Concepts and Models of Parallel and Data-centric Programming

MapReduce – Hadoop Distributed File System

Lecture, Summer 2020

Simon Schwitanski
Dr. Christian Terboven

Outline

0. Organization
1. Foundations
2. Shared Memory
3. GPU Programming
4. Bulk-Synchronous Parallelism
5. Message Passing
6. Distributed Shared Memory
7. Parallel Algorithms
8. Parallel I/O
9. **MapReduce**
10. Apache Spark

a. MapReduce Programming Model
b. Parallelizing MapReduce
c. Hadoop Ecosystem
d. Hadoop Distributed File System
e. Yet Another Resource Negotiator
f. Comparison to Other Approaches
g. MapReduce Design Patterns

Distributed File System

- Datasets to analyze much larger than storage capacity of single machine
- Idea: Partition data across several machines
- *Distributed File System*: Managing files across network of machines
- Hadoop Distributed File System (HDFS)
  - De-facto standard storage system in Apache Hadoop

HDFS Design Goals (1)

Large Datasets

- Applications running on HDFS have large datasets
- Typical file size: Gigabytes to terabytes

Streaming Data Access

- Write-once-read-many access pattern
- Dataset generated or copied once, different analyses on same dataset
- Time to read complete dataset more important than low latency access

Commodity Hardware

- No special purpose hardware (interconnect, CPU, …) required
- “Cheap” components

HDFS Design Goals (2)

Data Locality

- “Moving computation is cheaper than moving data”
- Especially for huge data sets
- Perform computation (physically) close to data
- Minimizes network congestion, increases throughput

Fault Tolerance

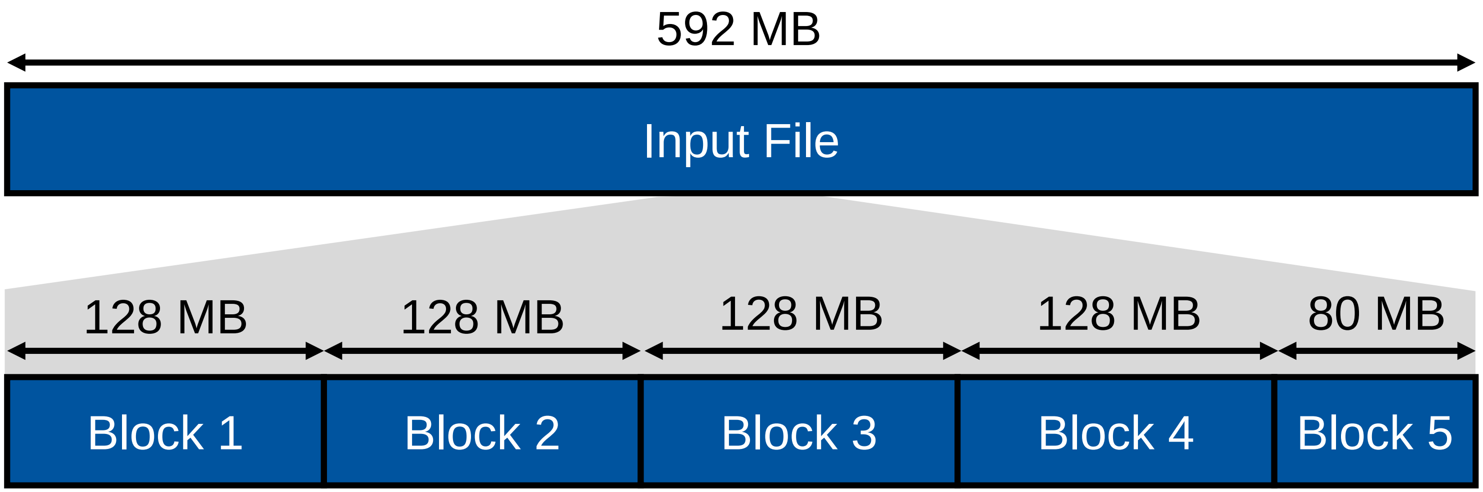
- 100 – 10,000 machines, each storing part of the data
- Probability of failures high
- Quick detection and recovery of faults

Portability

- Support of heterogeneous hardware and software platforms

HDFS Blocks

- HDFS: Logical abstraction of underlying physical file system (ext4, …)
- Each file split into equally-sized chunks called *blocks*
- Blocks treated as independent units in file system
- Block size larger (128 MB) than in physical file systems (1 – 8 KB)
  – File smaller than single block does not occupy full block

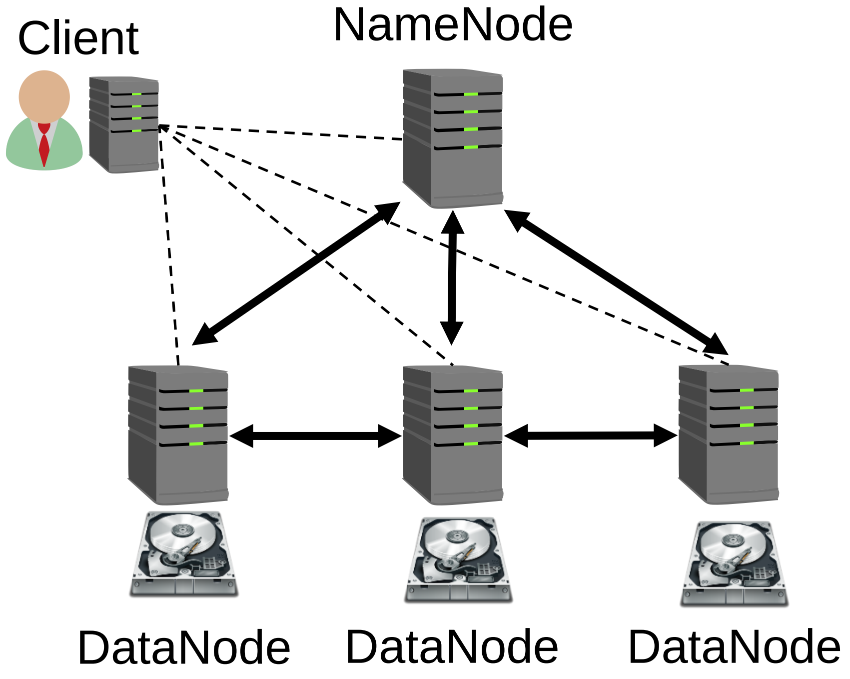


Benefits of Block Abstraction

- Files can be larger than storage capacity of single machine
  - Store blocks of file on different machines
- Simplified storage management (mostly equally-sized blocks)
- Fault tolerance easier to implement
  - Replicate blocks on different machines
  - Lost block (machine failure, corruption) recovered using replication
  - *Replication factor*: Number of replicated blocks (default: 3)

NameNode and DataNode

- Master / Worker architecture
- NameNode: (Single) Master
  - Manages file system namespace and metadata (e.g., timestamp, length)
  - Provides access to clients
- DataNode: Worker
  - Manages local storage attached to node (as HDFS blocks)
  - Periodically sends heartbeats to NameNode with list of stored blocks
- Client accesses data by communicating with NameNode and DataNodes



HDFS Architecture

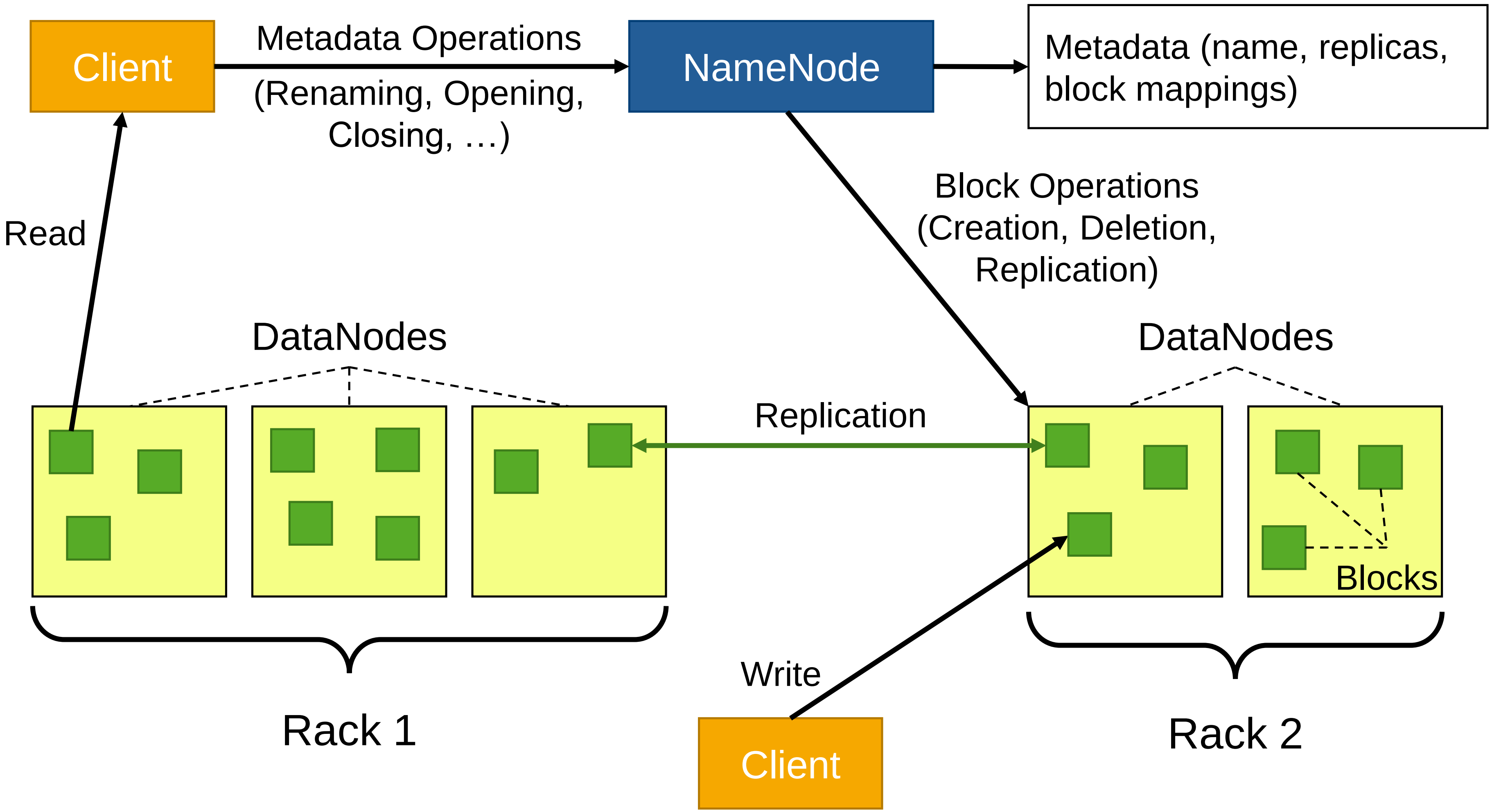
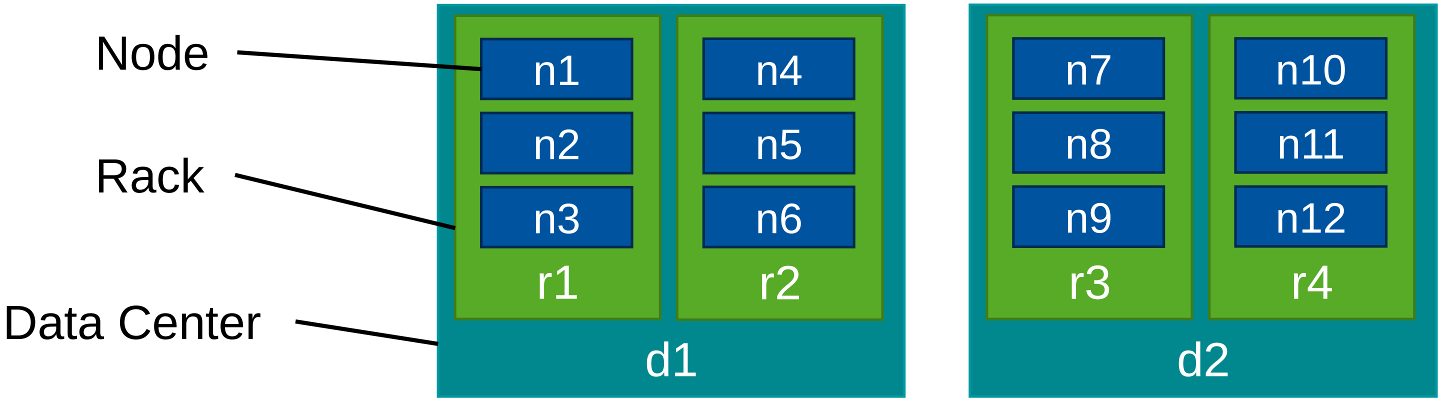


Illustration adapted from https://hadoop.apache.org/docs/r3.1.0/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

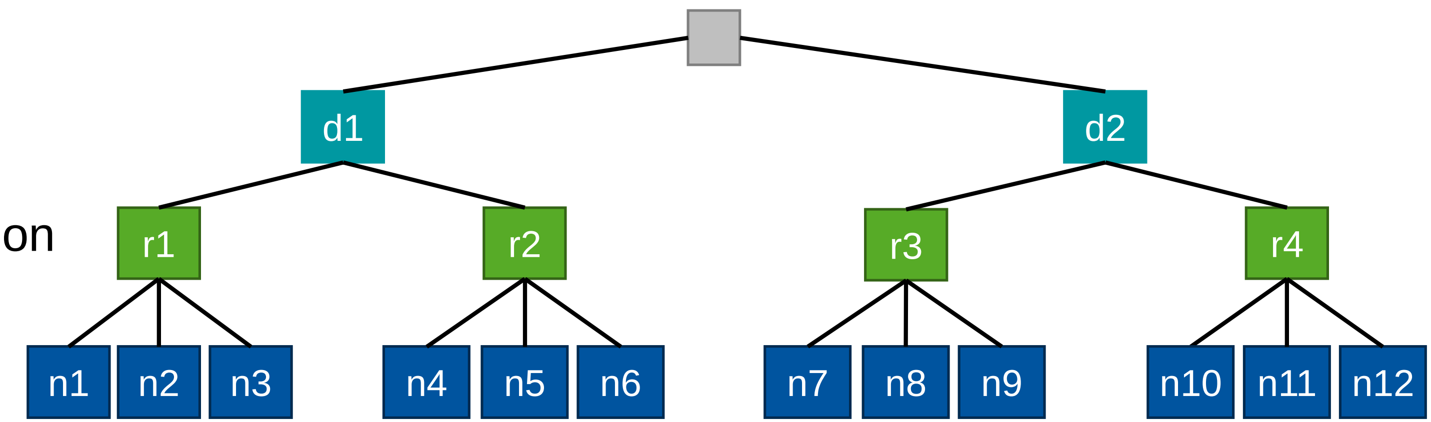
Network Topology in Hadoop

- DataNodes with requested blocks should be “close” to client node
  - Client node: Node that requests data blocks (for computation)
- Network bandwidth: Limiting factor when processing huge datasets
- Possible distance measure for two nodes: Bandwidth
  - Problem 1: Measuring bandwidth in a cluster is difficult
  - Problem 2: Number of pairs is in $O(n^2)$ for n nodes
- Distance measure in Hadoop
  - Network represented as tree
  - Distance between two nodes: Minimum distance between nodes in tree
  - DataNode with smallest distance to client node is chosen
  - Hadoop has to be made aware of network structure

Network Topology – Example



Tree Representation

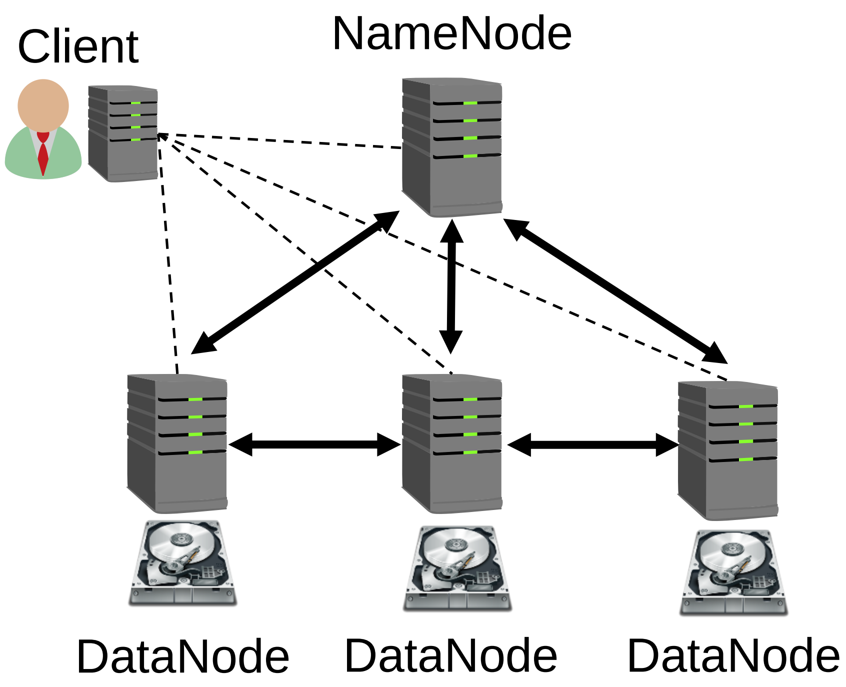


$distance(n1, n1) = 0$ (same node)
$distance(n1, n2) = 2$ (same rack)

$distance(n1, n4) = 4$ (different racks, same data center)
$distance(n1, n7) = 6$ (different data centers)

HDFS Architecture – Revisited

- NameNode is arbitrator for **all** metadata
  - Client asks NameNode before reading from or writing to any DataNode
  - User data written **directly** to DataNode (avoid overload of NameNode)
- Any problems with this approach?

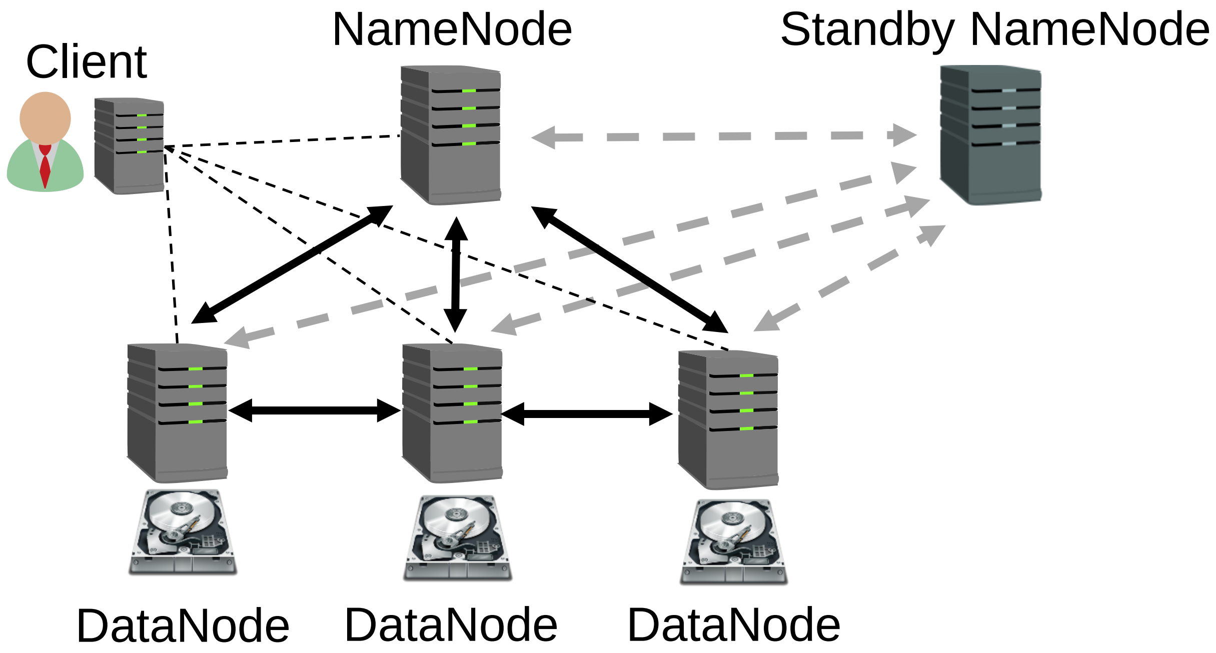


NameNode Failure

- NameNode is ***Single Point of Failure (SPOF)***
- If machine running NameNode fails
  - All data lost, no way of reconstruction
- **Solution:** Backup all relevant data on external file systems
- On failure: Restore file system state from backup data
  - Read file system image backup
  - Apply (potentially buffered) metadata modifications
  - Wait for DataNodes reporting blocks (block mapping lost on NameNode failure)
- **However:** SPOF problem remains
  - Recovery can take about 30 minutes depending on data size

HDFS High Availability (Hadoop 2.0 onwards)

- Add a standby NameNode (active-standby configuration)
- On failure of primary node: Standby node takes over



DataNode Failure

- DataNode periodically sends heartbeats to NameNode
  - Contains report of stored blocks
- DataNode failure detection: Absence of heartbeat messages
- Default timeout: 10 minutes
- NameNode marks DataNode without heartbeats as dead
  - No further requests sent to this node
  - If replication factor below specified value: Re-replication on other DataNode
- Not covered in this lecture: Data integrity