Slides credit: Matthias Boehm

Data Integration and Large Scale Analysis

09- Distributed Data Storage

Dr. Lucas Iacono - 2025

Institute of
HUMAN-CENTRED COMPUTING

Agenda

- Announcements
- Motivation and Terminology
- Data Lakes
- Object Stores
- Distributed File Systems
- Key-Value Stores and Cloud DBMS (+ eWarehouses)

Announcements

Announcements

- **Project**
  - January 10. **Project Submission Deadline**
- **Final Exams**
  - January 30th (First Exam)
  - March 20th (Second Exam)

Motivation and Terminology

Institute of
HUMAN-CENTRED COMPUTING

Motivation and Terminology

Overview Distributed Data Storage: Distributed DBS (#03)

- **What?**
  - A DBS is a virtual (logical) database that appears as a **single database** to the user but is **composed** by **multiple physical databases** located in **different physical locations**.
- **Why?**
  - Store and process data efficiently (e.g. data spread across different geographical locations).
  - Lets users access and work with data as if it were all in one place, even though it's distributed.

Motivation and Terminology

Overview Distributed Data Storage: Components for Global Query Processing

What if you run a query in a DBS?

1. **Identify** Figure out which physical databases contain the requested information.
2. **Unify** Combine the data from multiple databases as if it came from a single source.
3. **Optimize** Ensure the distributed system is fast and efficient when handling queries.

Motivation and Terminology

Overview Distributed Data Storage: DBS Types

- **Virtual DBS (homogeneous):**
  - All databases use the same technology and schema.
  - **Example:** Several PostgreSQL databases distributed across locations, all with same tables, attributes and relationships.
- **Federated DBS (heterogeneous):**
  - The databases can use different technologies or schemas.
  - **Example:** Two PostgreSQL databases with different entities, relationships and attributes.

Motivation and Terminology

Overview Distributed Data Storage: Cloud & Distributed Storage

Why?

1. **Large-scale:** handle **very large amounts** of data.
2. **Semi-structured/nested.** Data doesn't align with traditional rows and columns (**JSON, XML**).
3. **Fault tolerance:** Data **available** and **reliable** despite system components' failures

Motivation and Terminology

Overview Distributed Data Storage: Cloud & Distributed Storage

Types: Cloud Storage

1. **Block Storage (e.g. AWS EBS):**
   a. Data splitted into **blocks**, which can be individually read or written.
   b. Used for systems that need fast, low-level access to data.

Motivation and Terminology

Overview Distributed Data Storage: Cloud & Distributed Storage

Types: Cloud Storage

1. **Object Storage (e.g. AWS S3):**
    a. Data stored as objects (data, metadata, and UID).
    b. Ideal for storing unstructured data like media files, backups, or large datasets.
    c. Objects of a limited size (e.g., 5TB in AWS S3).

Motivation and Terminology

Overview Distributed Data Storage: Cloud & Distributed Storage

Types: Distributed file systems

1. **Distributed File Systems (e.g. HDFS):**
   a. File systems built on top of **block** or **object** storage to manage files across multiple servers.
   b. Allow for large-scale file sharing and processing.

Motivation and Terminology

Overview Distributed Data Storage: Cloud & Distributed Storage

Types: Database as a Service (DBaaS) - 1

NoSQL Databases (e.g. Redis, MongoDB):

a. **Target:** Designed for flexibility and scalability.
b. **Types:**
   i. **Key-Value Stores**: Store **data** as **key-value** pairs
   ii. **Document Stores:** Store **data** as **documents**

Motivation and Terminology

Overview Distributed Data Storage: Cloud & Distributed Storage

Types: Database as a Service (DBaaS) - 2

Cloud DBMSs (e.g. Amazon RDS, Google Cloud SQL):

a. **Target:** handle OLTP and OLAP workloads.
b. Combine SQL databases with cloud scalability and flexibility.

Motivation and Terminology

Central Data Abstractions: Files and Objects

- **File:** large and continuous block of data saved in a specific format (CSV, JSON, Binary, etc.).
- **Object:** like a file, but binary and it includes metadata (E.g. Images on S3)

Motivation and Terminology

Central Data Abstractions: Distributed Collections

- **Logical multiset (bag)** of key-value pairs (unsorted collection)
- **Different physical representations** key-value pairs can be stored in various ways (e.g., database, across files, or in memory).
- **Easy Distribution via Horizontal Partitioning.** Data divided into "chunks" (shards or partitions) based on the keys. Each chunk stored on a different machine (easier to handle large-scale data).
- **How collections are created:** from single file with data or a folder of files (even if they're messy and unsorted).

Key	Value
173010	100
173025	110
173045	150
173058	160
175500	70
173110	190
175515	0

Data Lakes

Data Lakes

Data Lake concept: a massive collection of datasets that may…

- be **hosted** in different storage systems
- **vary** in their formats
- **not** be accompanied by any useful **metadata** or may use different formats to describe their metadata
- **change** autonomously over time

Data Lake Management: Challenges and Opportunities

Fatemeh Nargesian, University of Toronto, fnargesian@cs.toronto.edu
Erkang Zhu, University of Toronto, ekzhu@cs.toronto.edu
Renée J. Miller, Northeastern University, miller@northeastern.edu
Ken Q. Pu, UOIT, ken.pu@uoit.ca
Patricia C. Arocena, TD Bank Group, prg@cs.toronto.edu

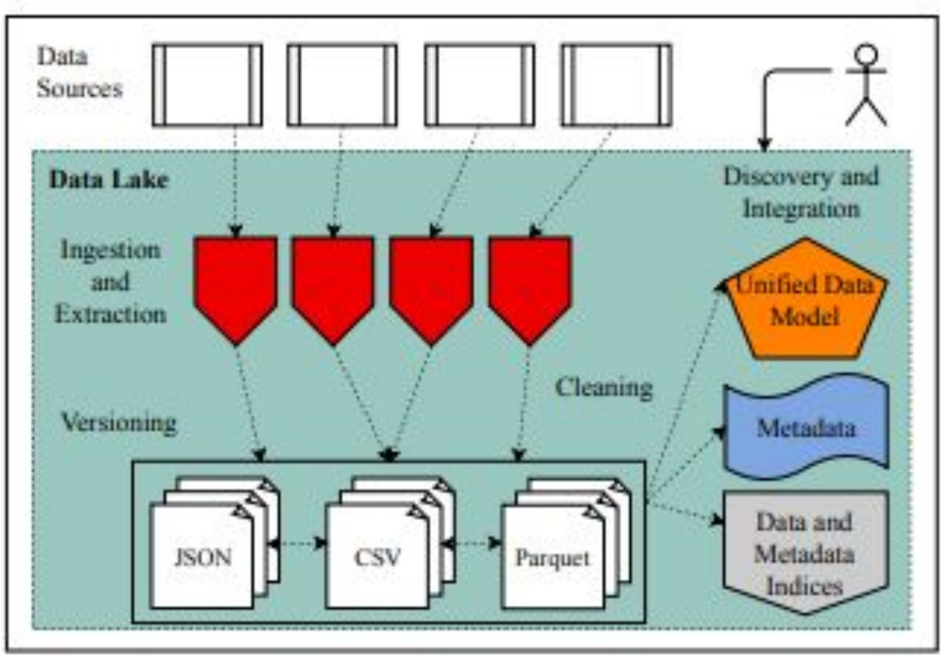
ABSTRACT

The ubiquity of data lakes has created fascinating new challenges for data management research. In this tutorial, we review the state-of-the-art in data management for data lakes. We consider how data lakes are introducing new problems including dataset discovery and how they are changing the requirements for classic problems including data extraction, data cleaning, data integration, data versioning, and metadata management.

While some of the data in a lake is extracted, transformed, and loaded into existing database management systems (DBMS) or data warehouses, some of it may be exclusively consumed on-demand by programming environments to perform specific data analysis tasks. Moreover, the value of some of this data is transient, meaning additional analysis is required to create information with sufficient value to load into a data warehouse. Even though some of this data is not destined for traditional DBMS, there are still many open and fascinating data management research problems.

Nargesian, F., Zhu, E., Miller, R. J., Pu, K. Q., & Arocena, P. C. (2019). Data lake management: challenges and opportunities. *Proceedings of the VLDB Endowment*, *12*(12), 1986-1989.

Data Lakes



Data Lake Management System [*]


Data Lake Management: Challenges and Opportunities

Fatemeh Nargesian, University of Toronto
Erkang Zhu, University of Toronto
Renée J. Miller, Northeastern University
Ken Q. Pu, UOIT
Patricia C. Arocena, TD Bank Group

[*] Nargesian, F., Zhu, E., Miller, R. J., Pu, K. Q., & Arocena, P. C. (2019). Data lake management: challenges and opportunities. *Proceedings of the VLDB Endowment*, *12*(12), 1986-1989.

Data Lakes

PROS

- **Store Everything:**
  - Store all kinds of data, no matter its structure.
  - Data added as-is (append-only) → not modified in place.
- **No Pre-Planning Required:**
  - No need for defining a schema before adding data.
  - Useful for situations where analysis to perform is not yet clear.

Data Lakes

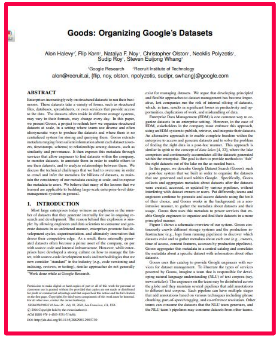
PROS

- **File-Based Storage:**
  - Data stored as raw files, open formats (CSV, JSON, etc).
  - Files can serve as **inputs** or **intermediate outputs** for further processing.
- **Scalable and Agile:**
  - Data lakes rely on **distributed storage** to handle **large datasets**.
  - They support **distributed analytics** for processing data quickly and efficiently. **(Not a Warehouse!)**

Data Lakes

CONS "Data Swamp"

- **Low Data Quality** Without a schema data might be incomplete, incorrect, duplicated or inconsistent.
- **Missing Metadata** hard to search and understand what the data is for.
- **No Data Catalog** Without a clear catalog, it's challenging to locate specific datasets in the lake.
- **Solution:** data curation, metadata management, data catalog, governance, provenance.


Goods: Organizing Google's Datasets

Halevy et.al. (2016, June). Goods: Organizing google's datasets. In Proceedings of the 2016 International Conference on Management of Data (pp. 795-806).

Institute of HUMAN-CENTRED COMPUTING

Object Stores

Object Storage

Key-Value stores

- **Key-value mapping**
  - Values can be of a variety of data types
  - E.g. **"250" {"sensor": "Speed_FW_Left" , "raw": 150}**
- **Scalability using Sharding**:
  - Datasets splitted into smaller chunks (shards) across multiple machines.
  - Each shard handles a subset of the **key-value** pairs, enabling the system to scale efficiently.
- **APIs for CRUD**
  - Enable entities to interact with the **key-value** store to perform these operations (Copy-Read-Update-Delete)

HCC
Institute of
HUMAN-CENTRED COMPUTING

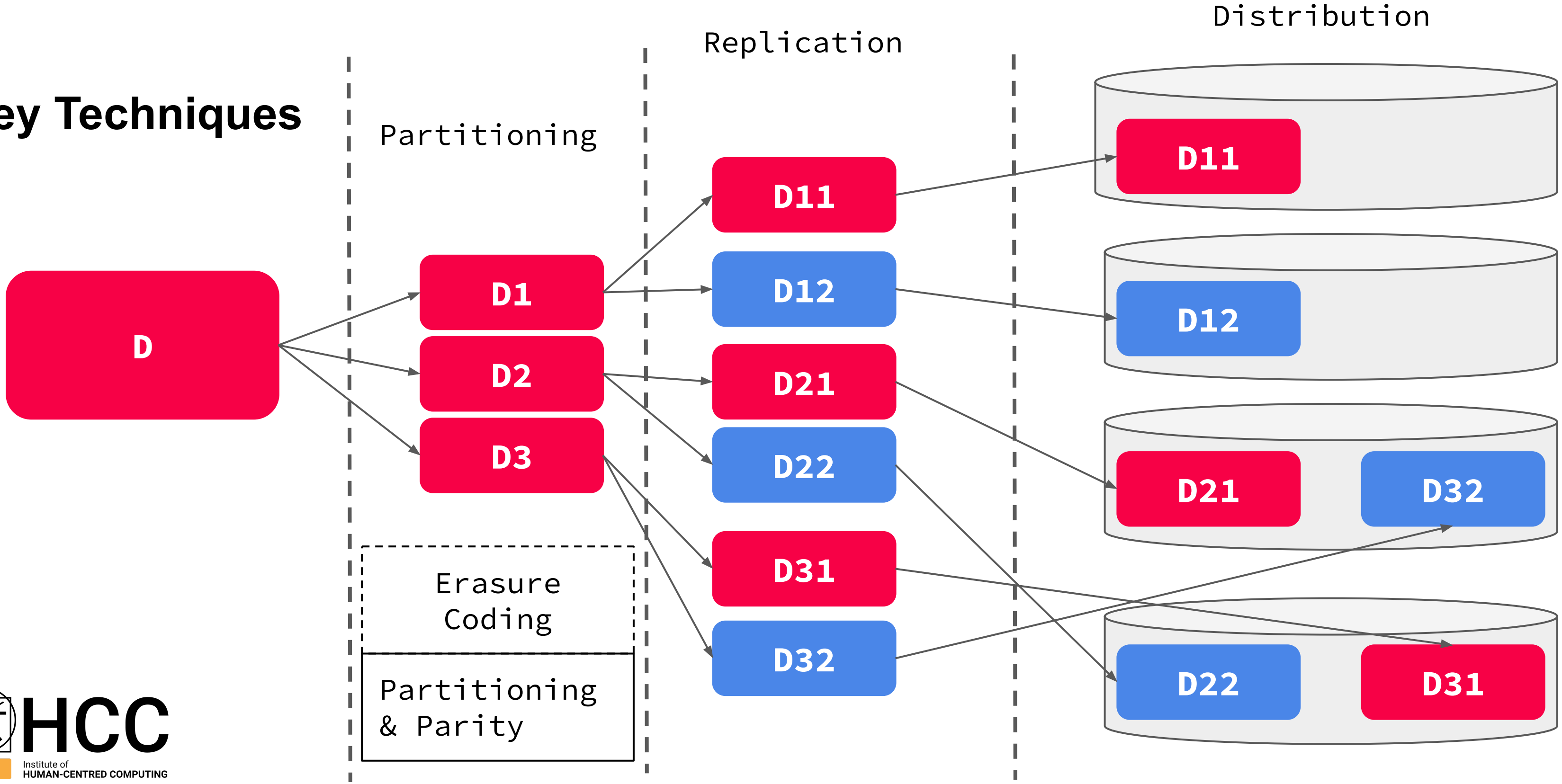
Object Storage

Object Store. Similar to key-value stores, but **optimized for large objects** (videos, backups, etc.).

- **Structure:**
  - **Object Identifier** (**Key**): **UID** to retrieve the object.
  - **Metadata** (e.g., size, type, creation date).
  - **Object** (**BLOB)**: The actual data, stored as a Binary Large Object. (E.g. Image at Azure Blob)
- **APIs:**
  - REST APIs: HTTP-based interfaces for CRUD
  - SDKs: Programming libraries for easier integration with applications.

Object Storage

Key Techniques



Object Storage

Examples: Amazon Simple Storage Service (S3)

- Reliable object store for photos, videos, documents or any binary data
- **Bucket**: Uniquely named, static data container
- **arn**:aws:s3:::distributed-storage
- **https**://distributed-storage.s3.us-west-2.amazonaws.com/Temperature-processed.csv
- Single (5GB)/ multi-part (5TB)
- AWS CLI Upload / Download
- Storage classes: STANDARD, STANDARD_IA, GLACIER

Distributed File Systems

Institute of
HUMAN-CENTRED COMPUTING

Hadoop Distributed File System (HDFS)

Brief Hadoop History

- Google's GFS + MapReduce [ODSI'04] → Apache Hadoop (2006).
- Spark (HDFS + Others)

HDFS Overview

- Hadoop's distributed file system → **clusters and large datasets**
- Implemented in **Java**, w/ native libraries for **compression, I/O, CRC32**
- Files split into **128MB blocks**, replicated **(3x)**, and distributed

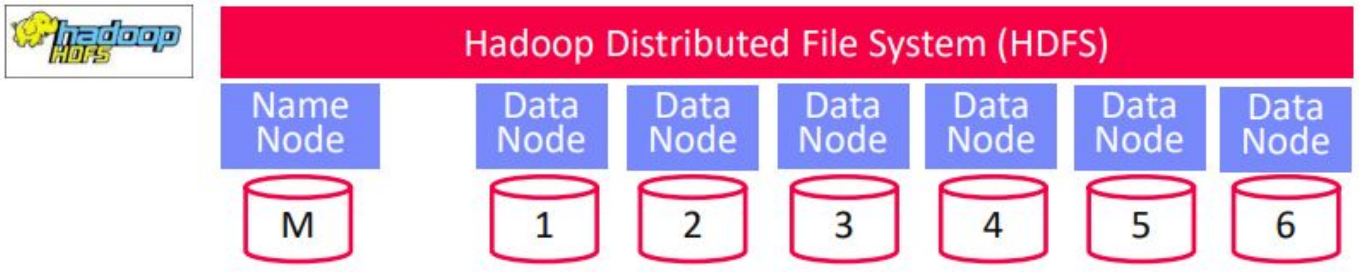
Hadoop Distributed File System (HDFS)

How HDFS works:

- **Split files** into **Blocks**
- **Store blocks** across **DataNodes**
- **Replicate blocks** across **DataNodes** (reliability)
- **NameNode coordinates** this process

Client

```
hdfs dfs -put /local/path /hdfs/path
```



Hadoop Distributed File System (HDFS)

NameNode

- **Coordinator daemon** that manage file system **namespace (metadata)**
- **Metadata** (replication, permissions, sizes, block ids, directory structure, etc)

DataNode

- **Worker daemon per cluster node** that manages block storage (list of disks)
- Block **creation, deletion, replication as individual files** in local FS
- **On startup:** *scan* local blocks and *send* block report to **NameNode**
- Serving data block **read and write** requests
- **Send heartbeats** to **NameNode** (capacity, current transfers) and **receives replies** (replication, removal of block replicas) from to **NameNode**

HDFS InputFormats and RecordReaders

Overview HDFS API

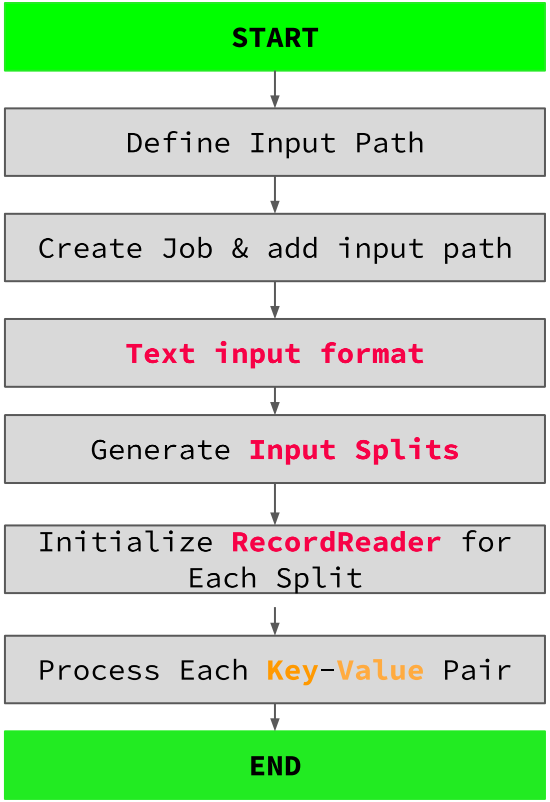
- InputFormats Implements access to distributed files (data collections)
- Split **Logical division (w/offsets)** of the input data aligned with the block size of HDFS **(128 MB)**.
- RecordReader: API for reading key-value pairs from file splits **(block creation)**
    - **Keys: offsets, Values: data between offsets**
- Examples: FileInputFormat, TextInputFormat, SequenceFileInputFormat

HDFS InputFormats and RecordReaders

Overview InputFormats Example

```
FileInputFormat.addInputPath(job,
path); # path: dir/file
TextInputFormat informat = new
TextInputFormat();
InputSplit[] splits =
informat.getSplits(job, numSplits);
LongWritable key = new LongWritable();
Text value = new Text();
for(InputSplit split : splits) {
RecordReader<LongWritable,Text> reader
= informat
.getRecordReader(split, job,
Reporter.NULL);
while( reader.next(key, value) )
... //process individual text lines
}
```

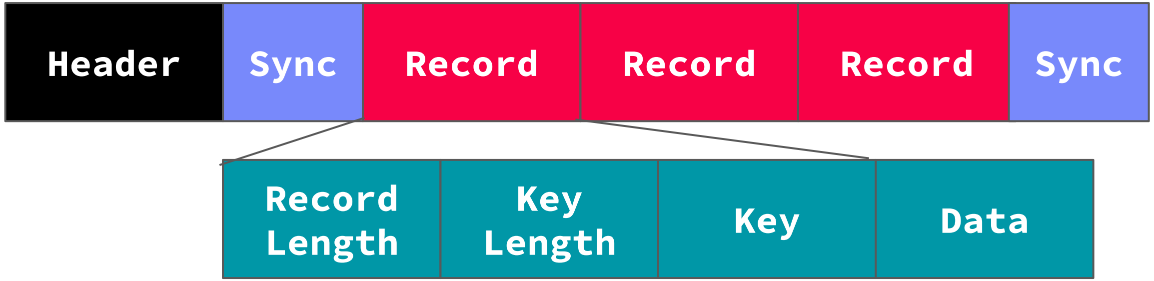
Key-Value	
Byte offset	Line content



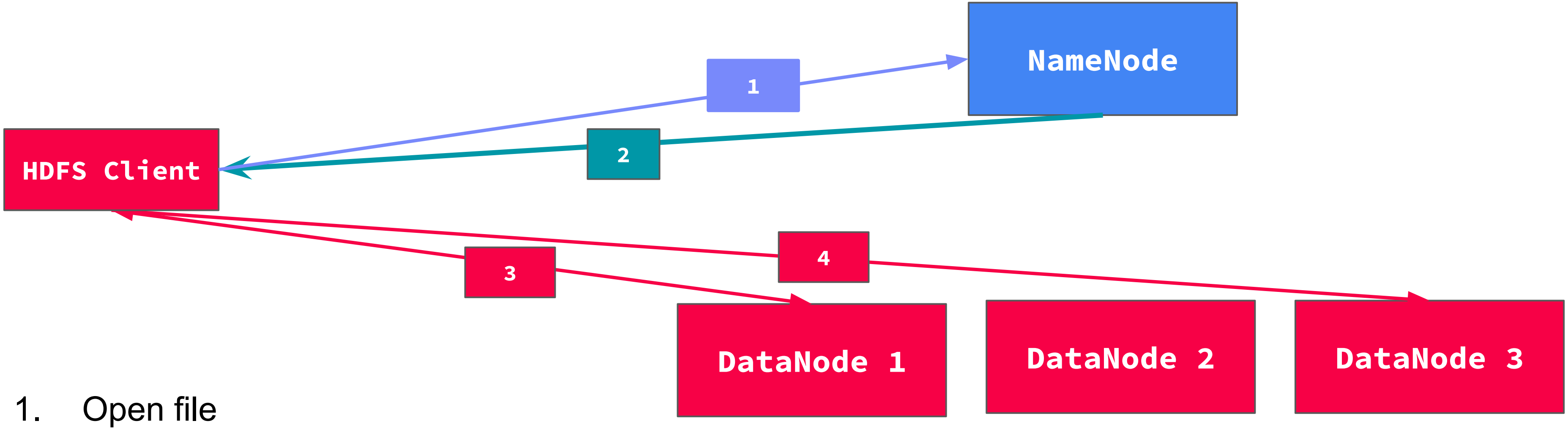
HDFS InputFormats and RecordReaders

Sequence Files (store **key-value** pairs efficiently)

- **Binary File format** (reducing storage overhead compared to plain text files)
- **Optimal Compression** Record-level and block-level compression
- **Input & Output** in MapReduce/Spark jobs
- **Intermediate storage** during MapReduce workflows
- **Splittable:** they can be splittable allowing parallel processing in Hadoop.



HDFS Read

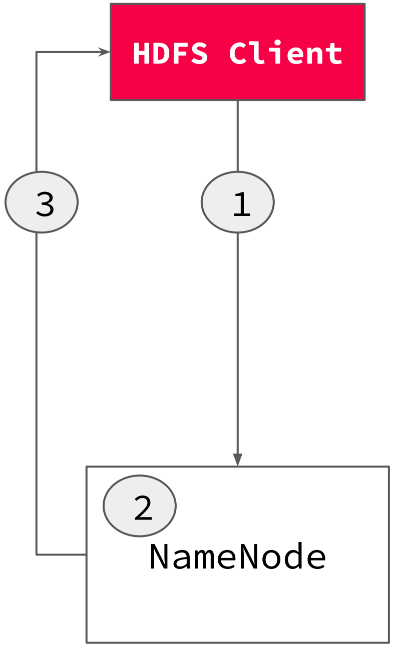


1. Open file
2. Metadata (Block Locations, Replicas)
3. Read Block 1 (DataNode 1)
4. Read Block 2 (Data Node 3)

HDFS Write

HDFS Client →NameNode

1. Create a new file in HDFS.
2. The **NameNode** checks whether the **file already exists** and if the client has the **necessary permissions** to create it.
3. If everything is valid, the NameNode:
    a. Allocates a **lease** to the client, granting it the right to write to the file.
    b. Identifies a set of **DataNodes** (+replica nodes) where the blocks of the file will be stored.



HCC
Institute of
HUMAN-CENTRED COMPUTING

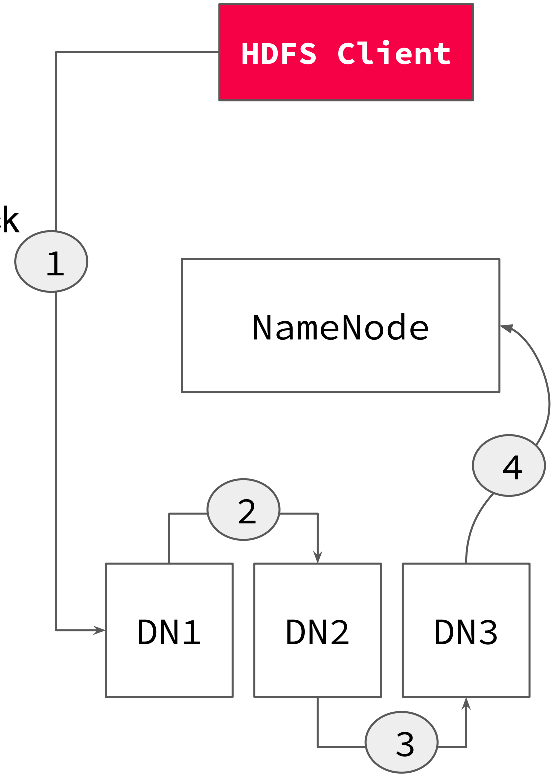
HDFS Write

Writing Blocks to DataNodes

1. The client begins writing data in **blocks** to the first DataNode in the pipeline.
2. Once the first DataNode receives a block, it forwards a copy of the block to the second DataNode in the pipeline.
3. The second DataNode then forwards the block to the third DataNode, completing the **replication pipeline**.

DataNodes Report to the NameNode

1. After receiving and storing the data blocks, each DataNode sends a **heartbeat**. It includes a report confirming that the DataNode has **successfully stored** the block and is **ready** for future tasks.



HDFS Data Locality

HDFS is generally rack-aware (node-local, rack-local, other)

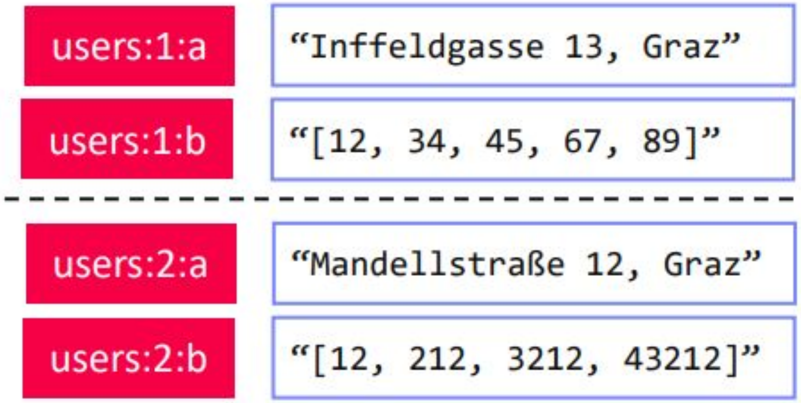
Scheduler reads from closest data node

Replica placement (x3): local DN, other-rack DN, same-rack DN

Key-Value Stores, Cloud DBMS, EWarehouses)

Key-Value Stores Motivation

- **Complex data models** (e.g., JSON) can be transformed into simple **key-value** pairs.
- Designed to operate reliably at **very-large scale** using commodity hardware and distributing the workload across multiple servers.
- **Key-value maps**, where values can be of a variety of data types
- **APIs for CRUD** operations (create, read, update, delete)
- **Scalability via sharding** (horizontal partitioning)



HCC
Institute of HUMAN-CENTRED COMPUTING

Key-Value Stores: Example

Amazon DynamoDB Simple, highly-available data storage for small objects in ~1MB range (data, shopping carts)

- Aims to achieve Service Level Agreements **(SLAs)** that guarantee **99.9%** of requests are served with low latency, even under high loads.
- **System interface**: get and put operations
- **Partitioning** using consistent hashing where Nodes are organized in a ring structure and each node is responsible for a specific range of **keys**.
- **Replication,** Each data is **replicated N times** to provide fault tolerance.
- **Eventual consistency**

Cloud Databases (DBaaS): Motivation and Key-aspects

DBaaS: A cloud service model that allows users to access, manage, and scale databases without dealing with the underlying infrastructure.

Providers handle database configuration, maintenance, security, updates, and availability.

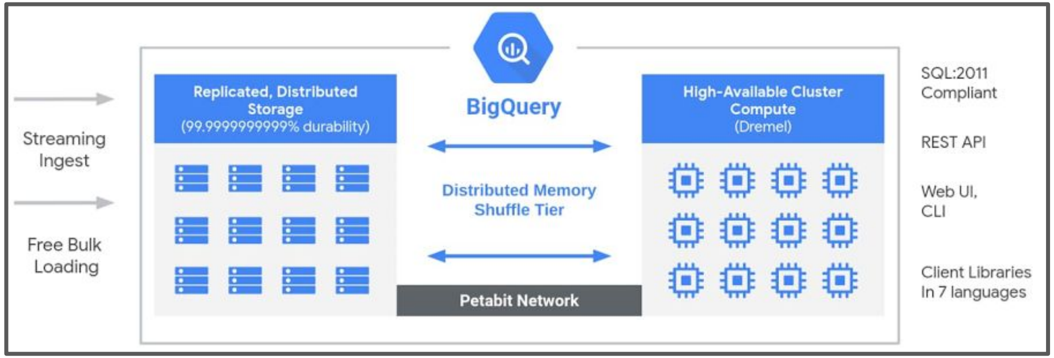
Key aspects

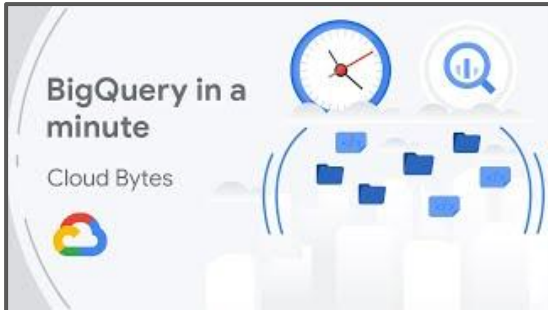
- SQL and NoSQL (MongoDB, DynamoDB).
- Auto-scaling, high availability, and support for multiple data types.
- Mainly used for OLTP (web apps, IoT, etc.)

Examples: Amazon RDS, Google Cloud SQL, Azure SQL Database

Elastic Data Warehouses

- **Data warehousing + Cloud Computing + Distributed Storage**
- Used for OLAP





Summary and Q&A

Institute of
HUMAN-CENTRED COMPUTING

Summary and Q&A

- Motivation and Terminology
- Data Lakes
- Object Stores (S3)
- Distributed File Systems (HDFS)
- Key-Value Stores, Cloud DBMS, Elastic Data Warehouses
- Next Lecture: Distributed, Data-Parallel Computation **[Dec 19]**

Vielen Dank!