Data Integration and Large Scale Analysis

Slides credit: Matthias Boehm - Shafaq Siddiqi

09- Distributed Data Storage

TU Graz
SCIENCE
PASSION
TECHNOLOGY

Lucas Iacono. PhD. - 2024

ISDS

Part B

Large-Scale Data Management & Analysis

- LU3. Cloud Computing
  - Cloud Computing Fundamentals [Nov 29]
  - Cloud Resource Management and Scheduling [Dec 06]
  - **Distributed Data Storage[Dec 13]**

Part B

Large-Scale Data Management & Analysis

- LU4. Large-Scale Data Analysis
  - Distributed, Data-Parallel Computation **[Dec 20]**
  - Distributed Stream Processing **[Jan 10]**
  - Distributed Machine Learning Systems **[Jan 17]**

Agenda

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

Announcements

Announcements

- **Course Evaluation and Exam**
  - Exercise submission deadline **January 13**
  - Evaluation Period: **Dec 09 - Feb 13**
  - Exam date: **Feb 07, 3:00 PM** (90 Min written exam)
  - Second Exam date: TBD (~ 2 weeks after first exam)

Motivation and Terminology

Motivation and Terminology

Overview Distributed Data Storage: Distributed DBS (L#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 structure (schema).
  - **Example:** Several MySQL databases distributed across locations, all set up identically.
- **Federated DBS (heterogeneous):**
  - The databases can use different technologies or schemas.
  - **Example:** A PostgreSQL database working together with a MongoDB database.

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.
   c. **Analogy.** Books are split into pages (blocks), and you can quickly access any page.

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).
   d. **Analogy:** Each book is stored as a single unit with a label and description.

Motivation and Terminology

Overview Distributed Data Storage: Cloud & Distributed Storage

Types: Distributed file systems

1. **Distributed File Systems (e.g. NFS, 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.
   c. **Analogy:** A librarian manages where the books (files) are stored across multiple shelves (servers).

Motivation and Terminology

Overview Distributed Data Storage: Cloud & Distributed Storage

Types: Database as a Service (DBaaS) - 1

NoSQL Stores (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 Online Transaction Processing (OLTP) and Online Analytical Processing (OLAP) workloads.
b. Combine the traditional database structure with the scalability and flexibility of the cloud.
c. **Analogy:** The library has different sections (SQL and NoSQL) to organize your books either as detailed indexes (SQL) or free-form notes (NoSQL). All this is in a building (cloud) you don't have to manage!

Motivation and Terminology

Central Data Abstractions: Files and Objects

File: large and continuous block of data saved in a specific format (CSV, Binary, etc.).

Object: like a file, but binary and it comes with metadata (Images on S3)

Analogy

- File = book: a single block of information in a specific format.
- Object = book with a cover that has extra info.

Motivation and Terminology

Central Data Abstractions: Distributed Collections

Logical multi-set (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).

Analogy: A distributed collection is like organizing a library where each shelf (server) holds books based on their first letter.

Key	Value
4	Delta
2	Bravo
1	Alfa
3	Charlie
5	Echo
6	Foxtrot
1	Alfa

Motivation and Terminology

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

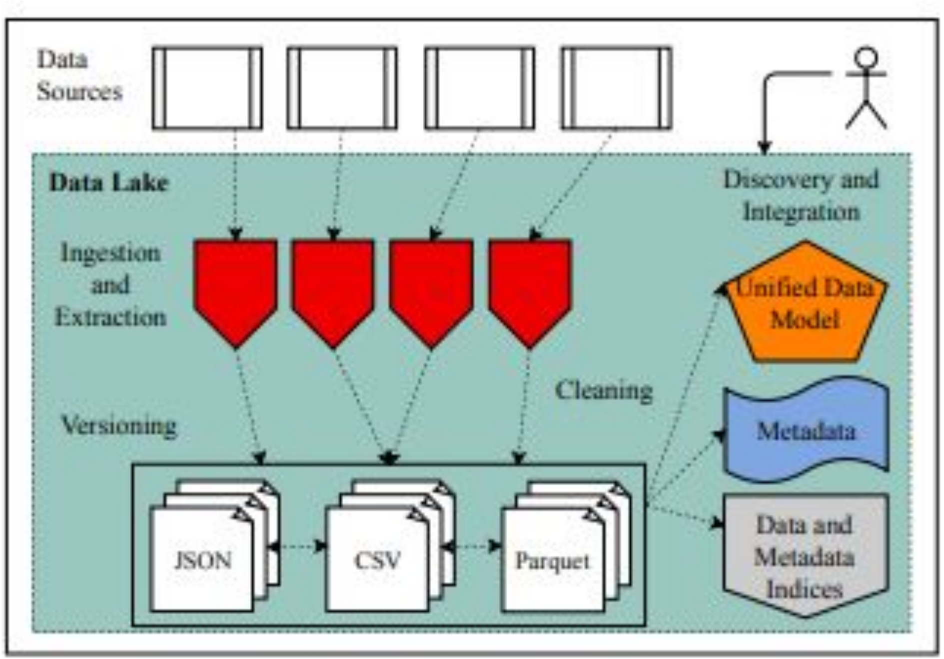
- be **hosted** in different storage systems
- **vary** in their formats
- **not** be accompanied by any useful **metadata** or may use different formats to describe their metadata
- **change** autonomously over time


Data Lake Management: Challenges and Opportunities

Fatemeh Nargesian, University of Toronto
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.

Motivation and Terminology



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.

Motivation and Terminology

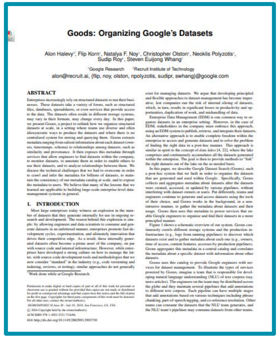
Data Lake key features:

- **Store Everything:**
  - Store all kinds of data, no matter its structure.
  - Data added as-is (append-only). Then, it’s not modified in place.
- **No Pre-Planning Required:**
  - No need for defining a fixed schema (data structure) before adding data.
  - Useful for situations where analysis to perform is not yet clear.
- **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 huge datasets.
  - They support distributed analytics for processing data quickly and efficiently.

Motivation and Terminology

Data Lake **downside** “Data Swamp”

- **Low Data Quality** Without a schema data might be incomplete, incorrect, 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

Alon Halevy[2], Flip Korn[1], Natalya F. Noy[1], Christopher Olston[1], Neoklis Polyzotis[1], Sudip Roy[1], Steven Euijong Whang[1]

[1]Google Research [2]Recruit Institute of Technology

alon@recruit.ai, {flip, noy, olston, npolyzotis, sudipr, swhang}@google.com

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

Object Stores and Distributed File Systems

Object Storage

Recap: Key-Value stores

- **Key-value mapping**
  - Values can be of a variety of data types
  - Example: “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)

Object Storage

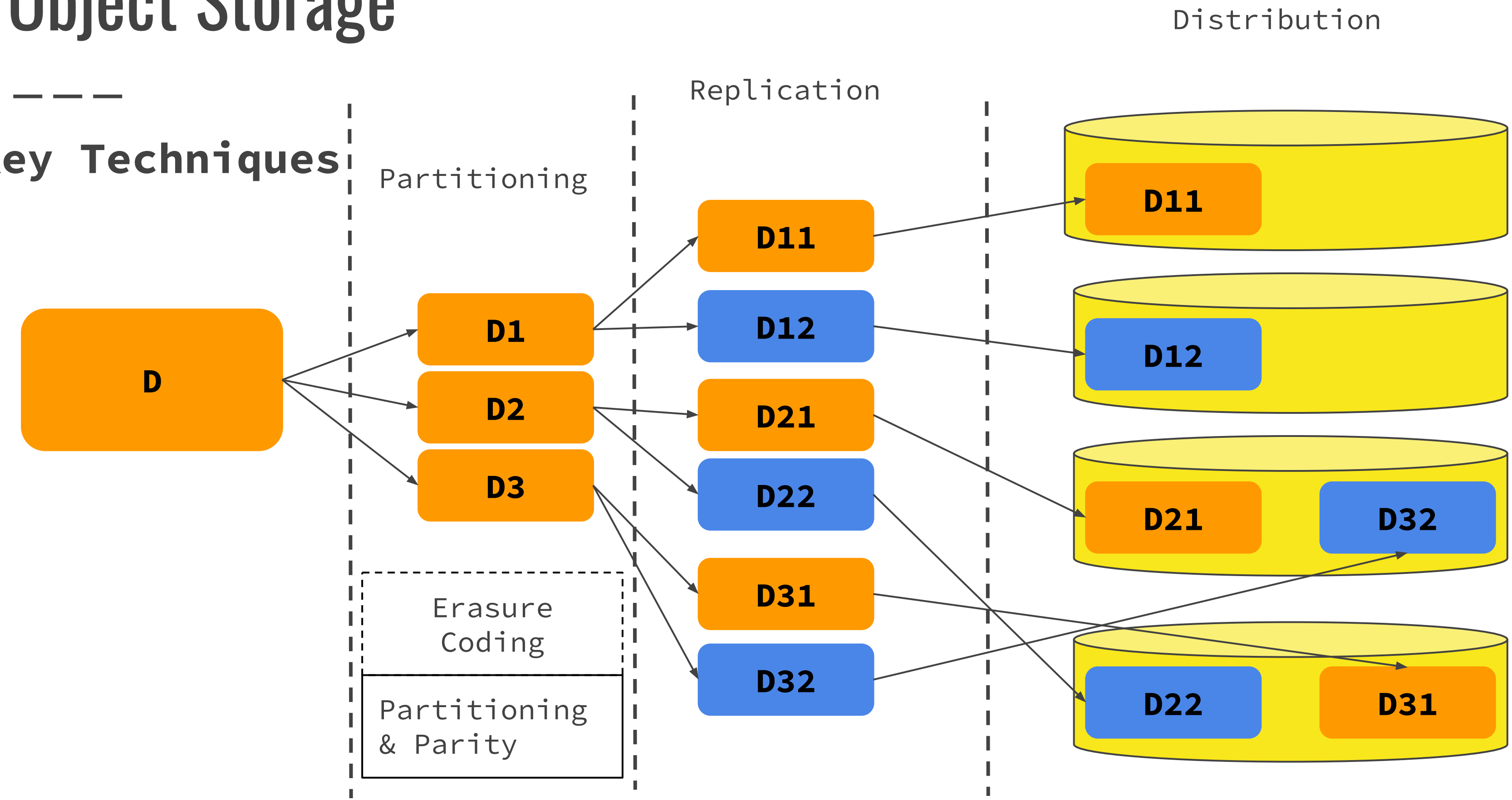
Object Store. Similar to key-value stores, but **optimized for large objects** (videos, backups, etc.).

- **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.

- **APIs:**
    - REST APIs: HTTP-based interfaces for CRUD
    - DFS APIs: APIs similar to Distributed File Systems (e.g., HDFS).
    - SDKs: Programming libraries for easier integration with applications.

Object Storage

Key Techniques



Object Storage

Examples: Amazon Simple Storage Service

- 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) upload and direct/BitTorrent download
- Storage classes: STANDARD, STANDARD_IA, GLACIER, DEEP_ARCHIVE
- Operations: GET/PUT/LIST/DEL, and SQL over CSV/JSON objects

Hadoop Distributed File System (HDFS)

Brief Hadoop History

- Google's GFS + MapReduce [ODSI'04] -> Apache Hadoop (2006).

HDFS Overview

- Hadoop's distributed file system, for **large clusters and 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 nodes
- **Replicate** blocks for reliability

Hadoop Distributed File System (HDFS)

HDFS NameNode

- **Keeps a record** of where every block is stored (it doesn’t store the actual data).
- **Metadata** for all files (e.g., replication, permissions, sizes, block ids, etc)

HDFS 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 name node
- Serving block **read and write** requests
- Send **heartbeats** to NameNode (capacity, current transfers) and **receives replies** (replication, removal of block replicas)

HDFS InputFormats and RecordReaders

Overview InputFormats

- **InputFormat:**
  - An interface or **class in Hadoop** that specifies how input data is divided into splits and provides access to data through **record readers.**
- **Determines:**
  - How files are **split** into manageable chunks for parallel processing.
  - How data is **presented** to the **Mapper** as **key-value** pairs.

HDFS InputFormats and RecordReaders

Overview InputFormats Split

- A **logical division** of the input data aligned with the block size of HDFS (**128 MB**).
- Each split is **processed** by a single **Mapper**, enabling parallelism.
- **Example:** 1 GB file and the HDFS block size is 128 MB, the file will be split into 8 chunks (splits).
- **Record alignment** ensures that splits **don’t break** data records (e.g., lines or rows).

HDFS InputFormats and RecordReaders

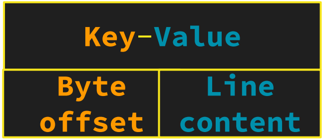
Overview InputFormats Record Reader

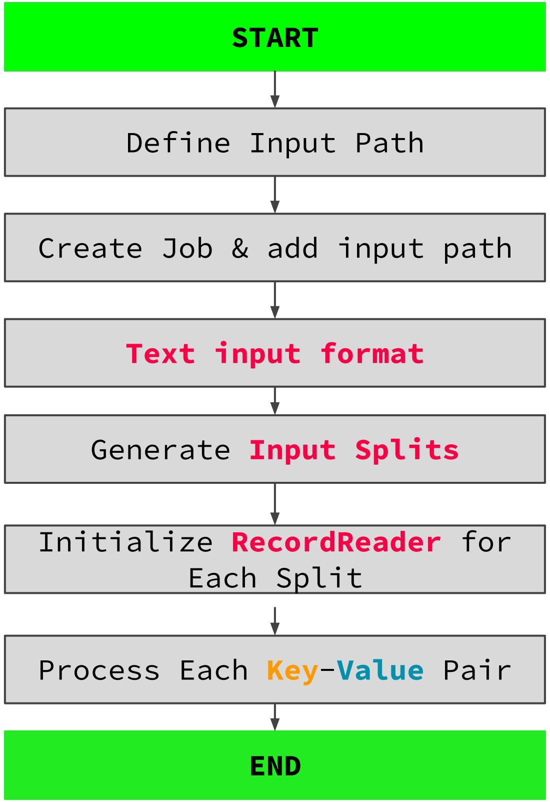
- **API** that converts each input split into **key-value** pairs.
- **Reads** the raw data (e.g., lines, binary records) and **formats** it into **key-value** pairs for the **Mapper**.

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
}
```

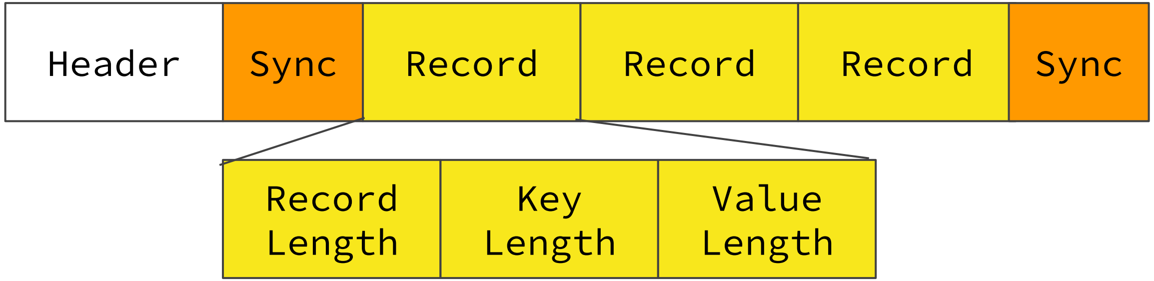




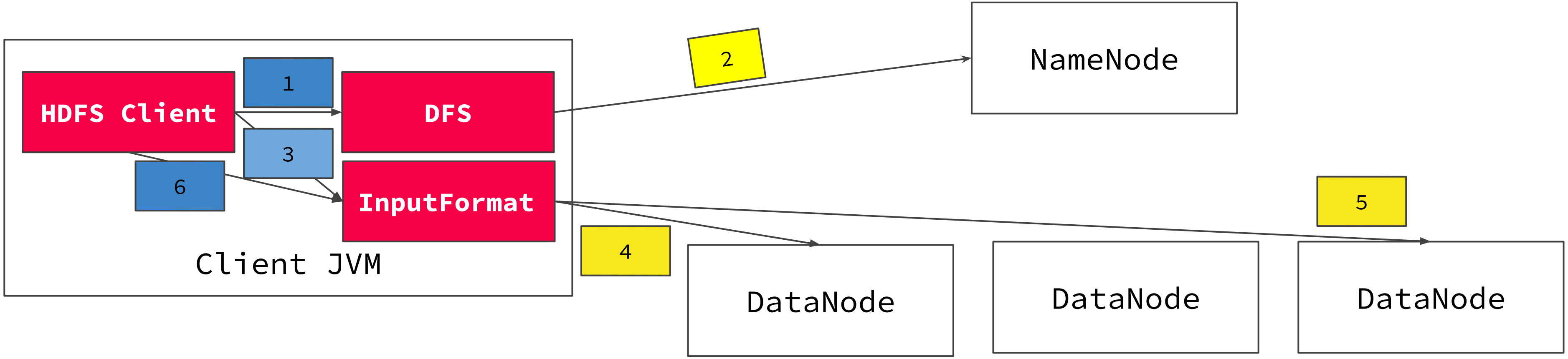
HDFS InputFormats and RecordReaders

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.
- **e.g.** frame1.jpg, frame2.jpg



HDFS Read

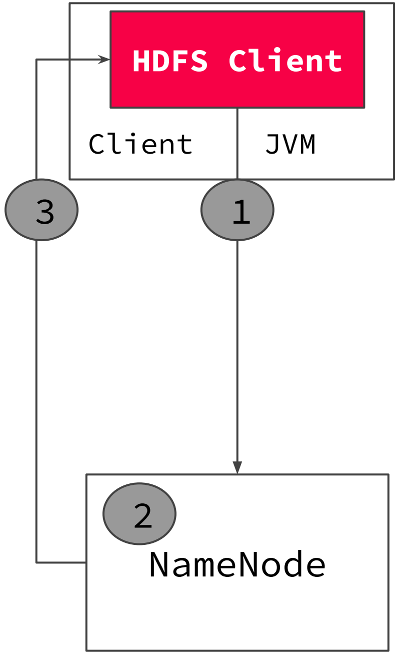


1. Open
2. Get Block Locations
3. Read
4. Read
5. Read
6. Close

HDFS Write

Client Communicates with the NameNode

1. **RPC (Remote Procedure Call)** to the **NameNode** to create a new file in HDFS.
2. The **NameNode** checks whether the **file already exists** and whether 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.



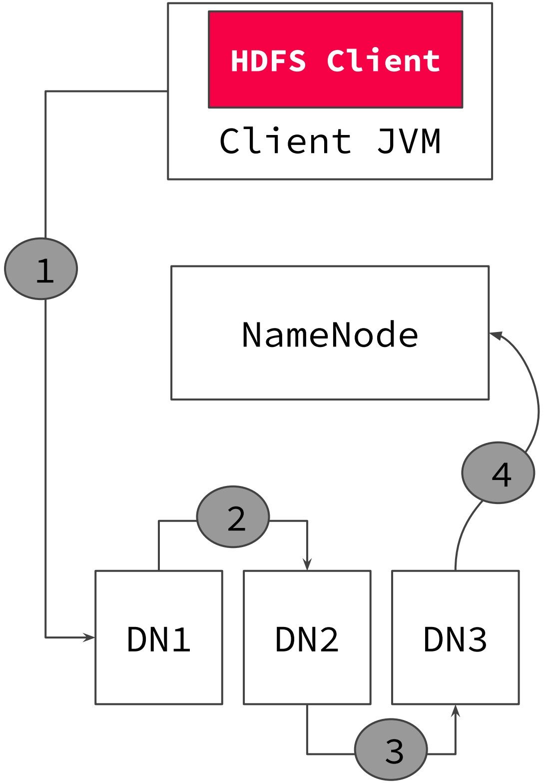
HDFS Write

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 (rep 3): local DN, other-rack DN, same-rack DN

Key-Value Stores and Cloud DBMS (+ EWarehouses)

Key-Value Stores Motivation

Motivation

Simple Data Representation: **Key-Value** Stores enable **mapping** data using a simple **API**, allowing more **complex data models** (e.g., JSON) to be transformed into simple **key-value** pairs.

Reliability at Scale: Designed to operate reliably at **very-large scale** using commodity hardware and distributing the workload across multiple servers. This is critical in cloud computing (**scalability and elasticity are essential**).


```
{
  "location": {
    "station_id": "KC002",
    "name": "Sandgasse Building",
    "coordinates": {
      "latitude": 47.0707,
      "longitude": 15.4395,
      "altitude": 400
    },
    "region": {
      "country": "Austria",
      "state": "Styria",
      "city": "Graz"
    }
  },
  "time_series": [
    {
      "timestamp": "2024-12-12T00:00:00Z",
      "data": {
        "temperature": {
          "value": -1.5,
          "unit": "°C",
          "quality_flag": "verified"
        },
        "humidity": {
          "value": 85,
          "unit": "%",
          "quality_flag": "verified"
        }
      },
      "timestamp": "2024-12-12T01:00:00Z",
      "data": {
        "temperature": {
          "value": -1.8,
          "unit": "°C",
          "quality_flag": "verified"
        },
        "humidity": {
          "value": 88,
          "unit": "%",
          "quality_flag": "verified"
        },
      }
    }
  ],
  "metadata": {
    "data_source": "WeatherUnderground",
    "sensor_details": {
      "temperature_sensor": " SHT31-D",
      "humidity_sensor": " SHT31-D",
    },
    "update_frequency": "10minutes",
    "quality_control": "automated + manual review",
    "notes": "Data not validated"
}
```

Key-Value Stores: Terminology

System architecture

- **Key-Value Map:** each key is associated with a single value (of a variety of data). E.g **"temperature:2024-12-12T00:00:00Z": -1.5**
- APIs for **CRUD** Operations:
  - **C**reate: Add new **key-value** pairs.
  - **R**ead: Retrieve the **value** associated with a **key**.
  - **U**pdate: Modify the **value** associated with an existing **key**.
  - **D**elete: Remove a **key**-**value** pair.
- Scalability via **Sharding** (Horizontal Partitioning):
  - Each server handles a subset of the data (e.g., a range of **keys**), **Server 1:** data for 2024-12-12, **Server 2:**data for 2024-12-13.
  - Scale horizontally by adding more servers as the data grow.

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**. Nodes hold multiple virtual nodes for load balance.
- **Replication,** Each data is **replicated N times** to provide fault tolerance.
- **Eventual consistency** (all replicas will eventually converge to the same state, but temporary inconsistencies may exist).

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.

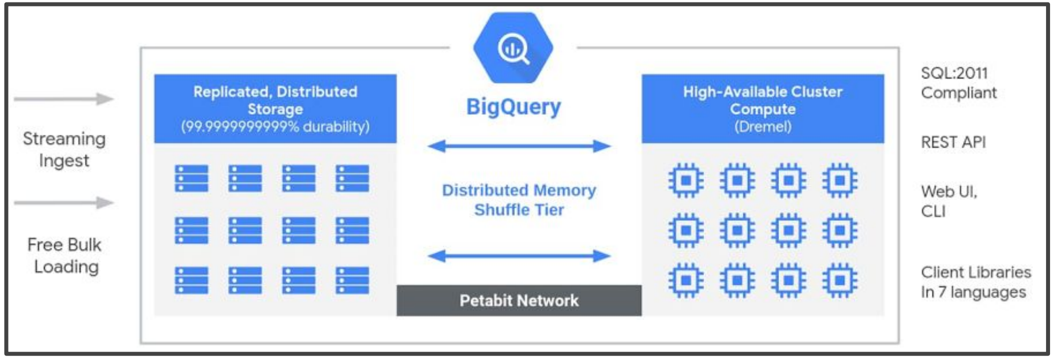
Key aspects

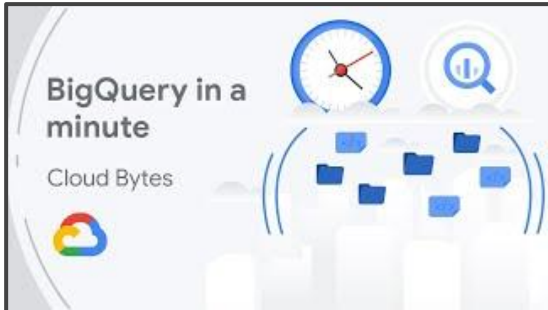
- SQL and NoSQL (MongoDB, DynamoDB).
- Auto-scaling, high availability, and support for multiple data types.
- Mainly used for transactional applications OLTP (web apps, IoT, etc.)

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

Elastic Data Warehouses

- **Data warehousing + Cloud Computing + Distributed Storage**
- Analytics and reporting (Online Analytical Processing - OLAP)





Summary and Q&A

Summary and Q&A

- **Summary and Q&A**
  - Motivation and Terminology
  - Object Stores and Distributed File Systems
  - Key-Value Stores and Cloud DBMS
- **Next Lectures**
  - Distributed, Data-Parallel Computation **[Dec 20]**

Vielen Dank!