MITS6005

Big Data

Session 6

NoSQL

Overview: last session

- Hadoop cluster
- Architecture of HDFS, MapReduce and YARN
- Data Lake vs Data Warehouse

Overview: this week

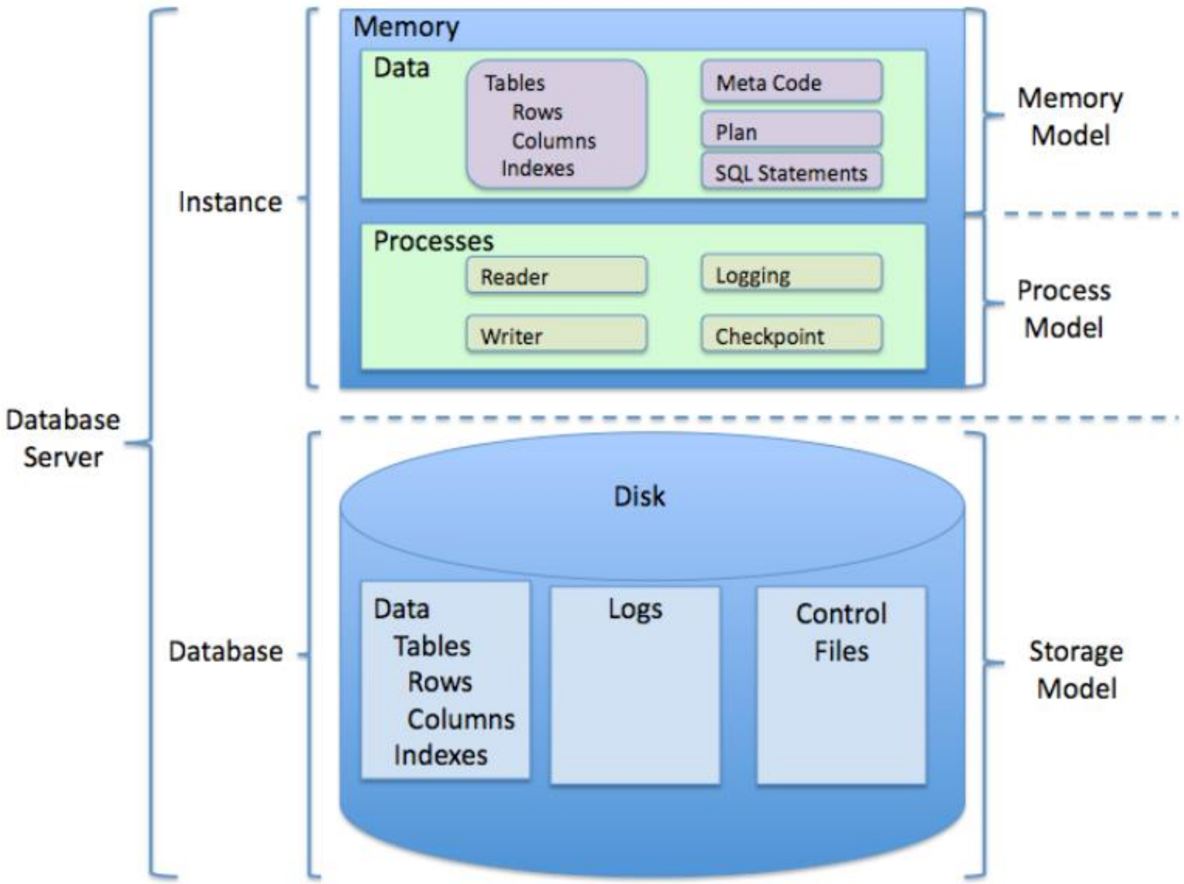
- Relational database
- What is NoSQL
- SQL vs NoSQL
- Motivation for NoSQL
- Advantages and disadvantages of NoSQL
- CAP theorem for RDBMS and Big Data
- NoSQL types/categories
- NoSQL in practice

Background - RDBMS

- A **relational database** is a digital database based on the relational model of data
- A software system to maintain relational databases is a relational database management system (RDBMS)
- Many relational database systems

  have an option of using the SQL

  for querying and maintaining the database.



Structure of a relational database

RDBMS – Advantages and Challenges

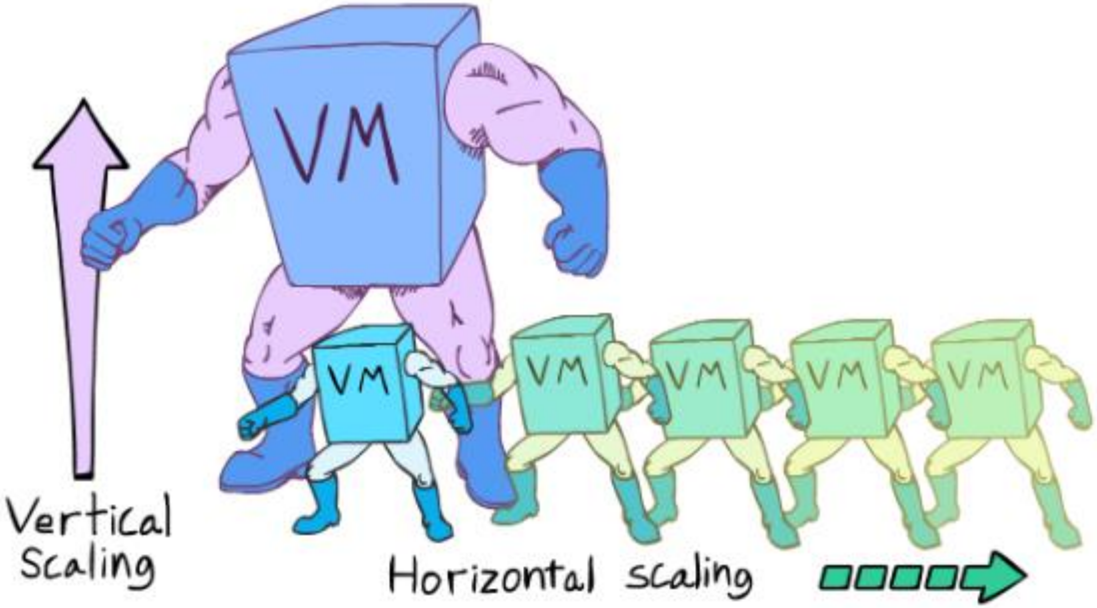
- Some advantages
  - ease of use;
  - low dependence between data distributed in different categories;
  - data management at the level of the internal language of the database is performed using simple and logically comprehensible commands based on the SQL language.

- Some Challenges:
  - the relational model has a low access speed and requires a lot of external memory;
  - as a result of the logical design, often a lot of tables makes it difficult to understand the data structure;
  - the subject area cannot always be represented as a set of tables;
  - Web-based applications caused spikes: e.g. social media sites such as Facebook and cloud-based solutions, such as Amazon S3, require large data
  - Hooking RDBMS to web-based application becomes trouble

Structure of a relational database

RDBMS - Issues with *scaling up*

- Best way to provide ACID and rich query model is to have the dataset on a single machine
- Limits to ***scaling up*** (or ***vertical scaling)*****:** increase the capacity of a single machine by adding more processing power and adding more storage and memory etc → dataset is just too big!
- ***Scaling out*** (or ***horizontal scaling)*****:** adding more machines or setting up a cluster or a distributed environment
- Approaches for horizontal scaling (multi-node database):
  - Master/Slave
  - Sharding (partitioning)

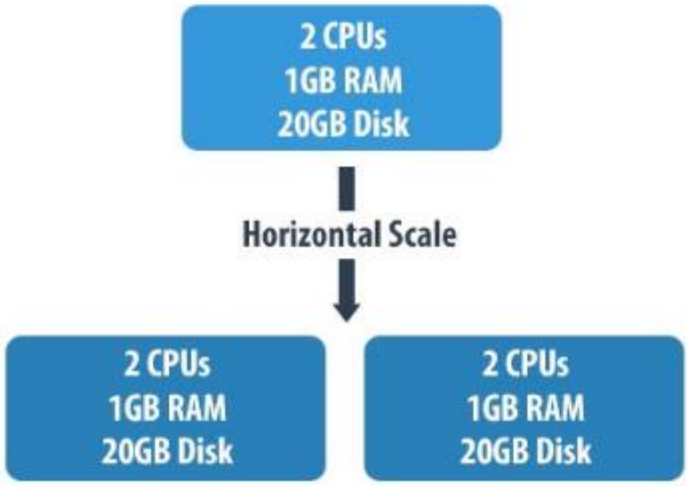


Scaling out RDBMS: Master/Slave

- Master/Slave
  - All writes are written to the master
  - All reads performed against the replicated slave databases
  - Critical reads may be incorrect as writes may not have been propagated down
  - Large datasets can pose problems as master needs to duplicate data to slaves

Scaling out RDBMS: Sharding

- Sharding (Horizontal Partitioning)
  - Scales well for both reads and writes
  - Not transparent, application needs to be partition-aware
  - Can no longer have relationships/joins across partitions
  - Loss of referential integrity across shards

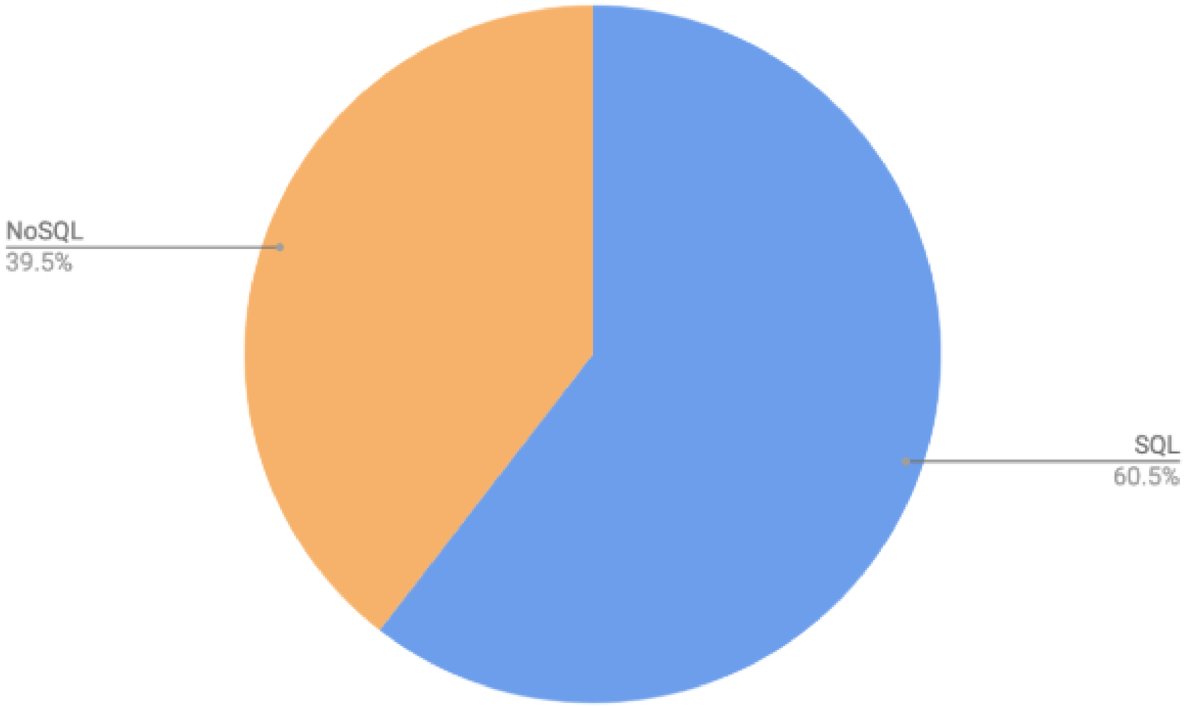


Other ways to scale out RDBMS

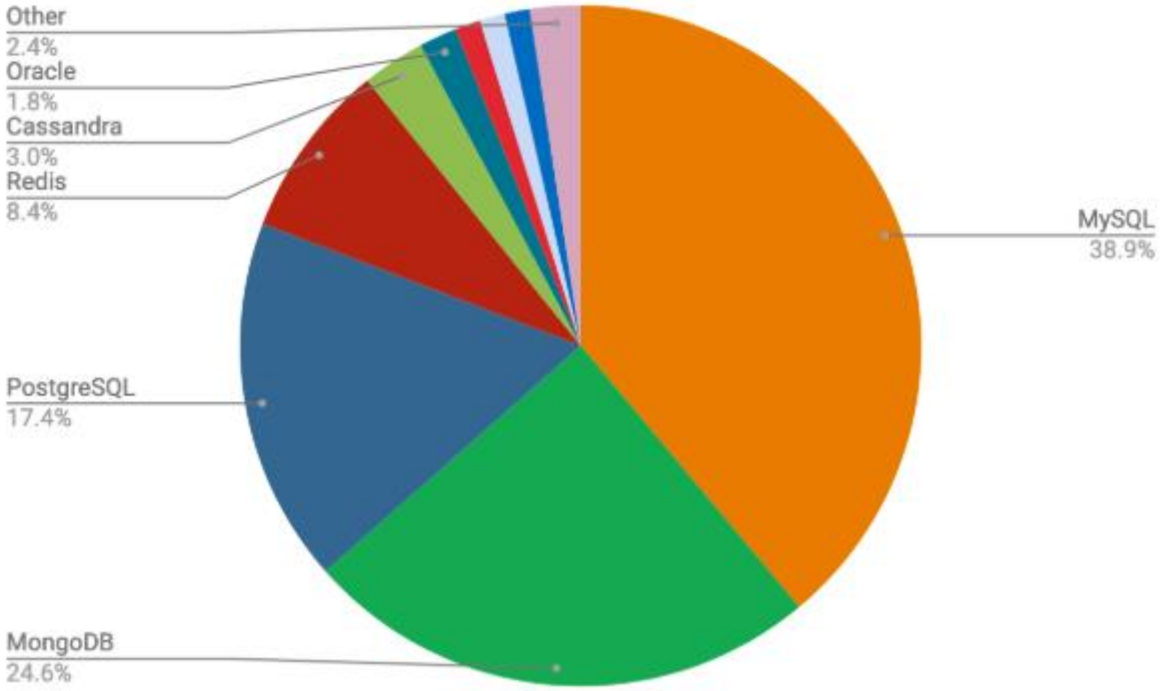
- Multi-Master replication
- INSERT only, not UPDATES/DELETES
- No JOINs, thereby reducing query time
  - This involves de-normalizing data
- In-memory databases

What is NoSQL?

- Stands for “**N**ot **O**nly **SQL**”



Uses of SQL and NoSQL in 2019



Most Popular Databases in 2019

NoSQL - Advantages

Key features (advantages):

- non-relational
- don’t require schema
- data are replicated to multiple nodes (so, identical & fault-tolerant) and can be partitioned
- Easy to scale: horizontal scalable
- cheap, easy to implement (open-source)
- massive write performance
- Ability to integrate with numerous third-party solutions: due to the lack of binding to specific data types, NoSQL APIs can be used to connect to a wide variety of web services and applications.
- High performance. NoSQL databases are optimized for working with big data. Thus, you do not have to sacrifice the speed of the application in order to scale it.

NOSQL - Disadvantages

Disadvantages/Cons:

- Don’t fully support relational features
  - no join, group by, order by operations (except within partitions)
  - no referential integrity constraints across partitions
- No declarative query language (e.g., SQL) $\rightarrow$ more programming
- Relaxed ACID (see CAP theorem) $\rightarrow$ fewer guarantees
- No easy integration with some applications that support SQL
- A number of difficulties with data sorting and other functions that require access to the elements by key

Who is using them?

SQL vs NoSQL

SQL Database	NoSQL database
Same type with fewer variations	Different types are available like document databases, Key-value stores, graph databases and wide –column stores.
They were developed in the 1970s to handle the data storage applications.	They were developed in the 21st century to overcome the limitations of SQL databases such as multi-structured data, agile development sprints, and scalability
Data is stored in tabular format.	Data storage varies with database type.
Data types and structure are fixed beforehand. The entire database needs to be altered to add a new data item.	Dynamic storage. Dissimilar data can be stored together which is not the case with SQL databases.
Vertical scalability.	Horizontal scalability.
Open technologies and closed source databases are used as a development model.	Open technologies are only used.
It supports multi-record ACID transactions.	Mostly does not support them.
Data manipulation is done using specific data manipulation language.	Data manipulation is done through object-oriented APIs
Strong consistency	Some products provide strong whereas others provide eventual consistency.
The velocity of data is moderate	The velocity of data is very high.
Suitable for structured data	Suitable for structured, semi-structured as well as unstructured data.
Examples are MySQL, Oracle Database, Postgres	Examples are MongoDB, HBase, Cassandra, Neo4j

3 major papers for NOSQL

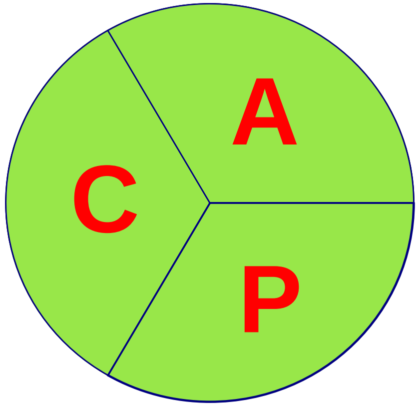
- Three major papers were the “seeds” of the NOSQL movement:
  - BigTable (Google)
  - DynamoDB (Amazon)
    - Ring partition and replication
    - Gossip protocol (discovery and error detection)
    - Distributed key-value data stores
    - Eventual consistency
  - CAP Theorem

The Perfect Storm

- Large datasets, acceptance of alternatives, and dynamically-typed data has come together in a “perfect storm”
- Not a backlash against RDBMS
- SQL is a rich query language that cannot be rivaled by the current list of NOSQL offerings

CAP Theorem

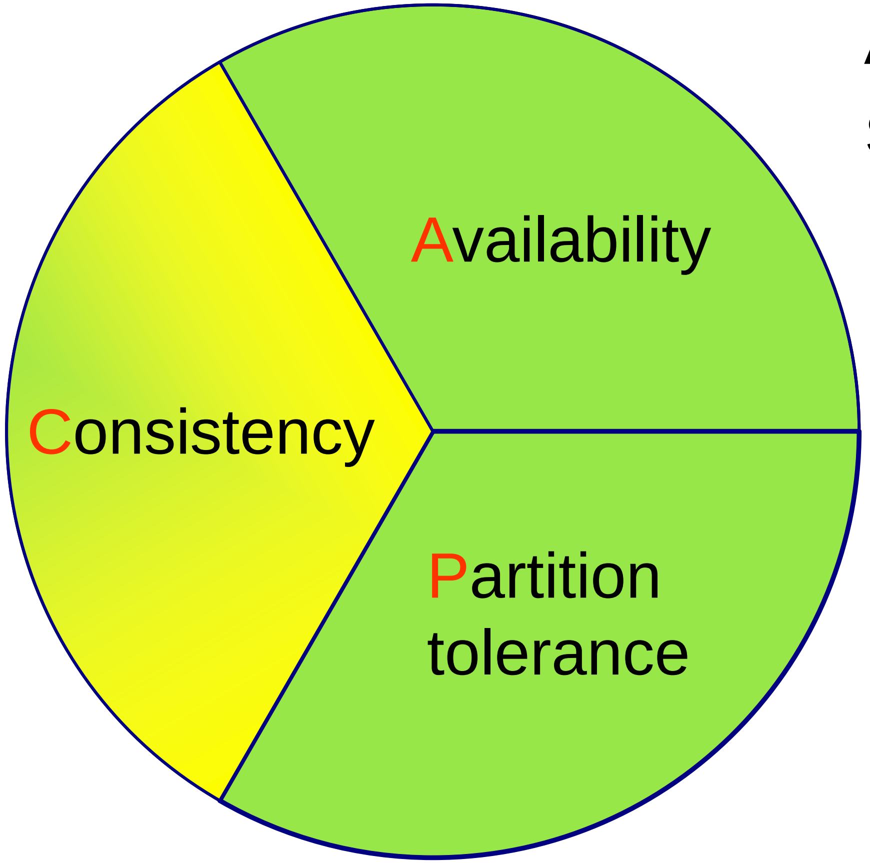
- Suppose three properties
  of a distributed system (sharing data)
  - **Consistency:**
    - all copies have same value
  - **Availability:**
    - reads and writes always succeed
  - **Partition-tolerance:**
    - system properties (consistency and/or availability) hold even when network failures prevent some machines from communicating with others



CAP Theorem

- **Brewer's CAP Theorem:**
  - *For any system sharing data, it is "impossible" to guarantee simultaneously all of these three properties*
  - You can have at most two of these three properties for any shared-data system
- Very large systems will "partition" at some point:
  - That leaves either **C** or **A** to choose from (traditional DBMS prefers **C** over **A** and **P**)
  - In almost all cases, you would choose **A** over **C** (except in specific applications such as order processing)

CAP Theorem



All client always have the same view of the data

CAP Theorem

- **Consistency**
  - 2 types of consistency:

1. Strong consistency – ACID (**A**tomicity, **C**onsistency, **I**solation, **D**urability)
2. Weak consistency – BASE (**B**asically **A**vailable **S**oft-state **E**ventual consistency)

CAP Theorem

- **ACID**
  - A DBMS is expected to support “ACID transactions,” processes that are:
  - **Atomicity:** either the whole process is done or none is
  - **Consistency:** only valid data are written
  - **Isolation:** one operation at a time
  - **Durability:** once committed, it stays that way

- **CAP**
  - **Consistency:** all data on cluster has the same copies
  - **Availability:** cluster always accepts reads and writes
  - **Partition tolerance:** guaranteed properties are maintained even when network failures prevent some machines from communicating with others

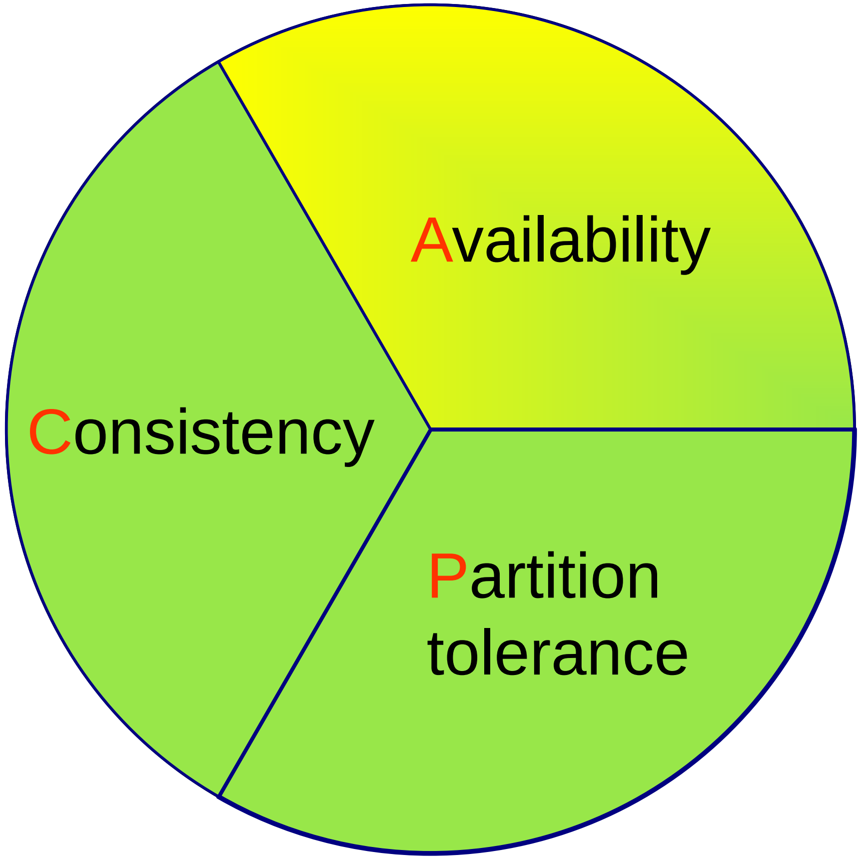
CAP Theorem

- A consistency model determines rules for visibility and apparent order of updates
- Example:
  - Row X is replicated on nodes M and N
  - Client A writes row X to node N
  - Some period of time t elapses
  - Client B reads row X from node M
  - **Does client B see the write from client A?**
  - Consistency is a continuum with tradeoffs
  - **For NOSQL, the answer would be: “maybe”**
  - CAP theorem states: *“strong consistency can't be achieved at the same time as availability and partition-tolerance”*

CAP Theorem

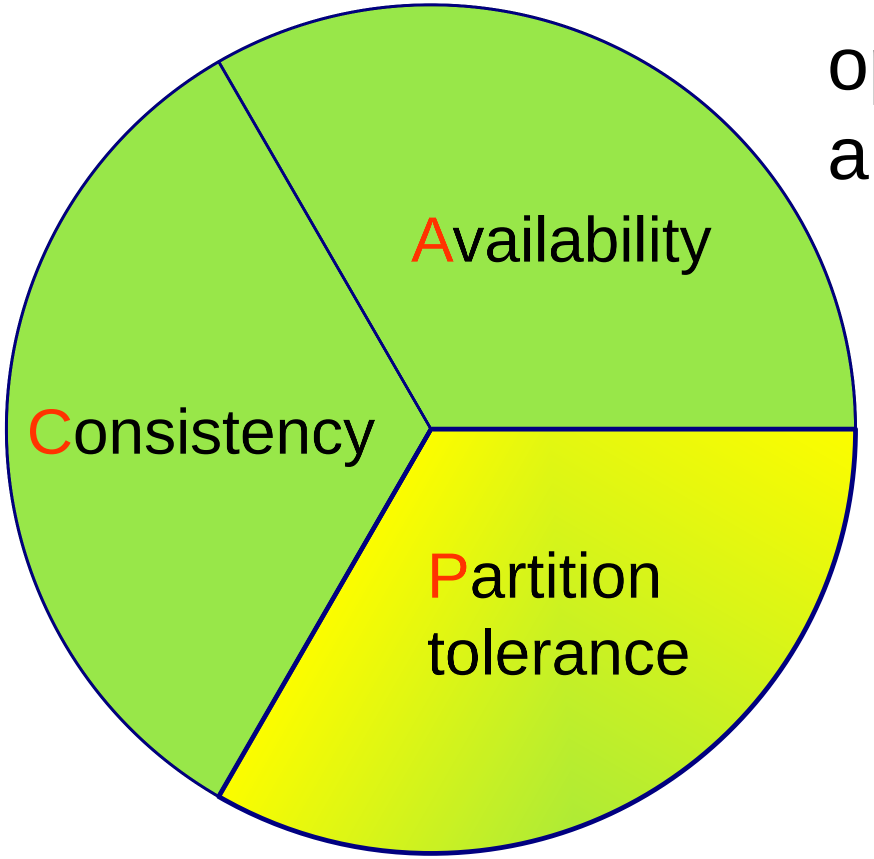
- Eventual consistency
  - When no updates occur for a long period of time, eventually all updates will propagate through the system and all the nodes will be consistent
- Cloud computing
  - ACID is hard to achieve, moreover, it is not always required, e.g. for blogs, status updates, product listings, etc.

CAP Theorem



Each client always can read and write.

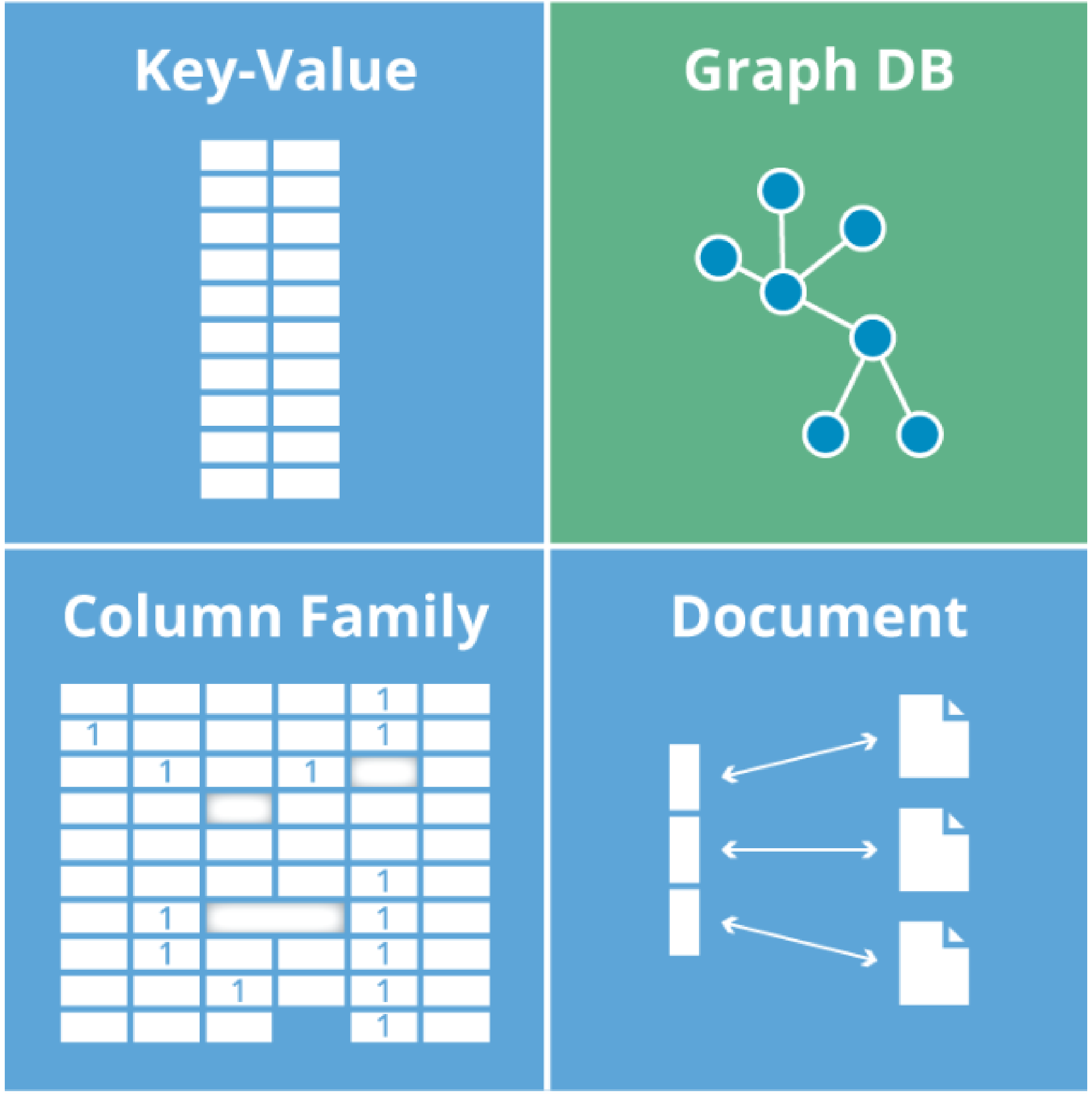
CAP Theorem



A system can continue to operate in the presence of a network partitions

NOSQL categories

1. Key-value
   - Example: DynamoDB, Voldermort, Scalaris
   - Redis, Memcached, Apache Ignite, Riak
2. Document-based
   - Example: MongoDB, Apache CouchDB, ArangoDB, Couchbase, Cosmos DB, IBM Domino, MarkLogic, OrientDB.
3. Column-based
   - Example: BigTable, Cassandra, Hbased
   - Cassandra, Hbase, Scylla
4. Graph-based
   - Example: Neo4J, InfoGrid
   - Neo4j, AllegroGraph



Key-value

- Focus on scaling to huge amounts of data
- Designed to handle massive load
- Based on Amazon's dynamo paper
- Data model: (global) collection of Key-value pairs
- *Dynamo ring partitioning* and *replication*
- Example: (DynamoDB)
  - *items* having one or more attributes (name, value)
  - An *attribute* can be single-valued or multi-valued like set.
  - items are combined into a *table*

Key-value

- Basic API access:
  - get(key): extract the value given a key
  - put(key, value): create or update the value given its key
  - delete(key): remove the key and its associated value
  - execute(key, operation, parameters): invoke an operation to the value (given its key) which is a special data structure (e.g. List, Set, Map .... etc)

Key-value

Pros:

- very fast
- very scalable (horizontally distributed to nodes based on key)
- simple data model
- eventual consistency
- fault-tolerance

Cons:

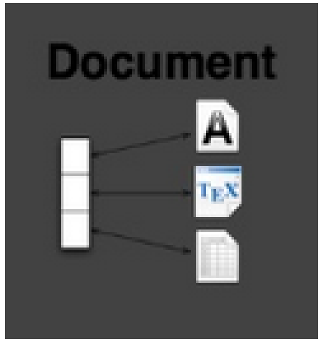
- Can’t model more complex data structure such as objects

Key-value

Name	Producer	Data model	Querying
SimpleDB	Amazon	set of couples (key, {attribute}), where attribute is a couple (name, value)	restricted SQL; select, delete, GetAttributes, and PutAttributes operations
Redis	Salvatore Sanfilippo	set of couples (key, value), where value is simple typed value, list, ordered (according to ranking) or unordered set, hash value	primitive operations for each value type
Dynamo	Amazon	like SimpleDB	simple get operation and put in a context
Voldemort	LinkeId	like SimpleDB	similar to Dynamo

Document-based

- Can model more complex objects
- Inspired by Lotus Notes
- Data model: collection of documents
- Document: JSON (**J**ava**S**cript **O**bject **N**otation is a data model, key-value pairs, which supports objects, records, structs, lists, array, maps, dates, Boolean with **nesting**), XML, other semi-structured formats.



Document-based

- Example: (MongoDB) document
  - {Name:"Jaroslav",

    Address:"Malostranske nám. 25, 118 00 Praha 1”,

    Grandchildren: {Claire: "7", Barbara: "6", "Magda: "3", "Kirsten: "1", "Otis: "3", Richard: "1“}

    Phones: [“123-456-7890”, “234-567-8963”]

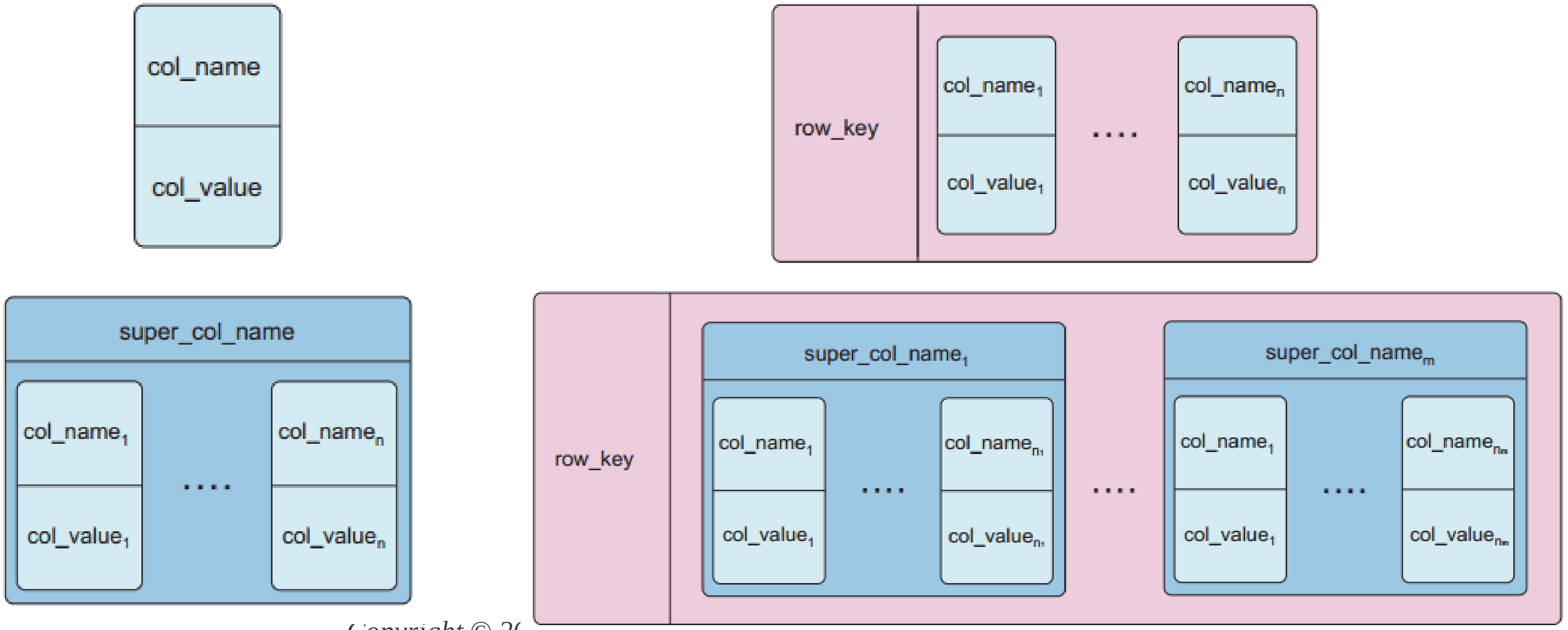
    }

Document-based

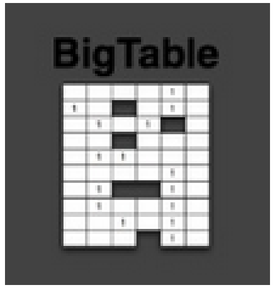
Name	Producer	Data model	Querying
MongoDB	10gen	object-structured documents stored in collections; each object has a primary key called ObjectId	manipulations with objects in collections (find object or objects via simple selections and logical expressions, delete, update,)
Couchbase	Couchbase[1]	document as a list of named (structured) items (JSON document)	by key and key range, views via Javascript and MapReduce

Column-based

- Based on Google’s BigTable paper
- Like column oriented relational databases (store data in column order) but with a twist
- Tables similarly to RDBMS, but handle semi-structured
- Data model:
  - Collection of Column Families
  - Column family = (key, value) where value = set of **related** columns (standard, super)
  - indexed by *row key*, *column key* and *timestamp*



Column-based



- One column family can have variable numbers of columns
- Cells within a column family are sorted “physically”
- Very sparse, most cells have null values
- **Comparison:** RDBMS vs column-based NOSQL
  - Query on multiple tables
    - **RDBMS:** must fetch data from several places on disk and glue together
    - **Column-based NOSQL:** only fetch column families of those columns that are required by a query (all columns in a column family are stored together on the disk, so multiple rows can be retrieved in one read operation → data locality)

Column-based

- Example: (Cassandra column family--timestamps removed for simplicity)

UserProfile = {

Cassandra = { emailAddress:”casandra@apache.org” , age:”20”}

TerryCho = { emailAddress:”terry.cho@apache.org” , gender:”male”}

Cath = { emailAddress:”cath@apache.org” , age:”20”,gender:”female”,address:”Seoul”}

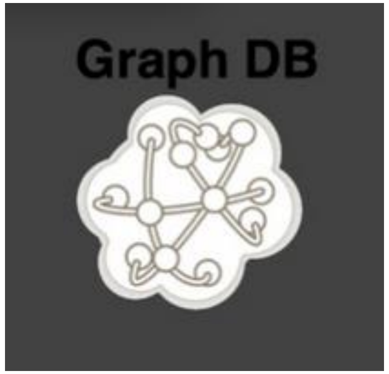
}

Column-based

Name	Producer	Data model	Querying
BigTable	Google	set of couples (key, {value})	selection (by combination of row, column, and time stamp ranges)
HBase	Apache	groups of columns (a BigTable clone)	JRUBY IRB-based shell (similar to SQL)
Hypertable	Hypertable	like BigTable	HQL (Hypertext Query Language)
CASSANDRA	Apache (originally Facebook)	columns, groups of columns corresponding to a key (supercolumns)	simple selections on key, range queries, column or columns ranges
PNUTS	Yahoo	(hashed or ordered) tables, typed arrays, flexible schema	selection and projection from a single table (retrieve an arbitrary single record by primary key, range queries, complex predicates, ordering, top-k)

Graph-based

- Focus on modeling the structure of data (*interconnectivity*)
- Scales to the complexity of data
- Inspired by mathematical Graph Theory (G=(E,V))
- Data model:
  - (Property Graph) nodes and edges
    - Nodes may have properties (including ID)
    - Edges may have labels or roles
  - Key-value pairs on both
- Interfaces and query languages vary
- *Single-step* vs *path expressions* vs *full recursion*
- Example:
  - Neo4j, FlockDB, Pregel, InfoGrid …



Conclusion

- NOSQL database cover only a part of data-intensive cloud applications (mainly Web applications)
- Problems with cloud computing:
  - SaaS (**S**oftware **a**s **a** **S**ervice or on-demand software) applications require enterprise-level functionality, including ACID transactions, security, and other features associated with commercial RDBMS technology, i.e. NOSQL should not be the only option in the cloud
  - Hybrid solutions:
    - Voldemort with MySQL as one of storage backend
    - deal with NOSQL data as semi-structured data

      $\rightarrow$ integrating RDBMS and NOSQL via SQL/XML

Conclusion

- next generation of highly scalable and elastic RDBMS: *NewSQL databases* (from April 2011)
  - they are designed to scale out horizontally on shared nothing machines,
  - still provide ACID guarantees,
  - applications interact with the database primarily using SQL,
  - the system employs a lock-free concurrency control scheme to avoid user shut down,
  - the system provides higher performance than available from the traditional systems.
- Examples: MySQL Cluster (most mature solution), VoltDB, Clustrix, ScalArc, etc.

References

- Rajshekhar Sunderraman
  - http://tinman.cs.gsu.edu/~raj/8711/sp13/berkeleydb/finalpres.ppt
- Tobias Ivarsson
  - http://www.slideshare.net/thobe/nosql-for-dummies
- Jennifer Widom
  - http://www.stanford.edu/class/cs145/ppt/cs145nosql.pptx
- Ruoming Jin
  - http://www.cs.kent.edu/~jin/Cloud12Spring/HbaseHivePig.pptx
- Seth Gilbert
  - http://lpd.epfl.ch/sgilbert/pubs/BrewersConjecture-SigAct.pdf
- Patrick McFadin
  - http://www.slideshare.net/patrickmcfadin/the-data-model-is-dead-long-live-the-data-model
- Chaker Nakhli
  - http://www.javageneration.com/wp-content/uploads/2010/05/Cassandra_DataModel_CheatSheet.pdf
- Ricky Ho
  - http://horicky.blogspot.com/2010/10/bigtable-model-with-cassandra-and-hbase.html


The End