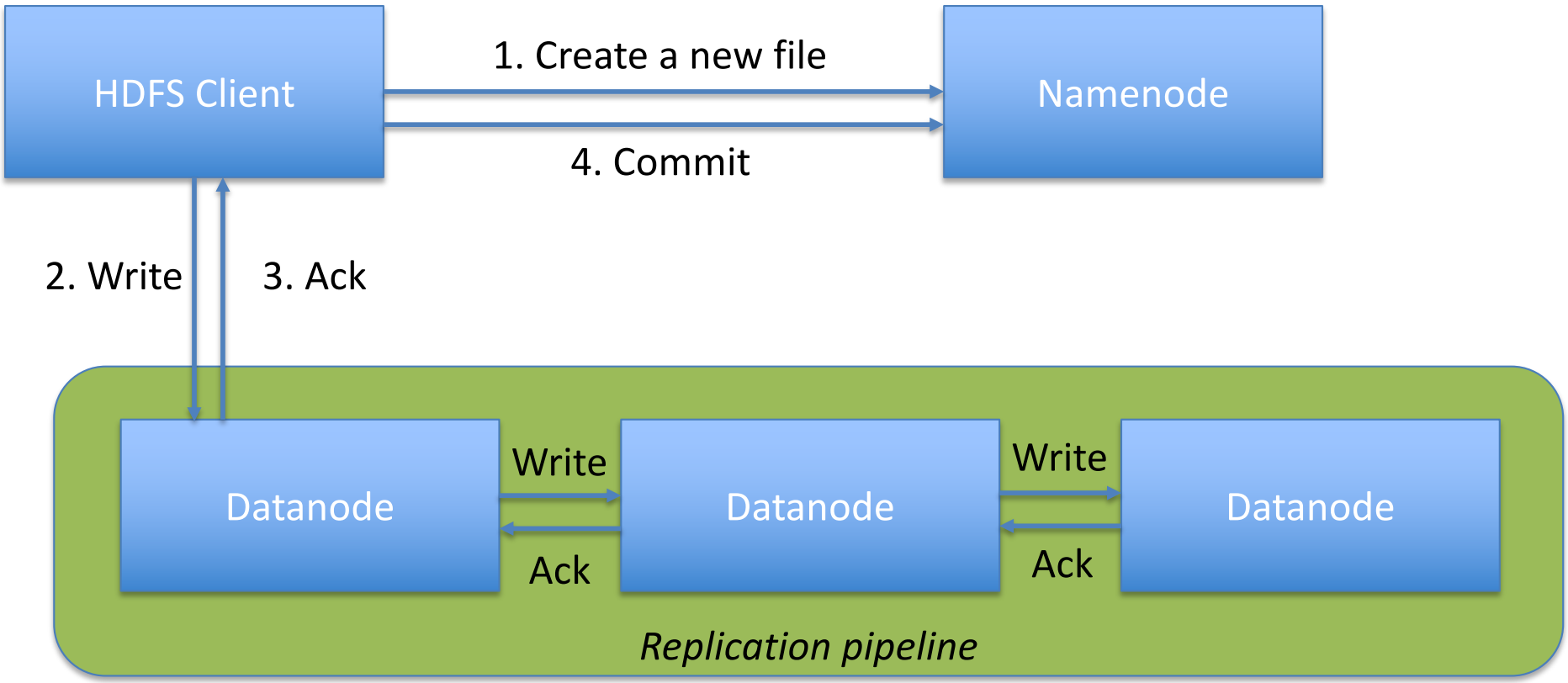
HDFS – Data Storage

- HDFS data is stored in each datanode's local file system.
  - The datanode has no knowledge about HDFS files.
- Each block of HDFS data is stored in a separate file in its local file system.
- A heuristic is used to determine the optimal number of files per directories and their subdirectories
- During the startup of a datanode, it scans through its local file system and generates a list of all HDFS data blocks corresponding to each of the local files and sends this report (*Blockreport*) to the namenode

HDFS Client File Read



HDFS Client File Write



HDFS Commands

https://images.linoxide.com/hadoop-hdfs-commands-cheatsheet.pdf

Hadoop Demo

- https://medium.com/analytics-vidhya/hadoop-single-node-cluster-on-docker-e88c3d09a256

Discussion

HDFS is great, but it is complex to maintain.

Are there other alternatives?

More reading

- https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html
- Ceph as a scalable alternative to the Hadoop Distributed File System: https://www.usenix.org/system/files/login/articles/73508-maltzahn.pdf
- https://databricks.com/blog/2017/05/31/top-5-reasons-for-choosing-s3-over-hdfs.html
- Hadoop Ecosystem Table: http://hadoopecosystemtable.github.io/

Pop Quiz

Q1: HDFS consists of a single _____

Pop Quiz

Q2: One of the attributes of distributed file system is an ability to self-heal during hardware failures.

What is it called? F_____ T_____

Pop Quiz

Q3: True or False

Every block of data in HDFS flows through the name node.

Concepts

- Structured Data
  - Data with structures: shapes, columns with size and types, rows
- Semi-Structured Data
  - *"Semi-structured data is a form of structured data that does not obey the tabular structure of data models associated with relational databases or other forms of data tables, but nonetheless contains tags or other markers to separate semantic elements and enforce hierarchies of records and fields within the data. Therefore, it is also known as self-describing structure"*
- Key/Value Store
  - *"A key-value database is a type of nonrelational database that uses a simple key-value method to store data. A key-value database stores data as a collection of key-value pairs in which a key serves as a unique identifier. Both keys and values can be anything, ranging from simple objects to complex compound objects. Key-value databases are highly partitionable and allow horizontal scaling at scales that other types of databases cannot achieve"*

Concepts

- Column Oriented
  - *"While a relational database is optimized for storing rows of data, typically for transactional applications, a columnar database is optimized for fast retrieval of columns of data, typically in analytical applications. Column-oriented storage for database tables is an important factor in analytic query performance because it drastically reduces the overall disk I/O requirements and reduces the amount of data you need to load from disk. Like other NoSQL databases, column-oriented databases are designed to scale "out" using distributed clusters of low-cost hardware to increase throughput, making them ideal for data warehousing and Big Data processing."*
  - *Also read, https://www.geeksforgeeks.org/difference-between-row-oriented-and-column-oriented-data-stores-in-dbms/, https://www.stitchdata.com/columnardatabase/*
- Random Access
  - *"Ability to access an arbitrary element of a sequence in equal time or any datum from a population of addressable elements roughly as easily and efficiently as any other, no matter how many elements may be in the set. In computer science it is typically contrasted to sequential access which requires data to be retrieved in the order it was stored."*
- Sparse
  - Homework: Do a research on what it means to have sparse data/datastore/database.

HBase

- Distributed key/value store, column-oriented database built on top of HDFS.
  - It promises strong consistency.
- Used for real-time read/write and random access to large datasets.
  - It is able to host large tables on clusters made from commodity hardware
- A sparse, distributed, persistent, multidimensional sorted map, indexed by row key, column key, and timestamp
- Built to scale linearly by adding more nodes.
- Not relational – no support for SQL
- Stores structured and semi-structured data naturally
  - Allows for dynamic and flexible data model
- Modeled after Google's Bigtable

Apache Accumulo

- Distributed key/value store.
  - Keys are sorted at all times.
- Created by NSA
- Based on Google's BigTable design
- Built on top of Apache Hadoop, Zookeeper, and Thrift.
  - Accumulo depends on Apache Hadoop for storage and Apache ZooKeeper for configuration.
  - Thrift proxy allows Accumulo API to non-JVM-based languages.

Apache Cassandra

- Highly available, distributed, partitioned row store NoSQL database.
- Built to deal with inevitable node failures following AWS Dynamo DB's design principles.
  - Replication Factors (RF) and required consistency level provides Cassandra clusters high availability even during nodes failures.
- Distributed database – A Cassandra cluster is a collection of nodes working together to serve the same dataset
  - Nodes can be grouped into logical data centers providing data locality an application.
  - Adding more nodes also add more disk footprint and the operational throughput.
- Partitioned Row Store – Rows of data are stored in tables based on the hashed value of the partition key (token). Each node in the cluster is assigned multiple token ranges.
  - The Replication Factor decides how many copies of each row will be stored in each data center
- Decentralized – every node in the cluster is identical. No single point of failure.

Hadoop Ecosystem

https://hadoopecosystemtable.github.io/

Paper Topic Examples

- *Compare three popular NoSQL distributed databases. Do a thorough analysis of each database and do a comparison between them.*
  - Good use case?
  - Strengths?
  - Weakness?
  - Data Modeling
  - What is unique about the particular DB?
  - Operations & maintenance?
  - Scalability, high availability, recovery during failure, etc.
- *The effect of [popular compression algorithm] on Hadoop data lakes*
  - Description of algorithm & competing algorithms
  - Storage before/after compression
  - Query performance before/after compression
  - Non-performance downsides to compression

Project Topic Examples

- *Data sources:*
  - *Kaggle Competitions*
  - *Google Dataset Search*
  - *OpenData (Baltimore, DC, NYC, etc)*
  - *Self-acquired (Reddit, Yelp, Wikipedia)*
- *Project scopes:*
  - *MUST INVOLVE SPARK!*
  - *Should involve one other technical*

Assignment

- Read the following articles:
  - https://www.digitalocean.com/community/tutorials/a-comparison-of-nosql-database-management-systems-and-models
  - https://databricks.com/glossary/data-lake
- Write a summary, no longer than ~250 words, of the technologies and practices described in the two articles
- The summary should be written in **Markdown**
  - Include a formatted title and at least two section headers
  - Embed at least one image
  - Cite the two articles at the end, using hyperlinks (nothing fancy needed, just a bullet or similar with the link)

Assignment Submission

- Same as last week
- Add your file **in a new** branch to your github repository as **homework/hw02.md**
- Open a pull request from this branch, tag me as the reviewer
- Submit the pull request URL (eg github.com/enkeboll/data603/pull/2) on Blackboard Assignments

Next Week

- MapReduce Design Patterns
- Introduction to Cloud Computing