Distributed File Systems

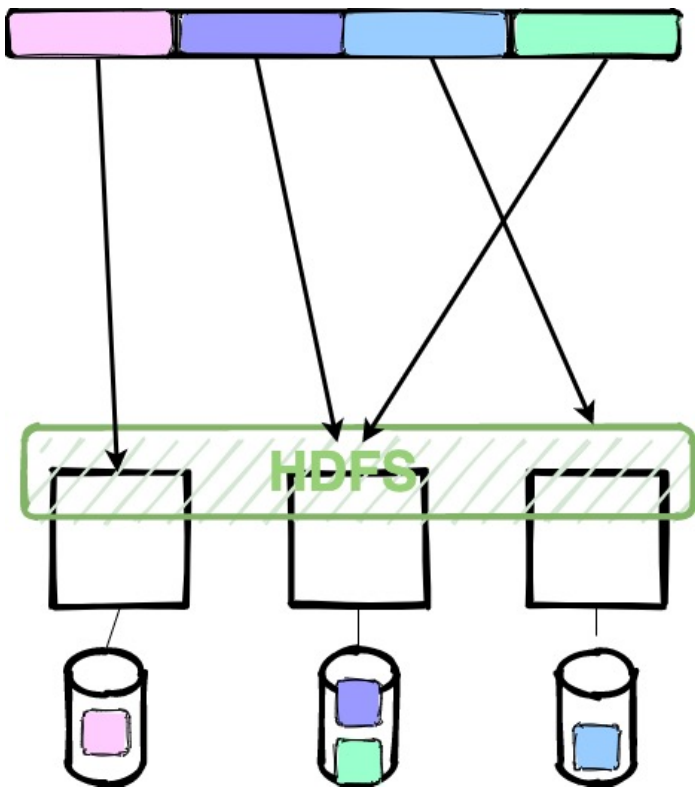
Case study : The Hadoop File System

Principles

- Open source implementation of the Google File System
- Distributed filesystem over multiple nodes
- Requirements
  - Fault tolerance
  - Scalability
  - Optimized for batch operations
    - Throughput more important than latency
    - Appending to files, no overwriting
- Provides a separate and global filesystem
  - Unix like paths
  - E.g. /home on HDFS is not /home of the machine
- Not part of the OS, added software
  - Written in Java, works on any machine with JVM support
  - Shipped with script for users and administrator

Architecture

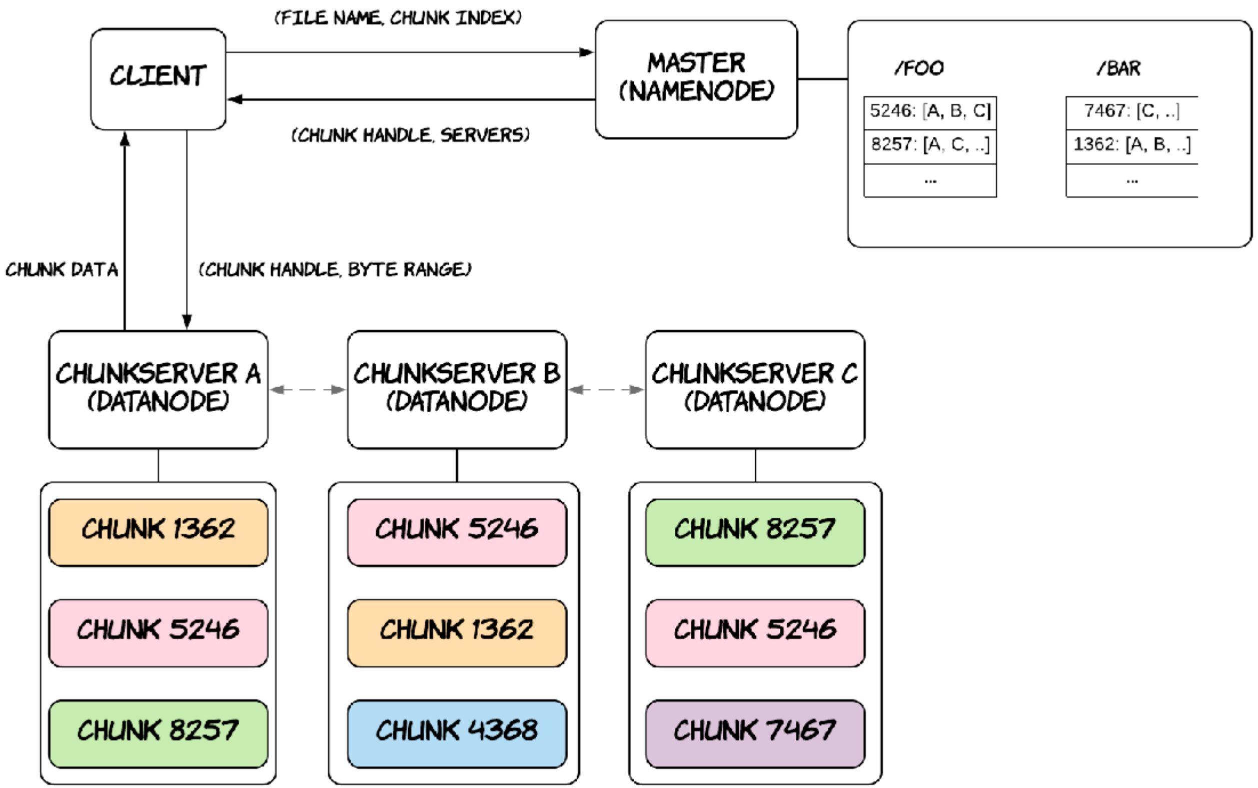
- Data and metadata are separated
  - Data are stored in DataNodes
  - Metadata are stored in NameNodes
- Data are divided into blocks
  - Default value 64MB or 128MB
- Blocks are stored on different machines



Architecture

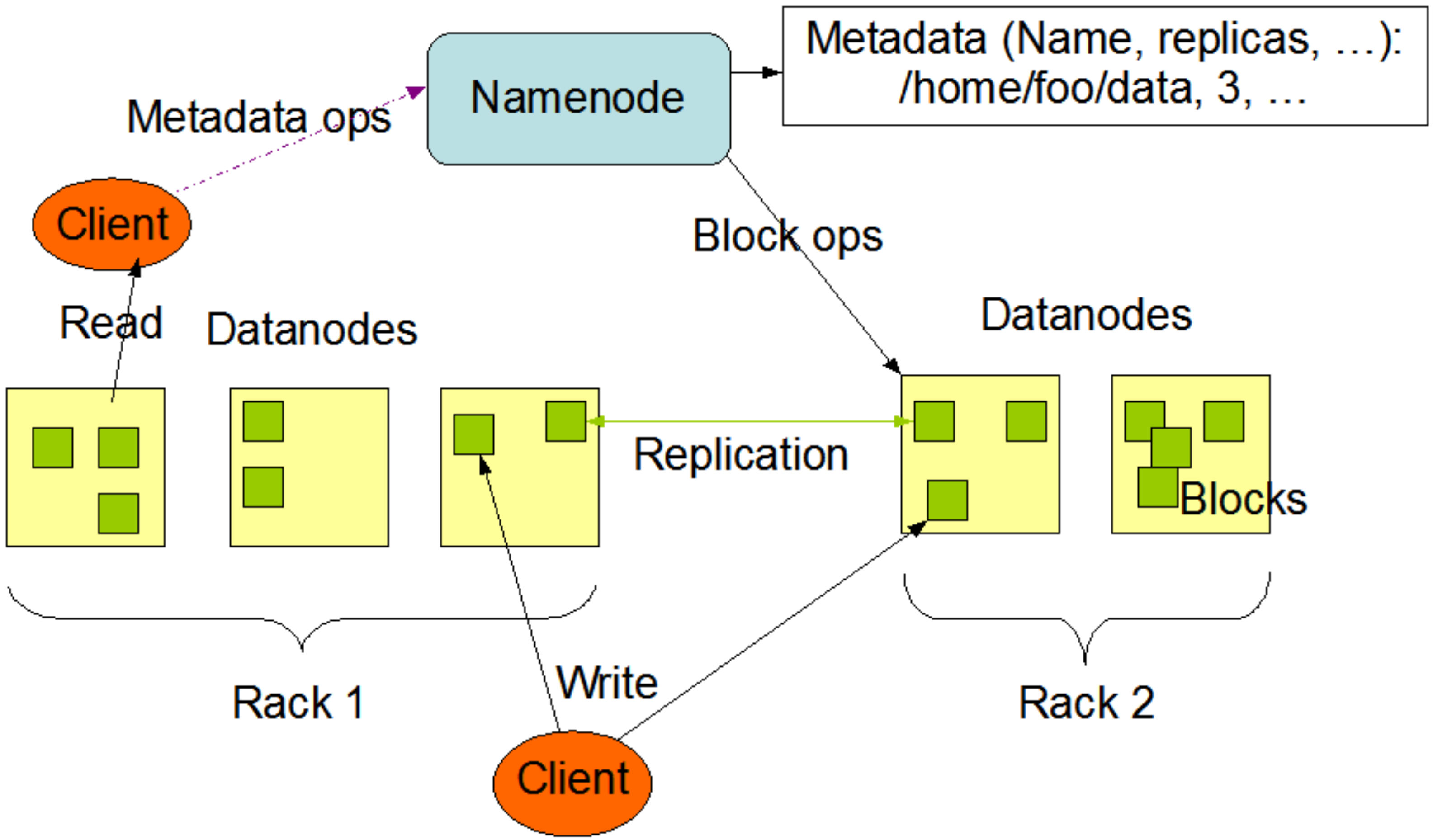
- Data can be replicated
  - Replication factor (default 3)
  - More costly (space) than RAID or erasure codes
- Not all operations are supported
  - No random write, only append and truncate
- Permission
  - User and group for coarse grained control
    - Set access to owner or group of users
  - Access Control Lists (ACLs) for finer control
    - Set specific permission to specific user
    - Disabled by default

GFS (HDFS) Architecture



Source: *Distributed Systems for Practitioners, Dimos Raptis*

HDFS Architecture



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

Reading & writing

- All clients read and write requests go through NameNodes
  - Validation and metadata
- Actual data go directly to clients
  - Faster access
  - Blocks can be requested in parallel
- Reading workflow
  - Client send request for file “/test.txt”
  - NameNode checks access rights and returns list of blocks+DataNodes

Reading & writing

- Writing workflow
  - Client contact NameNode
  - If writing allowed, check if file exists
    - If yes, error
  - NameNode returns a list of DataNode
  - Client sends data to DataNodes in round-robin
  - After writing all blocks, the client notifies the NameNode
- Replication is handled by DataNodes while receiving data

HDFS Blocks

- Large blocks
  - Not suited for storing small files
  - Limit overhead of metadata
- Transfer cost of a block
  - Disk access + latency + throughput
  - Large blocks minimize impact of disk access and latency
- Replication for
  - Fault tolerance
    - Tries to put replica on different machine and different racks
  - Faster access
    - Load blocks from node closer to client
- Also support Erasure Codes

Fault tolerance

- NameNode is a single point of failure
  - If down, cannot access data anymore
  - If destroyed (metadata lost), data are lost
- Possibility to use a Secondary NameNode
  - Maintains snapshot of metadata + edit log
  - Periodically apply edit log to metadata and store new state
- In case of crash
  - Restart NameNode
  - Get snapshot + log of Secondary NameNode and rebuilt recent state
- But
  - Rebuilding might be long
  - Might still lose some metadata

Fault tolerance

- *High Availability*
  - Feature introduced in Hadoop 2
- Support 2 NameNode
  - 1 active and 1 passive in standby
  - If active fails, standby takes its place
- How to ensure consistency?
  - Relies on JournalNode
  - Active write edit log to JournalNode
  - Passive regularly get edit log from JournalNode
- Protection against failures of JournalNode
  - Usually use 3 of them but N possible
  - Agreement based on a Quorum algorithm
  - Can tolerate (N-1)/2 failures

Commands

- All commands are done using bin/hdfs or bin/hadoop
  - hdfs <command> [options]
  - hadoop <command> [options]
- 3 types : client, admin and daemon
- Client commands : use the filesystem
  - *hdfs dfs <args>* or *hadoop fs <args>* : run the command args on the HDFS filesystem
  - Examples : -mkdir, -put, -copyfromlocal, -get, -copytolocal, -rmr

Commands

- Admin commands : manage the filesystem
  - hdfs fsck : check and repair the filesystem
- Daemon commands : manage the infrastructure
  - hdfs balancer : run the balancer utility
  - hdfs datanode : start a new datanode
  - hdfs namenode : start a new namenode