Modern Database Systems

HDFS

Doc. RNDr. Irena Holubova, Ph.D.
Irena.Holubova@matfyz.cuni.cz

Big Data Related Technologies

- Distributed file systems
  - e.g., HDFS
- Distributed databases
  - Primarily NoSQL databases
  - And many other types
- Cloud computing
- Data analytics
  - Batch
  - Real-time
  - Stream
- …

Apache Hadoop

- Open-source software framework
- Running of applications on large clusters of commodity hardware
  - Multi-terabyte data-sets
  - Thousands of nodes
- Derived from Google's MapReduce and Google File System (GFS)
  - Not open-source

http://hadoop.apache.org/

Apache Hadoop
Modules

- Hadoop Common
  - Common utilities
  - Support for other Hadoop modules
- Hadoop Distributed File System (HDFS)
  - Distributed file system
  - High-throughput access to application data
- Hadoop YARN
  - Framework for job scheduling and cluster resource management
- Hadoop MapReduce
  - System for parallel processing of large data sets

Apache Hadoop
Hadoop-related Projects

- Avro – a data serialization system
- Cassandra – a scalable multi-master database with no single points of failure
- Chukwa – a data collection system for managing large distributed systems
- HBase – a scalable, distributed column-family database that supports structured data storage for large tables
- Hive – data warehouse infrastructure that provides data summarization and ad hoc querying
- Mahout – scalable machine learning and data mining library
- Pig – high-level data-flow language and execution framework for parallel computation
- ZooKeeper – high-performance coordination service for distributed applications

HDFS (Hadoop Distributed File System)
Basic Features

- Free and open source
- Crossplatform
  - Pure Java
  - Has bindings for non-Java programming languages
- Fault-tolerant
- Highly scalable

HDFS
Fault Tolerance

- Idea: “failure is the norm rather than exception”
  - A HDFS instance may consist of thousands of machines
    - Each storing a part of the file system’s data
  - Each component has non-trivial probability of failure

→ Assumption: “There is always some component that is non-functional.”
  - Detection of faults
  - Quick, automatic recovery

HDFS

Data Characteristics

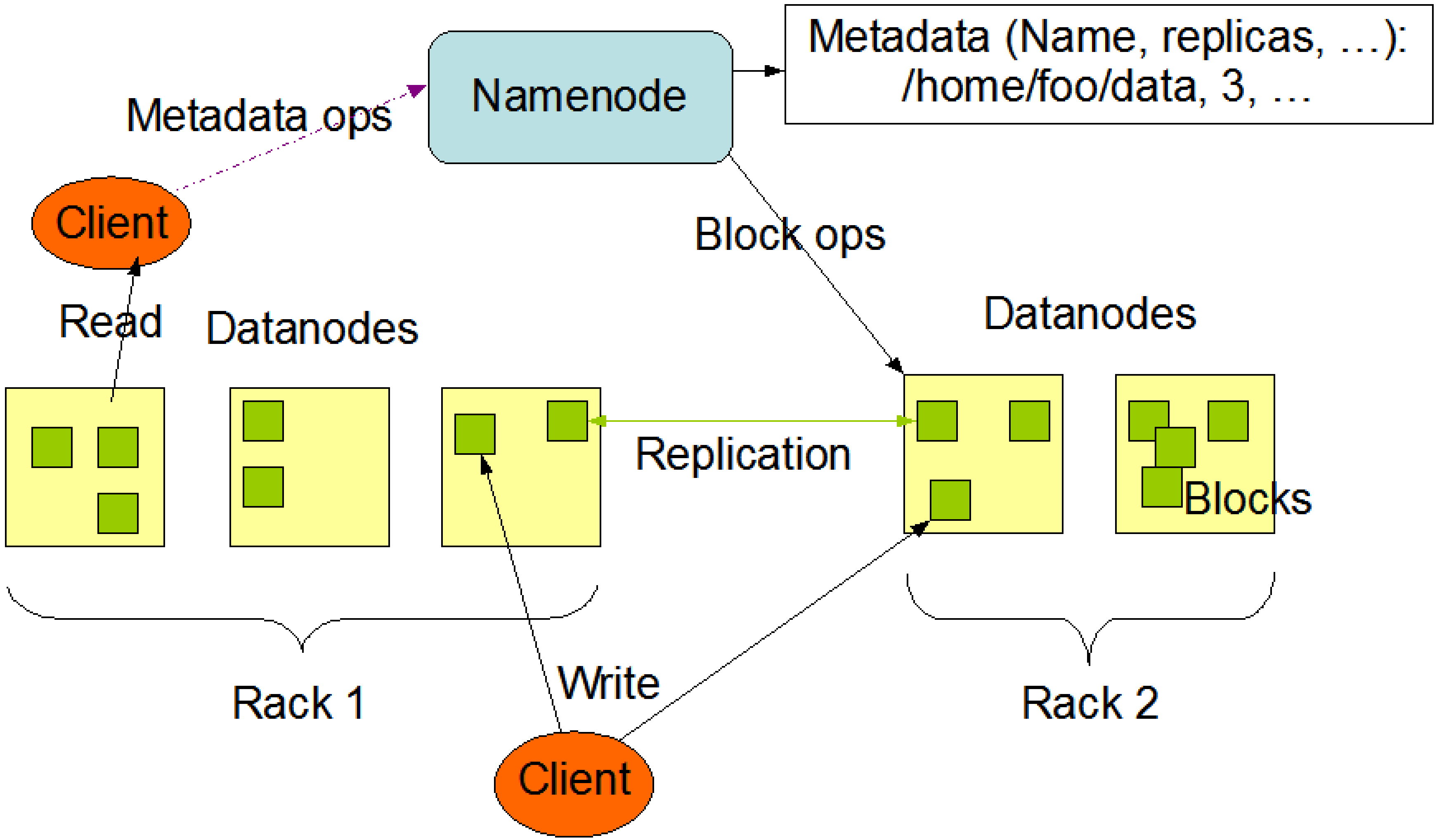
- Assumes:
  - Streaming data access
  - Batch processing rather than interactive user access
- Large data sets and files
- Write-once / read-many
  - A file once created, written and closed does not need to be changed
    - Or not often
  - This assumption simplifies coherency
- Optimal applications for this model: MapReduce, web-crawlers, …

HDFS

NameNode, DataNodes

- Master/slave architecture
- HDFS exposes file system namespace
- File is internally split into one or more blocks
  - Typical block size is 64MB (or 128 MB)
- NameNode = master server that manages the file system namespace + regulates access to files by clients
  - Opening/closing/renaming files and directories
  - Determines mapping of blocks to DataNodes
- DataNode = serves read/write requests from clients + performs block creation/deletion and replication upon instructions from NameNode
  - Usually one per node in a cluster
  - Manages storage attached to the node that it runs on

HDFS Architecture



HDFS

Namespace

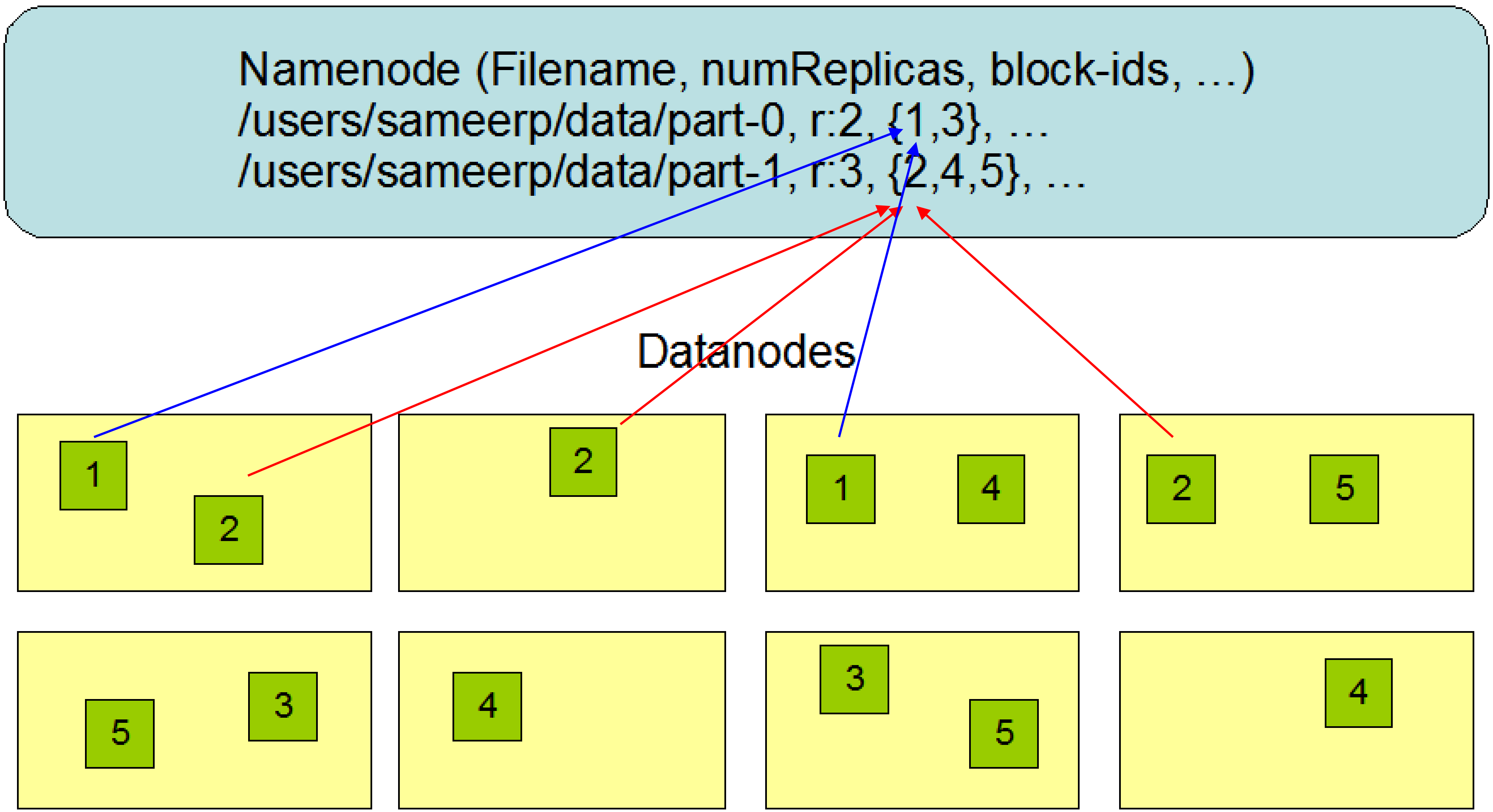
- Hierarchical file system
  - Directories and files
- Create, remove, move, rename, ...
- NameNode maintains the file system
  - Any meta information changes to the file system are recorded by the NameNode
- An application can specify the number of replicas of the file needed
  - Replication factor of the file
  - The information is stored in the NameNode

HDFS

Data Replication

- HDFS is designed to store very large files across machines in a large cluster
  - Each file is a sequence of blocks
  - All blocks in the file are of the same size
    - Except the last one
    - Block size is configurable per file
- Blocks are replicated for fault tolerance
  - Number of replicas is configurable per file
- NameNode receives HeartBeat and BlockReport from each DataNode
  - BlockReport contains a list of all blocks on a DataNode

Block Replication



HDFS

Replica Placement

- Placement of the replicas is critical to reliability and performance
- Rack-aware replica placement = to take a node's physical location into account while scheduling tasks and allocating storage
  - Needs lots of tuning and experience
- Idea:
  - Nodes are divided into racks
  - Communication between racks through switches
  - Network bandwidth between machines on the same rack is greater than those in different racks
- NameNode determines the rack id for each DataNode

HDFS

Replica Placement

- Any ideas?
- First idea: replicas should be placed on different racks
  - Prevents losing data when an entire rack fails
  - Allows use of bandwidth from multiple racks when reading data
    - Multiple readers
  - Writes are expensive (transfer to different racks)
    - We need to write to all replicas
- Common case: replication factor is 3
  - Replicas are placed:
    - One on a node in a local rack
    - One on a different node in the local rack
    - One on a node in a different rack
  - Decreases the inter-rack write traffic

HDFS

How NameNode Works?

- Stores HDFS namespace
- Uses a transaction log called EditLog to record every change that occurs to the file system's meta data
  - E.g., creating a new file, change in replication factor of a file, ..
  - EditLog is stored in the NameNode's local file system
- FsImage – entire file system namespace + mapping of blocks to files + file system properties
  - Stored in a file in NameNode's local file system
  - Designed to be compact
    - Loaded in NameNode's memory
    - 4 GB of RAM is sufficient

HDFS

How NameNode Works?

- When the filesystem starts up:
  1. It reads the FsImage and EditLog from disk
  2. It applies all the transactions from the EditLog to the in-memory representation of the FsImage
  3. It flushes out this new version into a new FsImage on disk = checkpoint
  4. It truncates the edit log
- Checkpoints are then built periodically
- Recovery = last checkpointed state

HDFS

How DataNode Works?

- Stores data in files in its local file system
  - Has no knowledge about HDFS file system
- Stores each block of HDFS data in a separate file
- Does not create all files in the same directory
  - Local file system might not be support it
  - Uses heuristics to determine optimal number of files per directory
- When the file system starts up:
  1. It generates a list of all HDFS blocks = BlockReport
  2. It sends the report to NameNode

HDFS

Failures

- Primary objective: to store data reliably in the presence of failures
- Three common failures:
  - NameNode failure
  - DataNode failure
  - Network partition

HDFS

Failures

- Network partition can cause a subset of DataNodes to lose connectivity with NameNode
  - NameNode detects this condition by the absence of a Heartbeat message
  - NameNode marks DataNodes without HearBeat and does not send any IO requests to them
  - Data registered to the failed DataNode is not available to the HDFS
- The death of a DataNode may cause replication factor of some of the blocks to fall below their specified value → re-replication
  - Also happens when replica is corrupted, hard disk fails, replication factor is increased, …

HDFS

API

- Java API for application to use
  - Python access can be used
  - C language wrapper for Java API is available
- HTTP browser can be used to browse the files of a HDFS instance
- Command line interface called the FS shell
  - Lets the user interact with data in the HDFS
  - The syntax of the commands is similar to bash
  - e.g., to create a directory `/foodir`

```
/bin/hadoop fs -mkdir /foodir
```

Hadoop file system

- Browser interface is available to view the namespace

References

- **Apache Hadoop:** http://hadoop.apache.org/
- Hadoop: **The Definitive Guide**, by Tom White, 2nd edition, Oreilly's, 2010