Managing and Processing (Big) Scientific Data

(MPBSD)

Nov 10, 2021

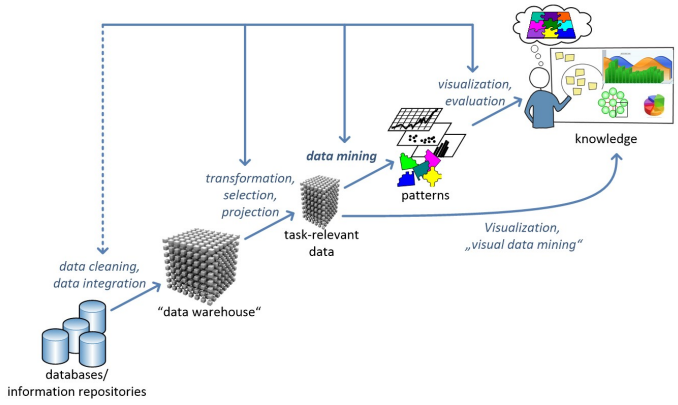
Prof. Dr. Peer Kröger
Department for Computer Science
Information Systems and Data Mining Group

Welcome

- This lecture aims at giving you a sort of big picture rather than details:
  What is generally out there to manage scientific data and what are potential pitfalls?
- If you have questions during the presentation, please feel free to interrupt at any time
- If you are interested in more details, please feel free to contact me bilateral:
  pkr@informatik.uni-kiel.de

BTW: Why care?

As simple as that: before you can the magic stuff, you need to manage the data



The Knowledge Discovery in Databases Process following Fayyad, Piatetsky-Shapiro, Smyth (1996)

Agenda

1. What is Big Data?
2. Relational Database Management Systems
3. Managing Spatial (/Geo) Data
4. Managing Big Data
5. Distributed Data Processing

Big Data is a Buzzword

- Term first triggered by McKinsey in 2011
  - business perspective (main focus)
  - technical perspective
- Indeed, today, people pay with data
  - e.g., Facebook, Google, Twitter, Amazon
    - "use service" means "provide data"
    - data is used to "improve" service target advertisement (user pays indirectly)
    - data is sold
- Over the years, other terms have emerged or have been re-discovered and are used sometimes as synonyms:
  - Data Science, Artificial Intelligence
  - Business Intelligence, Data Mining, Machine Learning, ...
  - Industry 4.0, IoT, ...

McKinsey Global Institute, Big data: The next frontier for innovation, competition, and productivity, June 2011

Image source: The New York Times: https://www.nytimes.com/2018/03/06/business/economy/user-data-pay.html

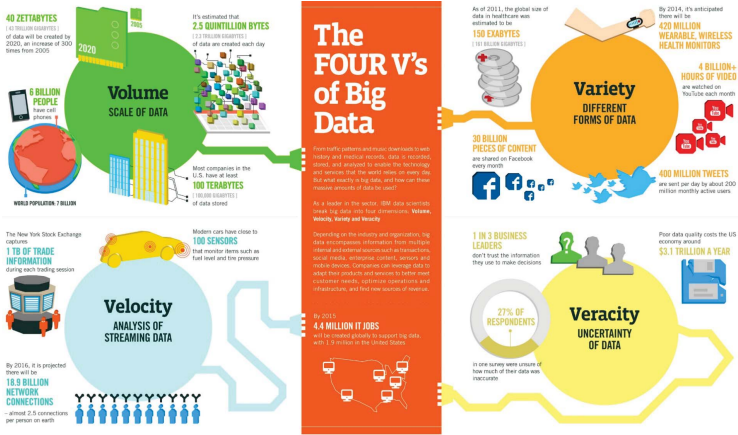
What Big Data is — is Up to You!

Big Data is characterized by some "V's"[1], e.g. four:

- Volume: data from many sources
  - volume on disk
  - number of instances (in tables: rows) or features (in tables: columns)
- Velocity: data is changing/new data is arriving (potentially at a high pace)
  - sensors constantly produce data
  - communication is constantly going on
- Variety: not all data is the same
  - data can have different structures beyond tables: vectors, sequences, graphs, tensors
  - different sources rely on different formats
- Veracity: the meaning of the data is unsecure
  - inputs may be noisy, manipulated or misinterpreted
  - consider data objects as samples not facts (cf. "learning" above)

[1] The original definition by McKinsey lists three V's: volume, velocity, variety

My Favorite Big Data Meme (Four V's)

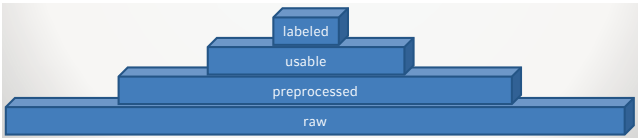


Picture from: IBM (no, I do not get any money from them, I just like the pic)

From Data to Knowledge

The Data Pyramid:

- Raw data is often big (somehow), OK
- BUT during the Data Science Process (see above) data shrinks
- AND FINALLY, for complex tasks, high-quality data is often still small (e.g. not enough labels, noise, irrelevant, too high resolution)



The Data Pyramid

Consequences:

- Analyses still run in RAM (Python/R scripts, Viz tools, ...
- Big Data systems often support the first step in the process

Agenda

1. What is Big Data?
2. Relational Database Management Systems
3. Managing Spatial (/Geo) Data
4. Managing Big Data
5. Distributed Data Processing

Data Management in the Pre-Big Data Era

Some history:

- 60s: IBM developed the Hierarchical Database Model
  - Tree-like structure
  - Data stored as records connected by links

(Forget about it; just in case, if your grumpy grandpa is telling stories!)

- Mid 80's: Rise of Relational Database Model
  - Data stored in a collection of tables (rows and columns)
  - Strict relational schemas (first Normal Form!)
  - SQL became standard language (based on relational algebra)

- Relational Database Management Systems (RelDBMS) became (and still are) the *de facto* standard for managing cooperate data

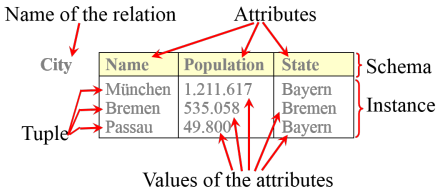
Properties of RelDBMS

- RelDBMS have been designed for transaction data (daily business of companies), e.g. financial transactions, flight booking, order transaction, etc.
  - data consistency, availability, and fault-tolerance are most important
  - short read/write operations on the data
  - scale: usually in the range of MBytes-GBytes
- Advantages:
  - "Data independence" improves flexibility over file systems
  - Advanced transaction concept ("ACID" properties) implement e.g.
    - synchronization of multiple users
    - back-up/ and recovery strategies
  - Automatic/transparent query optimization
  - ...

Relational Data Model: Basics

- The relational model uses tables as the one and only way to structure data
- A table represents objects with a given set of features
- Rows = objects / entries / tuples / instances
- Columns = attributes / features / variables (Statistics)

- Columns have atomic data types (fixed length in terms of bits)
- Keys are identifiers for rows (see next)
- Complex relationships between objects are modeled through references between the corresponding tables (foreign keys)



Relational Data Model: Keys

- Each row must be uniquely identifiable
- Different to programming languages (objects are identified by identifiers/references), in the relational model, rows are identified by attribute values
- One or more attributes must be flagged as *key*
- Values of these attributes must be unique
- Keys identify rows uniquely
- Keys can be used to refer to (unique) rows from other tables:

Mitarbeiter

PNr	Name	Vorname	Abteilung
001	Huber	Erwin	01
002	Mayer	Hugo	01
003	Müller	Anton	02

Abteilungen

ANr	Abteilungsname
01	Buchhaltung
02	Produktion
03	Marketing

Here, we use the key of the table Abteilungen (= division), "ANr", to link each entry in the table Mitarbeiter (= employee) to its corresponding division

Relational Data Model: Normalization

- Let's consider this table:

Lectures

LectureNr	**Title**	**Term**
012	Java 1-0-1	Summer 2018
013	Machine Learning	Winter 2017
013	Machine Learning	Winter 2018

- **LectureNr** and **Term** are the key attributes (we need both to be unique)
- What is the problem?
- We need to store the title of a lecture several times (here "Machine Learning"):
  For any two rows r_1 and r_2 it holds: if the values of r_1 and r_2 in LectureNr are the same, the values in Title of r_1 and r_2 must be identical, too
- We call this a functional dependency ("FD"), here Title depends on LectureNr)

Relational Data Model: Normalization

- Now consider, we want to change the name to e.g. "Introduction to ML"
- Obviously, we need to do that multiple times in order to be consistent
- If we do not know, that the title is redundant, we might miss some tuples, i.e., we get inconsistent data

Lectures

LectureNr	**Title**	**Term**
012	Java 1-0-1	Summer 2018
013	Machine Learning	Winter 2017
⋮	⋮	⋮
013	Machine Learning	Winter 2018
⋮	⋮	⋮
013	Machine Learning	Winter 2019

- So, redundancies waste storage (Huhh, now really???)
- But the real problem is, they cause anomalies, like the "insert"-anomaly mentioned above

Relational Data Model: Normalization

- By the definition of keys, any entire row of a relation is always functionally dependent on the key
- These FDs are welcome and do not cause redundancy (keys are used to uniquely identify tuples!)
- However, there may be more FDs:
  - an attribute may be functional dependent on only a part of a key (if the key consists of more than one attribute), e.g. Title depends on LectureNr
    this is called a partial FD
  - an attribute may be functional dependent on an attribute that is not part of the key
    this is called transitive FD
- Normalization gets rid of these FDs that are not related to the properties of keys because they can cause anomalies (due to redundancies)
- In theory, there may be more FDs to take care of, but in practice, only the (aforementioned) FDs are erased

Relational Data Model: Normalization

- We can fix the problem from above by splitting the table (which suffers from a partial FD) into two ones:

Lectures

LectureNr	**Title**
012	Java 1-0-1
013	Machine Learning
⋮	⋮

Lectures

LectureNr	**Term**
012	Summer 2018
013	Winter 2017
⋮	⋮
013	Winter 2018
⋮	⋮
013	Winter 2019

- Schemas do not have
  - any FDs of attributes on a part of any key candidate
  - any FDs of attributes on non-key candidate attributes

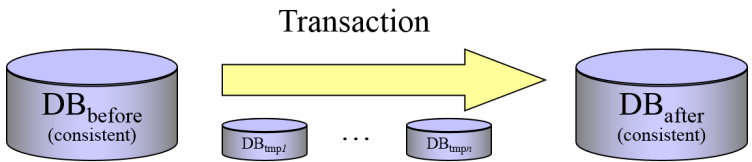
  are called "3NF" schemas (because they meet a formally defined set of "normal forms" — three in that case)

Just a short word on SQL

- The Structured Query Language (SQL) is the programming language of relational DBMS
- Super mature API to
  - Define data structures (schema)
  - Store/manage data (insert/delete)
  - Manipulate and query data
- To be good in using SQL, you need approx. 2 weeks and then continuous practice

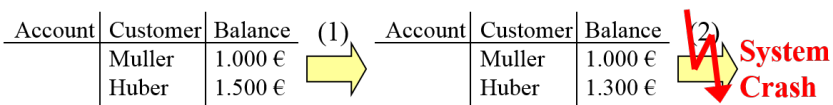
Transaction Concept

- The key to synchronizing multiple users and availability in case of errors/crashes is the transaction concept
- Typically, a user interaction (program, session, ...) is a transaction
- Thus, a transaction is a sequence of read/write operations that preserves a consistent database state
- The transaction management of a DBMS has two main tasks to do:
  - Synchronization: coordination of several users
  - Recovery: solution of crashes and other errors



Transaction Concept

- Example: Crash (at the bank)
  - Transfer 200.- EUR from the account of Huber to the account of Muller
  - Possible schedule:
    1. Decrease the balance of Huber by 200.- EUR
    2. Increase the balance of Muller by 200.- EUR
  - Possible scenario with this schedule:



Transaction Concept

- Example: Uncoordinated Users (at the airport)
  - Flight operations (FO) prints out the names of passengers and the number of passengers on flight LH001
  - Check-in (CI) checks-in another passenger (one of these late arriving HONs — I am really jealous because of her status)
  - Possible schedule:
    1. FO: prints out list of names
    2. CI: checks passenger "Phantomime" in
    3. FO: prints out number of passengers
  - What went wrong?
  - FO gets inconsistent information because "Phantomime" is not on the list but is counted

ACID

The ACID principle defines a set of properties for transactions taht should be guaranteed by a DBMS:

- Atomicity: a transaction either runs to completion or (if it does not complete) has no effect at all
- Consistency: a transaction preserves any integrity constraints
- Isolation: even though transactions are executed concurrently (for performance reasons), the effect of each transaction must be the same as if they have been operated serially (without concurrency)
- Durability: the effect of a transaction that successfully completes (gets a “commit” from the system) must remain in the database even in case of a later crash, error, etc.

The Big Picture

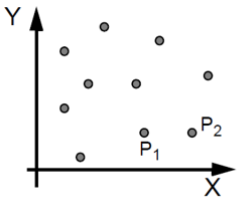
- Relational DBMS are the de facto standard in data management
- RelDBMS offer a mature and powerful technology for managing cooperate data (superior to simple file system-based solutions)
- Data is modeled in tables (just like Excel tables)
- Consider normalization for non-redundant, anomaly-free storage
- ACID is the “holy grail” of RelDBMS, it ensures a secure, consistent and available data source under multiple users with parallel access
- SQL is a powerful standard API for data management (modeling, storing, retrieving)

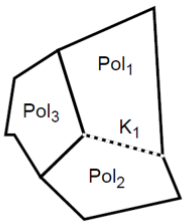
Agenda

1. What is Big Data?
2. Relational Database Management Systems
3. Managing Spatial (/Geo) Data
4. Managing Big Data
5. Distributed Data Processing

Spatial/Geo Data

- Geo objects ...
  - have spatial coordinates — usually 2D, sometimes 3D (e.g. a.o.s.l)
  - may have a temporal dimension
  - usually have additional (non-spatial) attributes
- Types of attributes
  - Spatial (coordinates of a point, area of a polygon, ...)
  - Topological (polygon 1 and polygon 2 are "neighbors", have a common face, ...)
  - Thematic (polygon 1 is a cornfield, point 2 is the city hall, ...)

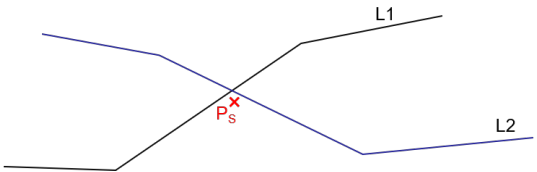




Spatial Coordinates and Realms

Challenge of the Euclidean space (here 2D):

- Point coordinates: (x, y) where $x, y \in \mathbb{R}$
- Computers store real numbers as a so called "floating point number" (float) with a specific precision
- Not all real numbers can be represented as float ...
- Problem: the intersection (P_S) of two lines ($L1$ and $L2$) may not be representable by floats

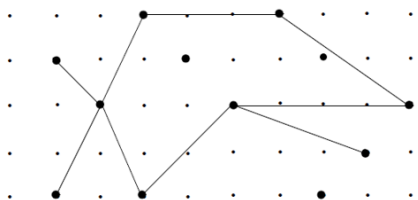


- Work-around: interpolation to the next float produces inconsistencies (P_S is not part of $L1$ nor $L2$)

Spatial Coordinates and Realms

Solution: Realms (Schneider, Güting 1994)

- Given a set of points $PR \subseteq \mathbb{N} \times \mathbb{N}$ over a grid
- A realm is a set of points $P \subseteq PR$ and non-intersecting line segments S over PR
- Intersections of line segments must be on a grid point in PR
- A realm can be interpreted as a (planar) graph:
  - Nodes: points P
  - Edges: line segments S



Modeling Spatial Data

- In GIS, spatial data is usually mapped on a relational model, i.e. we store all information in tables
- There are many different ways to do that, e.g.
  - Spaghetti Model
    - Each object is a sequence of distinct point coordinates ((x, y), e.g. border points)
    - Each object is represented independently (strong redundancy!!!)
    - Simple model with no explicit topological properties (e.g. no holes in polygons)
  - TIGER Model: Topologically Integrated Geographic Encoding and Referencing Model (Marx, 1986)
    - Complex non-redundant/normalized model (point coordinates are referenced)
    - Different data types for points, lines between points, surface (closed set of lines), and cover (union of surfaces)
    - Problem (of normalization): high costs to get all information of more complex structures; result of a query does not return an "object" (e.g. line) but a set of entries (e.g. points)

Modeling Spatial Data

Example TIGER model:

Parzellen

OID	PolID	Q-Meter
Par1	Pol1	1200
Par2	Pol2	1435

Polygone

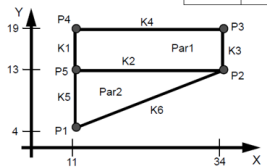
PolID	KID
Pol1	K1
Pol1	K2
Pol1	K3
Pol1	K4
Pol2	K2
Pol2	K5
Pol2	K6

Kanten

KID	Von	Nach
K1	P4	P5
K2	P5	P2
K3	P2	P3
K4	P3	P4
K5	P5	P1
K6	P1	P2

Punkte

PID	x	y
P1	11	4
P2	34	13
P3	34	19
P4	11	19
P5	11	13



Legend:

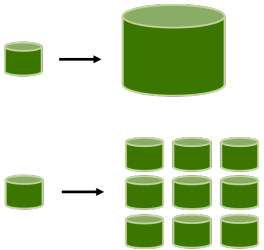
Parzelle = table of the covers
Polygone = table of the surfaces
Kanten = table of the lines
Punkte = table of the points

Agenda

1. What is Big Data?
2. Relational Database Management Systems
3. Managing Spatial (/Geo) Data
4. Managing Big Data
5. Distributed Data Processing

Some History (again)

- Rise of the Web 2.0 in the mid 2000's required DBMS to scale up
- Approaches
  - Vertical scaling: enlarge a single machine
    Limited in space, expansive
  - Horizontal scaling: use many machines to form clusters/grids
    Limited by cluster maintenance



- Horizontal scale won
- That was the birth of the NoSQL movement in the mid 2000's
  - Problem: RelDBMS do not scale well horizontally
  - Thus, big players like Google or Amazon developed their own storage systems
  - NoSQL ("Not-Only" SQL) databases were born
  - Today: Age of NoSQL: several different NoSQL systems available (> 250)

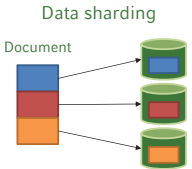
Characterstics of NoSQL Databases

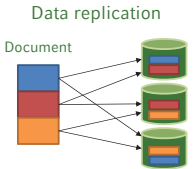
- There is no unique definition but rather some characteristics for NoSQL Databases:
  - Horizontal scalability (cluster-friendliness)
  - Non-relational
  - Distributed
  - Schema-less
  - Open-source (at least most of the systems)
- Most importantly: **ACID (the holy grail of RelDBS) is no more!!!**
- Rather: there is BASE — an artificial concept for NoSQL databases:
  - Basically Available: The system is generally available, but some data might not at any time (e.g. due to node failures)
  - Soft State: The system's state changes over time. Stale data may expire if not refreshed.
  - Eventual consistency: The system is consistent from time to time, but not always. Updates are propagated through the system if there is enough time.
- BASE is placed on the opposite site to ACID when considering a consistency-availability spectrum (see later)!

Consistency

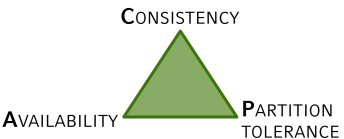
Two types of consistency corresponding to distribution type:

- Logical consistency (corresponds to sharing): data is consistent within itself (Data Integrity)
- Replication consistency: data is consistent across multiple replicas (on multiple machines)
- Consequence: CAP-theorem





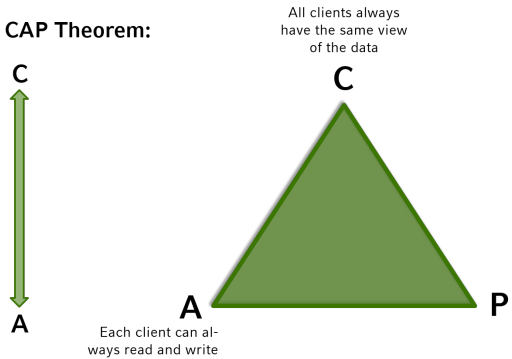
Brewer's CAP Theorem:



Any networked shared-data system can have at most two of the three desired properties!

The Big Picture

Thus, any partitioned system (horizontally scaled) needs to give up either perfect availability or perfect consistency:



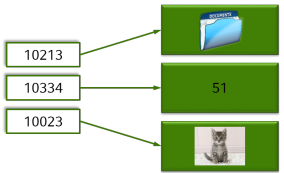
NoSQL Data Models

The four main NoSQL data models out there:

- Key/Value Stores
- Document Stores
- Wide Column Stores
- Graph Databases

Key Value Stores

- Most simple form of database systems
- Store key/value pairs and retrieve values by keys
- Values can be of arbitrary format



- There are very heterogeneous systems
  - Some systems support ordering of keys, which enables efficient querying, like range queries
  - Some systems support in-memory data maintenance, some use disks
- Examples: redis, Dynamo

Document Stores

- Store documents in form of XML or JSON
- Semi-structured data records that do not have a homogeneous structure
- Columns can have more than one value (arrays)
- Documents include internal structure, or metadata
- Data structure enables efficient use of indexes
- Examples: mongoDB, CouchDB

Wide Column Stores

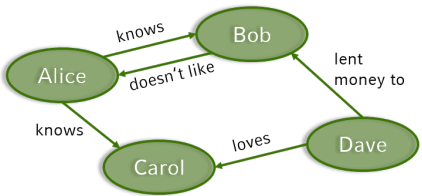
Wide Column Stores

- Rows are identified by keys
- Rows can have different numbers of columns (up to millions)
- Order of rows depend on key values (locality is important!)
- Multiple rows can be summarized to families (or tablets)
- Multiple families can be summarized to a key space
- main focus usually: availability
- Examples: Cassandra, Hbase

Graph Databases

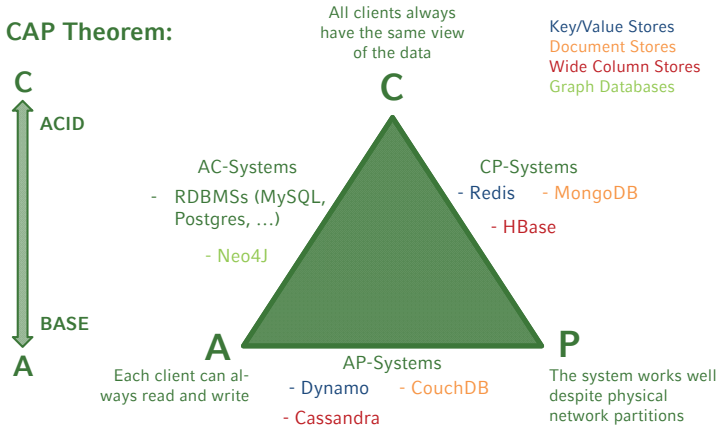
Graph Databases

- Use graphs to store and represent relationships between entities
- Composed of nodes and edges
- Each node and each edge can contain properties (Property-Graphs)



- Example: Neo4J

NoSQL DB — Big Picture, Finally



Agenda

1. What is Big Data?
2. Relational Database Management Systems
3. Managing Spatial (/Geo) Data
4. Managing Big Data
5. Distributed Data Processing

Motivation

- Drawbacks of RDBMS (in the context of Big Data)
  - Database system are difficult to scale
  - Database systems are difficult to configure and maintain
  - Diversification in available systems complicates its selection
  - Peak provisioning leads to unnecessary costs
- Advantages of NoSQL systems
  - Elastic scaling
  - Less administration
  - Better economics
  - Flexible data models

But ...

Motivation

- NoSQL drops a lot of functionality of RDBMS
  - No real data dictionaries, but semi-structured models for providing meta-data (usually still hard to access without explicit knowledge of the data model)
  - Transaction processing constrained to CAP-theorem
  - Often limited access control (no user groups, roles)
  - Limited indexing / efficiency is most replaced with scalability
- So what is left???
  - Storing massive amount of data in cluster environments (sharding and replication)
  - Eventual consistency (at some point after the change, every instance of the data is replication consistent)
  - Some database like APIs (e.g., CQL)

OK, and what exactly are NoSQL DBs so much different from a classical File-System?

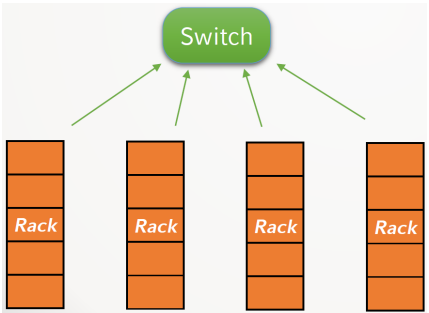
Distributed File Systems

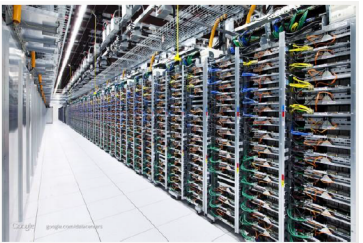
Well ... — not too much?

- Remember: there was something about a Data Science "Process" involving "preprocessing", and there was something about a "data pyramid" ...
  - Majority of analysis is still run on files
  - Machine learning, statistics and data mining methods usually access all available data
  - Most data mining and statistics methods require a well-defined input and not semi-structured objects (aha, this is one of the tasks to be done in preprocessing ...)
- Scalable Data Analytics often suffices with a distributed file system
- Analytics methods may be parallelized on top of the distributed file systems

Distributed File Systems

- Parallel computing ("cluster computing") architecture
  - computing nodes are stored on racks (8-64)
  - nodes on a single rack connected by a network (usually GB Ethernet)





Racks of servers (and switches at the top), at Google's Mayes County, Oklahoma data center

Picture from: extremetech.com

Large-Scale File-Systems

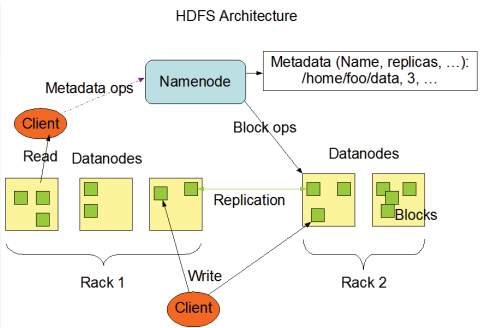
- Characteristics
  - Files are several TBs in size (Facebook's daily logs: 60TB; 1000 genomes project: 200TB; Google Web Index; 10+ PB)
  - Files are rarely updated
  - Reads and appends are common
- Examples
  - Google File System (GFS)
  - Hadoop Distributed File System (HDFS, by Apache)
  - CloudStore
  - HDF5
  - S3 (Amazon EC2)
  - ...

Large-Scale File-Systems

- Organization
  - Files are divided into chunks (typically 16-64MB in size)
  - Chunks are replicated n times (e.g. default in HDFS: $n = 3$) at n different nodes (best case scenario: replicas are located on different racks optimizing fault tolerance)
- How to find files?
  - There is a master node which holds a meta-file (directory) about location of all copies of a file
  - Thus, all participants using the DFS know where copies are located (through the master node)

Architecture

- HDFS: A distributed file system that provides high-throughput access to application data
- HDFS has a master/slave architecture, it looks like this (puhh ...):



Picture from http://hortonworks.com/hadoop/hdfs/#section_2

Map-Reduce: Differentiation to Other Systems

- Map-Reduce vs. RelDBMS:

	Map-Reduce	**RelDBMS**
Data size	Petabytes	Gigabytes
Access	Batch	Interactive and Batch
Updates	Write once, read many times	Read & Write many times
Structure	Dynamic schema	Static schema
Integrity	Low	High (normalized data)
Scaling	Linear	Non-linear

Map-Reduce: Differentiation to Other Systems

- Map-Reduce vs. High performance computing (HPC)
  - Accessing large data volumes becomes a problem in HPC, as the network bandwidth is the bottleneck
    Solved through Data Locality by Map-Reduce
  - Data flow must be handled explicitly (by programmers) in HPC
    Solved through higher level programming (data flow is implicit) by Map-Reduce
  - The handling of partial failures depends on the HPC architecture
    Map-Reduce is a shared-nothing-architecture (no dependence of tasks), so detection of failures and rescheduling of missing operations is “easy”

Map-Reduce: Differentiation to Other Systems

- In general, Map-Reduce can be used to manage large-scale computations in a way that is tolerant of hardware faults
- System itself manages automatic parallelization and distribution, I/O scheduling as well as coordination of tasks that are implemented and encapsulated in functions called map() and reduce()
- System is able to cope with unexpected system failures or stragglers automatically
- Several implementations of the basic model available: Google's internal implementation, open-source implementation Hadoop (using HDFS), ...

Apropos Apache Hadoop

- Tools within Apache Hadoop Package (list is not exhaustive):

HBase	Distributed, column-oriented database
Hive	Distributed data warehouse
Pig	Higher-level data flow language and parallel execution framework
ZooKeeper	Distributed coordination service
Sqoop	Tool for bulk data transfer between structured data stores and HDFS
Oozie	Complex job workflow service
Chukwa	System for collecting management data
Mahout	Machine learning and data mining library
BigTop	Packaging and testing
Avro	serialization system for cross-language RPC and persistent data storage

Apache Hadoop: Limitations of Map-Reduce

- Map-Reduce is a successful batch-oriented programming model
- However, there is an increasing demand for additional processing modes:
  - Graph Analysis
  - Stream data processing
  - Text Analysis
  - ...
- Demand is growing for real-time and ad-hoc analysis
- Analyses with specific queries including only subsets of data (with additional time constraints)
- Solution in the Hadoop / Map-Reduce-World: YARN
- And finally: not everything is parallelizable ...