

# Data Integration and Large Scale Analysis

## 09 Distributed Data Storage

**Shafaq Siddiqi**

Graz University of Technology, Austria



# Announcements/Org

## ■ Course Evaluation and Exam

- Exercise submission deadline **Jan 12**
- Exam registration is open till **Jan 19**
- Evaluation period open till **Feb 15**
- Exam date: **Feb 02, 3:00pm** in HSi11 (90 min written exam)
- Second Exam date **TBA** (Probably after two weeks of first exam)

# Course Outline Part B:

## Large-Scale Data Management and Analysis

**12 Distributed Stream Processing**

**13 Distributed Machine Learning Systems**

**11 Distributed Data-Parallel Computation**

**10 Distributed Data Storage**

Compute/  
Storage

**09 Cloud Resource Management and Scheduling**

**08 Cloud Computing Fundamentals**

Infra

# Agenda

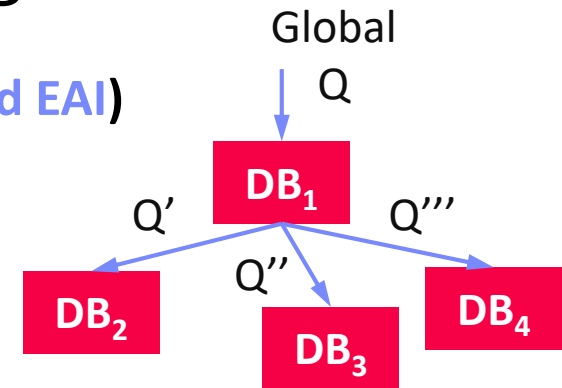
- **Motivation and Terminology**
- **Object Stores and Distributed File Systems**
- **Key-Value Stores and Cloud DBMS**

# Motivation and Terminology

# Overview Distributed Data Storage

## Recap: Distributed DBS (03 Replication, MoM, and EAI)

- **Distributed DB:** Virtual (logical) DB, appears like a local DB but consists of multiple physical DBs
- Components for global query processing
- **Virtual DBS** (homo.) vs **federated DBS** (hetero.)



## Cloud and Distributed Data Storage

- **Motivation:** **size** (large-scale), **semi-structured**/nested , **fault tolerance**
- **#1 Cloud and Distributed Storage**
  - **Block storage:** files split into blocks, read/write (e.g., SAN, AWS EBS)
  - **Object storage:** objects of limited size (e.g., 5TB), get/put (e.g., AWS S3)
  - **Distributed file systems:** file system on block/object stores (NFS, HDFS)
- **#2 Database as a Service**
  - **NoSQL stores:** Key-value stores, document stores
  - **Cloud DBMSs** (SQL, for OLTP and OLAP workloads)

# Central Data Abstractions

## ■ #1 Files and Objects

- **File:** Arbitrarily large sequential data in specific file format (CSV, binary, etc)
- **Object:** binary large object, with certain meta data

## ■ #2 Distributed Collections

- Logical multi-set (**bag**) of **key-value pairs** (**unsorted collection**)
- Different physical representations
- **Easy distribution** of pairs via horizontal partitioning (aka shards, partitions)
- Can be created from single file, or directory of files (unsorted)

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

# Data Lakes

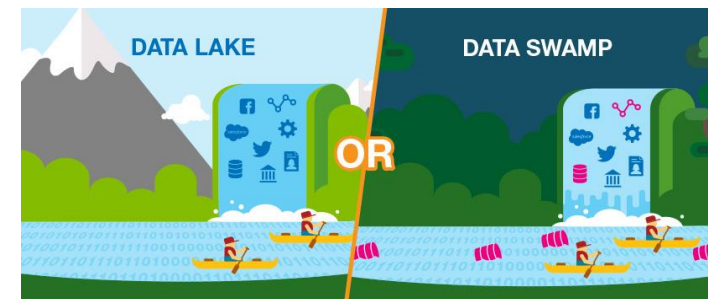
## ■ Concept “Data Lake”

- **Store massive amounts of un/semi-structured, and structured data** (append only, no update in place)
- **No need for architected schema** or upfront costs (unknown analysis)
- Typically: file storage in open, raw formats (inputs and intermediates)
- ➔ **Distributed storage and analytics** for scalability and agility

## ■ Criticism: Data Swamp

- Low data quality (lack of schema, integrity constraints, validation)
- Missing meta data (context) and data catalog for search

➔ **Requires proper data curation / tools**  
According to priorities (data governance)



[Credit: [www.collibra.com](http://www.collibra.com)]





# Object Stores and Distributed File Systems

# Object Storage

## Recap: Key-Value Stores

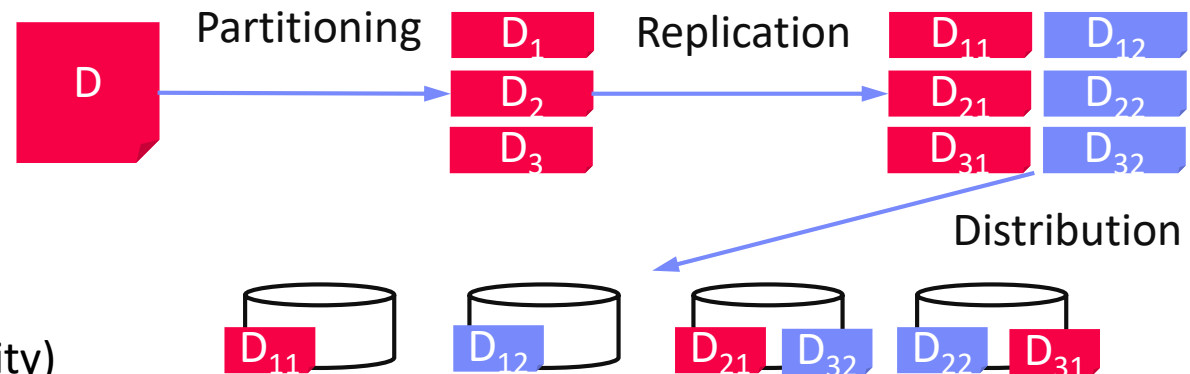
- **Key-value** mapping, where values can be of a variety of data types
- APIs for CRUD operations; scalability via sharding (**objects** or object segments)

## Object Store

- Similar to key-value stores, but: **optimized for large objects in GBs and TBs**
- Object identifier (**key**), **meta data**, and object as binary large object (**BLOB**)
- APIs: often REST APIs, SDKs, sometimes implementation of DFS APIs

## Key Techniques

- Partitioning
- Replication & Distribution
- Erasure Coding (partitioning + parity)



# Object Storage, cont.

## ■ Example Object Stores / Protocols

- Amazon Simple Storage Service (S3)
- OpenStack Object Storage (Swift)
- IBM Object Storage
- Microsoft Azure Blob Storage



## ■ Example Amazon S3

- Reliable object store for photos, videos, documents or any binary data
- **Bucket:** Uniquely named, static data container  
`http://s3.amazonaws.com/<identifier>`
- **Object:** key, version ID, value, metadata, access control
- 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)
- Apache Hive (SQL), Pig (ETL), Mahout/SystemML (ML), Giraph (Graph)

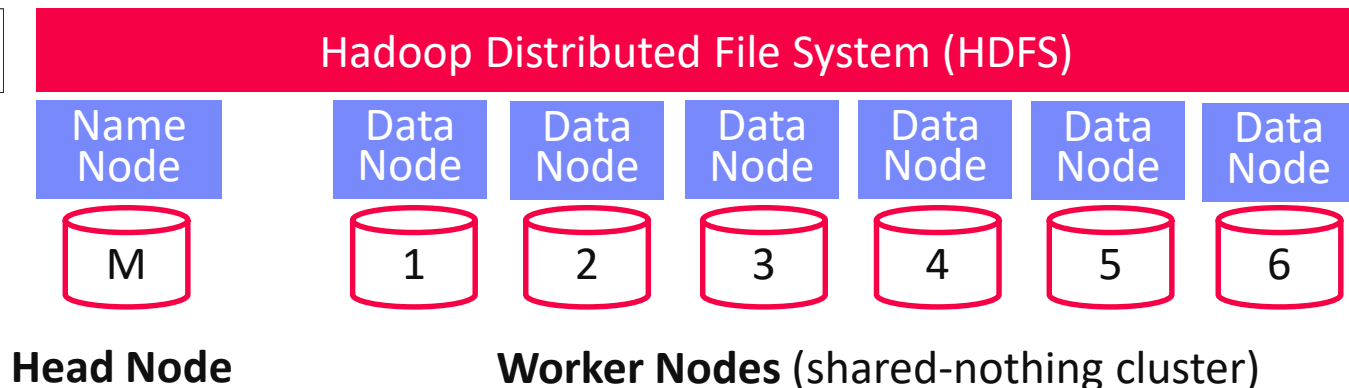
[Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung: **The Google file system**. SOSP 2003]



## 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

**Client**



# HDFS Daemon Processes

## ■ HDFS NameNode

- Master daemon that manages file system namespace and access by clients
- Metadata for all files (e.g., replication, permissions, sizes, block ids, etc)
- FSImage**: checkpoint of FS namespace
- EditLog**: **write-ahead-log (WAL)** of file write operations (merged on startup)

```
hadoop fs -ls ./data/mnist1m.bin
```

```
rw-r--r-- 3 mboehm hdfs 104510159 2018-10-20 22:59 /user/mboehm/data/mnist1m.bin/0-m-00000
rw-r--r-- 3 mboehm hdfs 137887319 2018-10-20 22:59 /user/mboehm/data/mnist1m.bin/0-m-00001
rw-r--r-- 3 mboehm hdfs 139012247 2018-10-20 22:59 /user/mboehm/data/mnist1m.bin/0-m-00002
rw-r--r-- 3 mboehm hdfs 139123247 2018-10-20 22:59 /user/mboehm/data/mnist1m.bin/0-m-00003
rw-r--r-- 3 mboehm hdfs 139053743 2018-10-20 22:59 /user/mboehm/data/mnist1m.bin/0-m-00004
rw-r--r-- 3 mboehm hdfs 138928955 2018-10-20 22:59 /user/mboehm/data/mnist1m.bin/0-m-00005
rw-r--r-- 3 mboehm hdfs 139016375 2018-10-20 22:59 /user/mboehm/data/mnist1m.bin/0-m-00006
rw-r--r-- 3 mboehm hdfs 139047923 2018-10-20 22:59 /user/mboehm/data/mnist1m.bin/0-m-00007
rw-r--r-- 3 mboehm hdfs 139042307 2018-10-20 22:59 /user/mboehm/data/mnist1m.bin/0-m-00008
rw-r--r-- 3 mboehm hdfs 139068143 2018-10-20 22:59 /user/mboehm/data/mnist1m.bin/0-m-00009
rw-r--r-- 3 mboehm hdfs 139029875 2018-10-20 22:59 /user/mboehm/data/mnist1m.bin/0-m-00010
rw-r--r-- 3 mboehm hdfs 138901043 2018-10-20 22:59 /user/mboehm/data/mnist1m.bin/0-m-00011
rw-r--r-- 3 mboehm hdfs 139042763 2018-10-20 22:59 /user/mboehm/data/mnist1m.bin/0-m-00012
rw-r--r-- 3 mboehm hdfs 139030751 2018-10-20 22:59 /user/mboehm/data/mnist1m.bin/0-m-00013
rw-r--r-- 3 mboehm hdfs 139172051 2018-10-20 22:59 /user/mboehm/data/mnist1m.bin/0-m-00014
rw-r--r-- 3 mboehm hdfs 138962735 2018-10-20 22:59 /user/mboehm/data/mnist1m.bin/0-m-00015
rw-r--r-- 3 mboehm hdfs 139079495 2018-10-20 22:59 /user/mboehm/data/mnist1m.bin/0-m-00016
rw-r--r-- 3 mboehm hdfs 63417008 2018-10-20 22:59 /user/mboehm/data/mnist1m.bin/0-m-00017
```

## ■ 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**: implements access to distributed collections in files
- **Split**: record-aligned block of file (aligned with HDFS block size)
- **RecordReader**: API for reading key-value pairs from file splits
- Examples: FileInputFormat, TextInputFormat, SequenceFileInputFormat

## Example Text Read

```
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
}
```

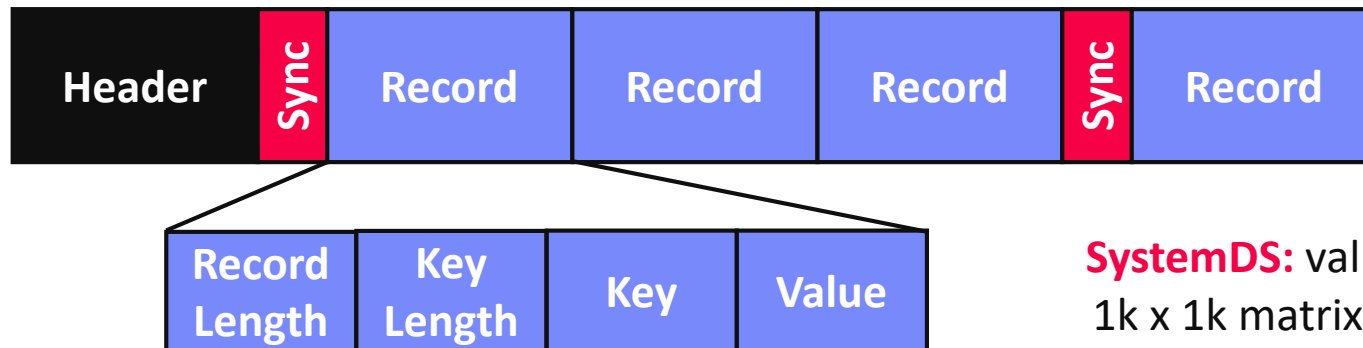
# HDFS InputFormats and RecordReaders, cont.

## Sequence Files

- Binary files for key/value pairs, w/ optional compression (MapReduce/Spark inputs/outputs, MapReduce intermediates)
- InputFormat with readers, writers, and sorters

## Example Uncompressed SequenceFile

- Header:** SEQ+version (4 bytes), keyClassName, valueClassName, compression, blockCompression, compressor class (codec), meta data
- Splittable binary representation of key-value pair collection



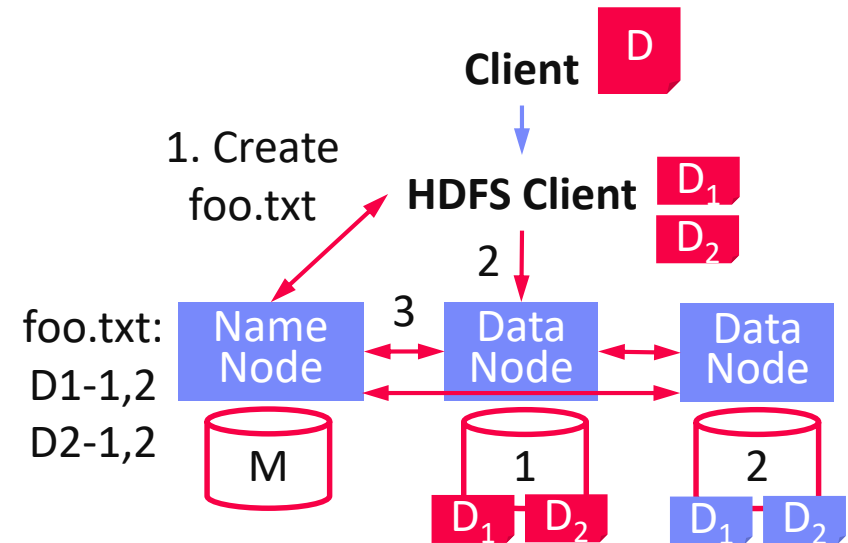
**SystemDS:** values are 1k x 1k matrix blocks



# HDFS Write and Read

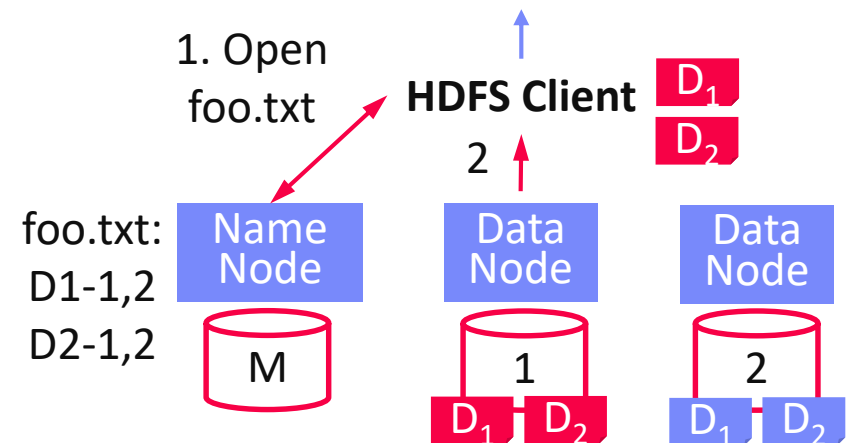
## HDFS Write

- #1 Client RPC to NameNode to create file → lease/replica DNs
- #2 Write blocks to DNs, pipelined replication to other DNs
- #3 DNs report to NN via heartbeat



## HDFS Read

- #1 Client RPC to NameNode to open file → DNs for blocks
- #2 Read blocks sequentially from closest DN w/ block
- InputFormats and RecordReaders as abstraction for multi-part files (incl. compression/encryption)



# HDFS Data Locality

## ■ Data Locality

- **HDFS is generally rack-aware** (node-local, rack-local, other)
- Schedule reads from closest data node
- **Replica placement** (rep 3): local DN, other-rack DN, same-rack DN
- MapReduce/Spark: locality-aware execution (**function vs data shipping**)

## ■ Custom Locality Information

- Custom **InputFormat** and **FileSplit** implementations
- Return customized mapping of locations on `getLocations()`
- Can use block locations of arbitrary files

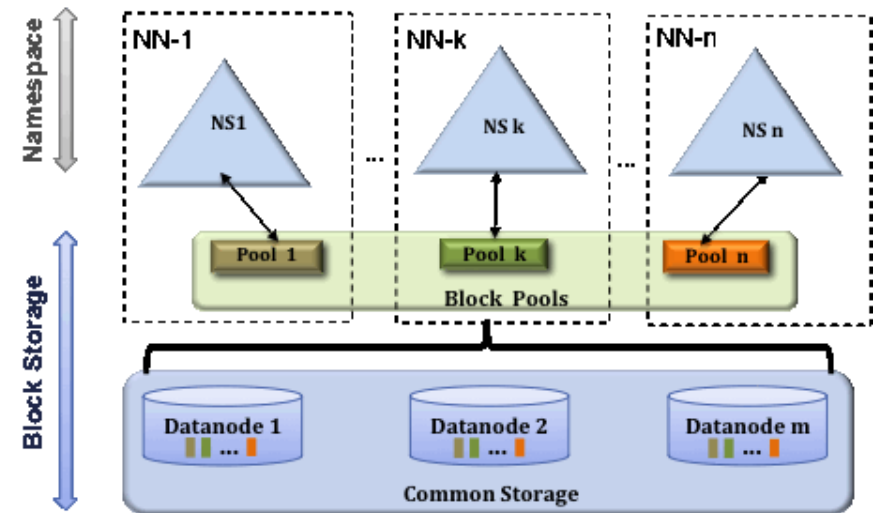
```
public class MyFileSplit extends FileSplit
{
    public MyFileSplit(FileSplit x, ...) {}
    @Override
    public String[] getLocations() {
        return new String[]{"node1", "node7"};
    }
}
```

```
FileStatus st = fs.getFileStatus(new Path(fname));
BlockLocation[] tmp1 = fs.getFileBlockLocations(st, 0, st.getLen());
```

# HDFS Federated NameNodes

## ■ HDFS Federation

- Eliminate NameNode as namespace scalability bottleneck
- Independent NameNodes, responsible for name spaces
- DataNodes store blocks of all NameNodes
- Client-side mount tables



[Credit: <https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/Federation.html>]

## ■ GFS Multiple Cells

- *"We also ended up doing what we call a "multi-cell" approach, which basically made it possible to put multiple GFS masters on top of a pool of chunkservers."*  
-- Sean Quinlan

[Kirk McKusick, Sean Quinlan:  
GFS: evolution on fast-forward.  
Commun. **ACM 53(3)** 2010]



# Other DFS

## ■ HDFS FileSystem Implementations (subset)

- LocalFileSystem (**file**), DistributedFileSystem (**hdfs**)
- FTPFileSystem, HttpFileSystem, ViewFilesystem (ViewFs – mount table)
- NativeS3FileSystem (**s3**, **s3a**), NativeSwiftFileSystem, NativeAzureFileSystem
- Other proprietary: IBM **GPFS**, Databricks FS (DBFS)

## ■ Google Colossus

- More fine-grained accesses, Google Cloud Storage

[WIRED: Google Remakes  
Online Empire With 'Colossus',  
<https://www.wired.com/2012/07/google-colossus/>]

## ■ High-Performance Computing

- **Scope**: Focus on high IOPs (instead of bandwidth) with block write
- IBM **GPFS** (General Parallel File System) / Spectrum Scale
- **BeeGFS** (Fraunhofer GFS) – focus on usability, storage/metadata servers
- **Lustre** (Linux + Cluster) – GPL license, LNET protocol / metadata / object storage
- RedHat **GFS2** (Global File System) – Linux cluster file system, close to local
- **NAS** (Network Attached Storage), **SAN** (Storage Area Network)
- **GekkoFS** (Uni Mainz / Barcelona SC) – data-intensive HPC applications

# Key-Value Stores and Cloud DBMS

# Motivation and Terminology

## ■ Motivation

- **Basic key-value mapping via simple API** (more complex data models can be mapped to key-value representations)
- **Reliability at massive scale on commodity HW** (cloud computing)

## ■ System Architecture

- **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)

users:1:a

“Inffeldgasse 13, Graz”

users:1:b

“[12, 34, 45, 67, 89]”

users:2:a

“Mandellstraße 12, Graz”

users:2:b

“[12, 212, 3212, 43212]”

## ■ Example Systems

- **Dynamo** (2007, AP) → **Amazon DynamoDB** (2012)
- **Redis** (2009, CP/AP)



redis



[Giuseppe DeCandia et al: Dynamo: amazon's highly available **key-value store**. SOSP 2007]



# Example Systems: Dynamo

[Giuseppe DeCandia et al:  
Dynamo: amazon's highly available  
**key-value store**. SOSP 2007]



## ■ Motivation

- **Simple, highly-available** data storage for small objects in ~1MB range
- Aim for **good load balance** (99.9<sup>th</sup> percentile SLAs)

## ■ #1 System Interface

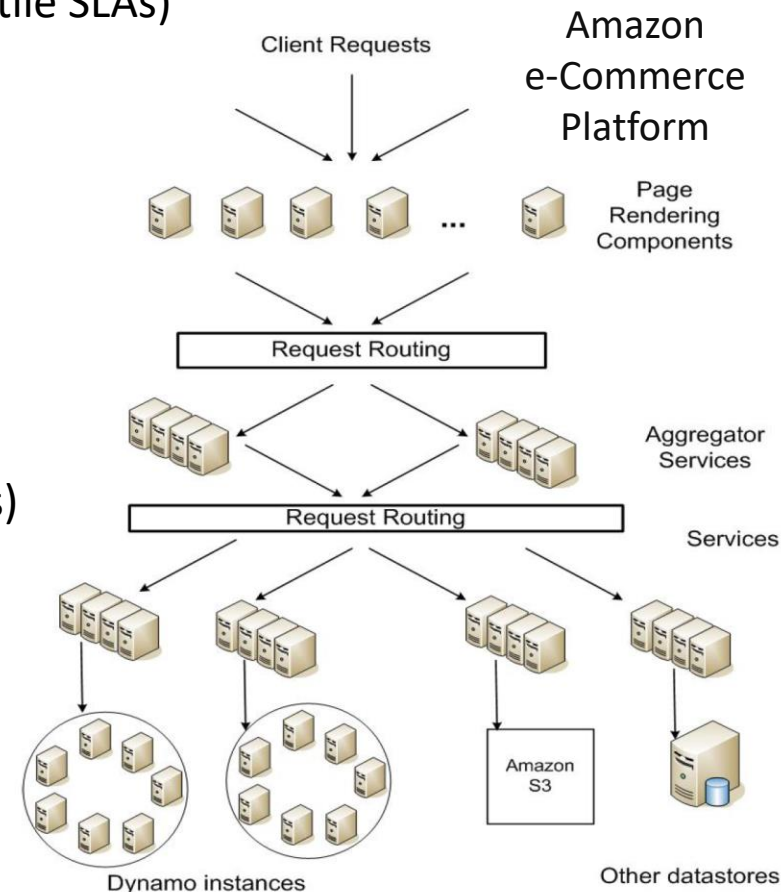
- Simple get(k, ctx) and put(k, ctx) ops

## ■ #2 Partitioning

- **Consistent hashing** of nodes and keys on circular ring for **incremental scaling**
- Nodes hold **multiple virtual nodes** for **load balance** (add/rm, heterogeneous)

## ■ #3 Replication

- Each data item **replicated N times** (at coord node and N-1 successors)
- Eventual consistency with async update propagation based on **vector clocks**
- Replica synchronization via **Merkle trees**



# Example Systems, cont.

## Redis Data Types



- Redis is not a plain KV-store, but “data structure server” with persistent log (**appendfsync no/everysec/always**)
- Key:** ASCII string (max 512MB, common key schemes: comment:1234:reply.to)
- Values:** strings, lists, sets, sorted sets, hashes (map of string-string), etc

## Redis APIs

- SET/GET/DEL:** insert a key-value pair, lookup value by key, or delete by key
- MSET/MGET:** insert or lookup multiple keys at once
- INCRBY/DECBY:** increment/decrement counters
- Others: EXISTS, LPUSH, LPOP, LRANGE, LTRIM, LLEN, etc

## Other systems

- Classic KV stores (AP): **Riak**, **Aerospike**, **Voldemort**, **LevelDB**, **RocksDB**, **FoundationDB**, **Memcached**
- Wide-column stores: **Google BigTable** (CP), **Apache HBase** (CP), **Apache Cassandra** (AP)



LEVELDB





# Log-structured Merge Trees

[Patrick E. O'Neil, Edward Cheng, Dieter Gawlick, Elizabeth J. O'Neil: The Log-Structured Merge-Tree (LSM-Tree). *Acta Inf.* 1996]

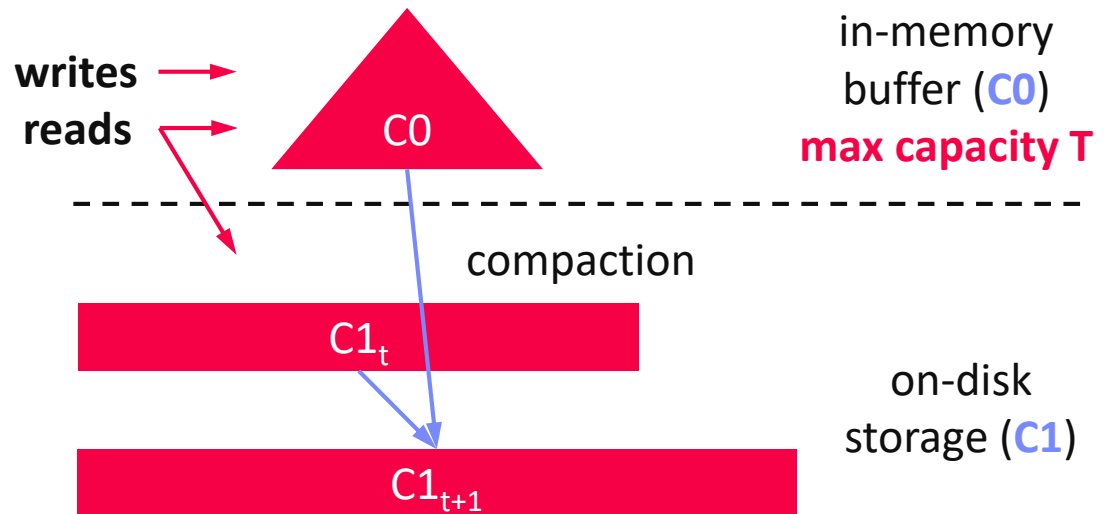


## LSM Overview

- Many KV-stores rely on LSM-trees as their storage engine (e.g., [BigTable](#), [DynamoDB](#), [LevelDB](#), [Riak](#), [RocksDB](#), [Cassandra](#), [HBase](#))
- Approach:** Buffers writes in memory, flushes data as sorted runs to storage, merges runs into larger runs of next level (compaction)

## System Architecture

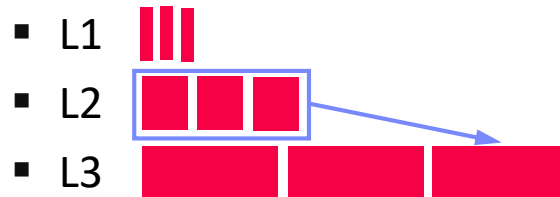
- Writes in C0
- Reads against C0 and C1 (w/ buffer for C1)
- Compaction (rolling merge): sort, merge, including [deduplication](#)



# Log-structured Merge Trees, cont.

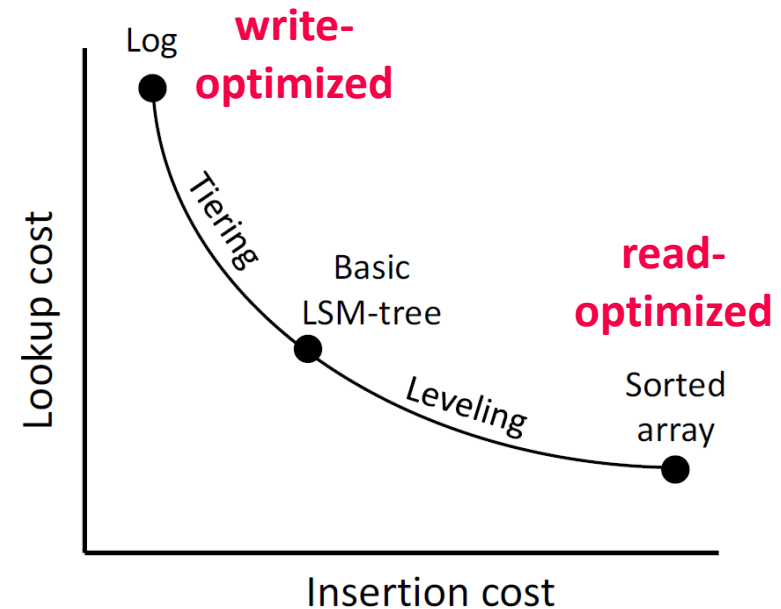
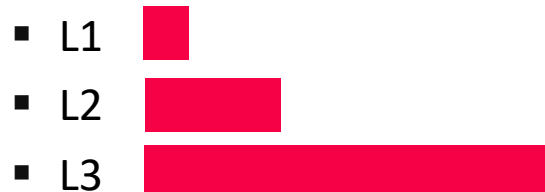
## ■ LSM Tiering (write optimized)

- Keep up to  $T-1$  runs per level  $L$
- Merge all runs of  $L_i$  into 1 run of  $L_{i+1}$



## ■ LSM Leveling (read optimized)

- Keep 1 run per level  $L$
- Merge run of  $L_i$  with  $L_{i+1}$



[Niv Dayan: Log-Structured-Merge Trees, **Comp115** guest lecture, 2017]



[Stratos Idreos, Mark Callaghan: Key-Value Storage Engines (Tutorial), **SIGMOD 2020**]



# Cloud Databases (DBaaS)

## ■ Motivation DBaaS

- Simplified setup, maintenance, tuning and auto scaling
- Multi-tenant systems (scalability, learning opportunities)
- Different types based on workload (OLTP vs OLAP)



## ■ Elastic Data Warehouses

- Motivation: Intersection of data warehousing (**02 DWH, ETL, SQL/OLAP**), cloud computing (**07/08 Cloud Computing**), Distributed Storage (**09 today**)
- Example Systems
  - **#1** Snowflake
  - **#2** Google BigQuery (Dremel)
  - **#3** Amazon Redshift
  - Azure SQL Data Warehouse

**Commonalities:**  
SQL, **column stores**,  
data on **object store / DFS**,  
**elastic cloud** scaling

# Example Snowflake

[Benoît Dageville et al.: The Snowflake Elastic Data Warehouse. **SIGMOD 2016**]



- **Motivation** (impl started late 2012)
  - Enterprise-ready DWH solution for the cloud (elasticity, semi-structured)
  - Pure SaaS experience, high availability, cost efficient

## Cloud Services

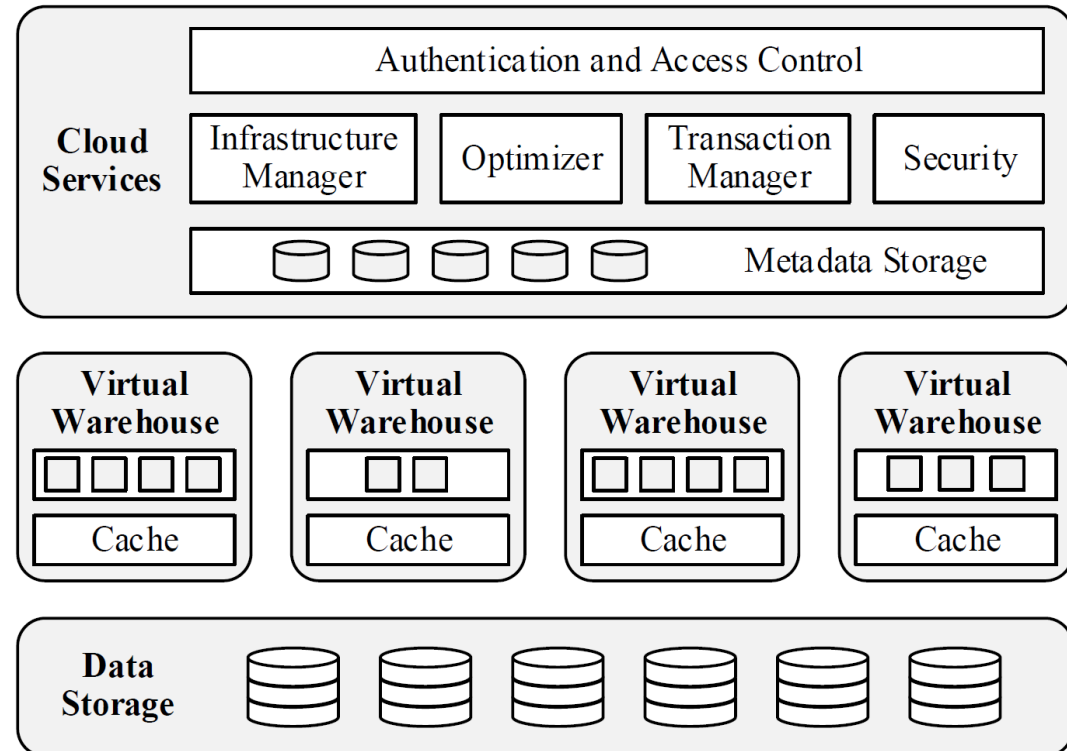
- Manage virtual DHWs, TXs, and queries
- Meta data and catalogs

## Virtual Warehouses

- Query execution in EC2
- Caching/intermediates

## Data Storage

- Storage in AWS S3
- PAX / hybrid columnar
- Min-max pruning



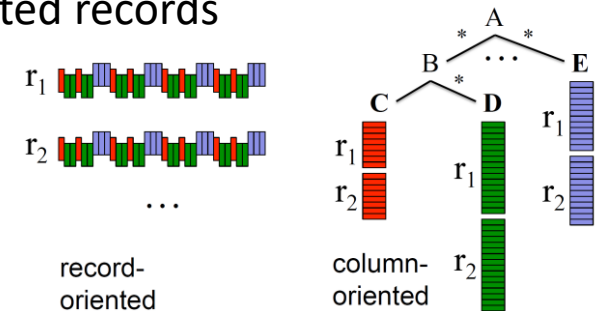
# Example Google BigQuery

[Sergey Melnik et al.: Dremel: Interactive Analysis of Web-Scale Datasets. **PVLDB 3(1) 2010**]



## Background Dremel

- Scalable and fast **in-situ analysis of read-only nested data** (DFS, BigTable)
- Data model:** protocol buffers - strongly-typed nested records
- Storage model:** **columnar storage of nested data** (efficient splitting and assembly records)
- Query execution via **multi-level serving tree**

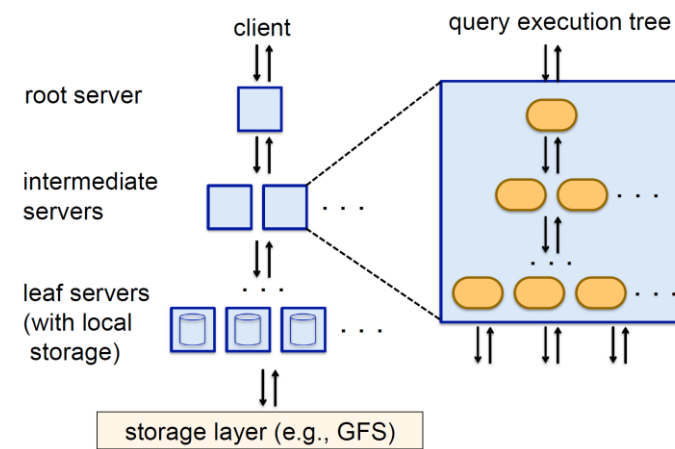


## BigQuery System Architecture

- Public impl of internal Dremel system (2012)
- SQL over structured, nested data (OLAP, BI)
- Extensions:** web Uis, REST APIs and ML
- Data storage:** Colossus (**NextGen GFS**)



[Kazunori Sato: An Inside Look at Google BigQuery, Google BigQuery White Paper 2012.]



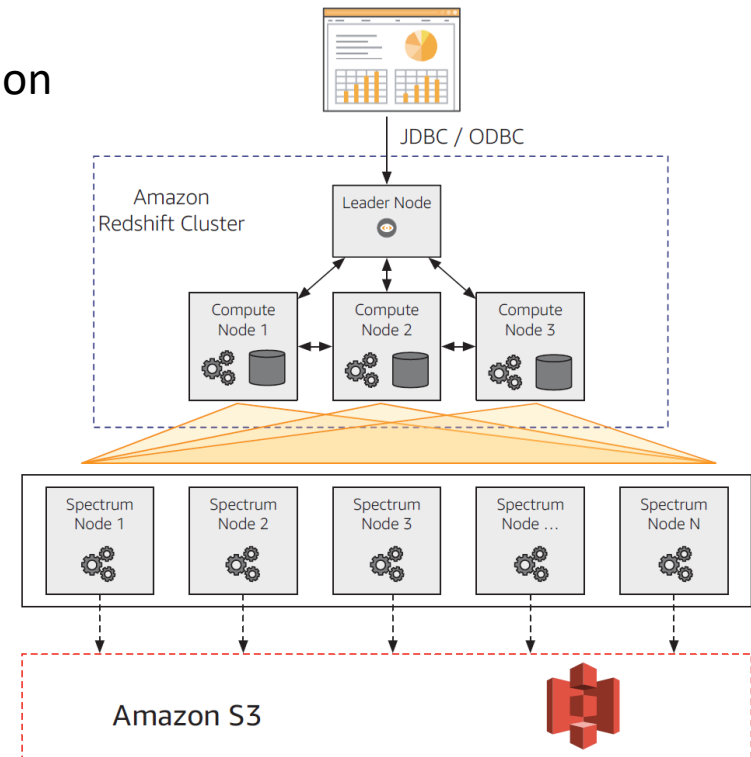
# Example Amazon Redshift

- **Motivation** (release 02/2013)
  - **Simplicity and cost-effectiveness**  
(fully-managed DWH at petabyte scale)
- **System Architecture**
  - **Data plane:** data storage and SQL execution
  - **Control plane:** workflows for monitoring, and managing databases, AWS services
- **Data Plane**
  - Initial engine licensed from ParAccel
  - Leader node + sliced compute nodes in **EC2** (with **local storage**)
  - Replication across nodes + **S3 backup**
  - **Query compilation** in C++ code
  - Support for **flat and nested files**

[Anurag Gupta et al.: Amazon Redshift and the Case for Simpler Data Warehouses. **SIGMOD 2015**]



[Mengchu Cai et al.: Integrated Querying of SQL database data and S3 data in Amazon Redshift. **IEEE Data Eng. Bull.** 41(2) 2018]



# Summary and Q&A

- **Motivation and Terminology**
- **Object Stores and Distributed File Systems**
- **Key-Value Stores and Cloud DBMS**
  
- **Next Lectures**
  - **11 Distributed, Data-Parallel Computation** [Jan 12]
  - **12 Distributed Stream Processing** [Jan 19]
  - **13 Distributed Machine Learning Systems** [Jan 19]