Introduction to Databases

Gianluca Quercini

Laboratoire de Recherche en Informatique
CentraleSupélec

2019 – 2020

CentraleSupélec

Distributed Databases

Distributed Database

- **Single-server database**:
    - Database on only one machine.
    - All data under the control of one DBMS. ☺
    - Performances of DBMS decrease as data volume increases. ☹
    - Best solution if the scale of data allows it.
- **Distributed database.**
    - Data reside on multiple machines (a.k.a., *nodes*).
    - Each machine is independent of the others (**shared-nothing architecture**).
    - Allows storage and management of large volumes of data. ☺
    - Far more complex than a single-server database. ☹
    - Scale out only if a single-server database is not viable.

Characteristics of Distributed Databases

- **Data distribution** options: **replication** and **sharding**.
- **Location transparency**: the user queries the data without even knowing that they are distributed.
- **Replication transparency**: consistency of the data that are replicated.
- **Data management functions**: security and concurrent access control.

Replication

Department		
codeD	**nameD**	**budget**
14	Administration	300,000
25	Education	150,000
62	Finance	600,000
45	Human Resources	150,000

Department		
codeD	**nameD**	**budget**
14	Administration	300,000
25	Education	150,000
62	Finance	600,000
45	Human Resources	150,000

Department		
codeD	**nameD**	**budget**
14	Administration	300,000
25	Education	150,000
62	Finance	600,000
45	Human Resources	150,000

+ **Scalability.** Multiple nodes receive queries on the same tuple.
+ **Latency.** Worldwide database, replica close to the user.
+ **Fault tolerance.** If a node fails, the others can still answer queries.
− **Consistency.** Keep all replicas up-to-date.
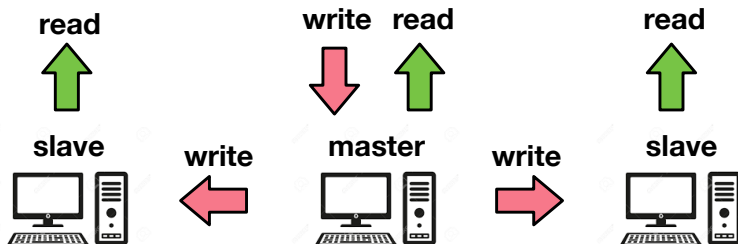
Replication and Consistency

Department		
codeD	**nameD**	**budget**
14	Administration	**500,000**
25	Education	150,000
62	Finance	600,000
45	Human Resources	150,000

Department		
codeD	**nameD**	**budget**
14	Administration	300,000
25	Education	150,000
62	Finance	600,000
45	Human Resources	150,000

Department		
codeD	**nameD**	**budget**
14	Administration	300,000
25	Education	150,000
62	Finance	600,000
45	Human Resources	150,000

- **Synchronous update**. The update is propagated immediately.
    - $+$ Short **inconsistency window**.
    - $-$ Not viable if updates are frequent.
- **Asynchronous update**. The update is propagated at regular intervals.
    - $+$ Best option when updates are frequent.
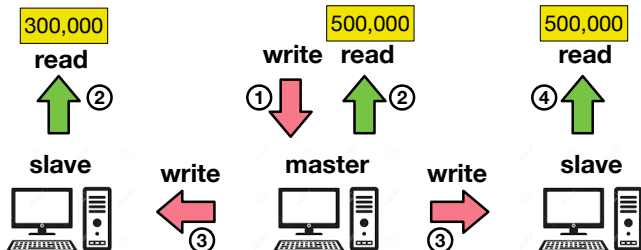   - $-$ Large inconsistency window.

Master-Slave Replication

- Write operations: on the master. Writes are propagated to the slaves.
- Read operations: from the master and from the slaves.
+ No write conflicts.
− Single point of failure.
  - If the master is down, write operations are not allowed.
  - Algorithms exist to **elect** a new master.

Master-Slave Replication – Read conflicts

**Write**: update (Department, budget=500,000)   **Read**: select (Department, budget)

Department		
codeD	**nameD**	**budget**
14	Administration	**300,000**
25	Education	150,000
62	Finance	600,000
45	Human Resources	150,000

Department		
codeD	**nameD**	**budget**
14	Administration	**500,000**
25	Education	150,000
62	Finance	600,000
45	Human Resources	150,000

| Department		
codeD	**nameD**	**budget**
14	Administration	**500,000**
25	Education	150,000
62	Finance	600,000
45	Human Resources	150,000

Peer-to-peer replication
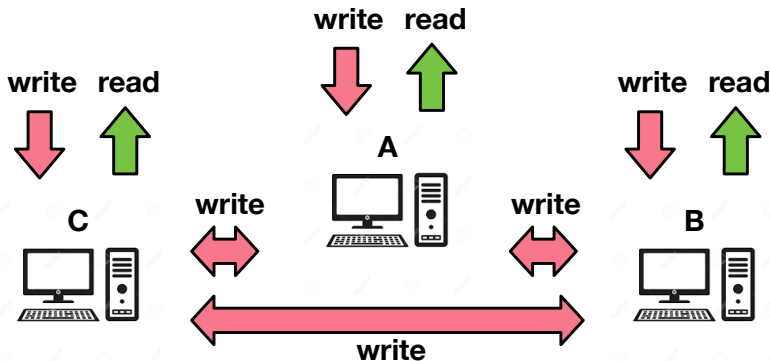
- Write and read operations on any node.
- + No single point of failure (very high availability).
- − Write and read conflicts (low consistency).

Sharding
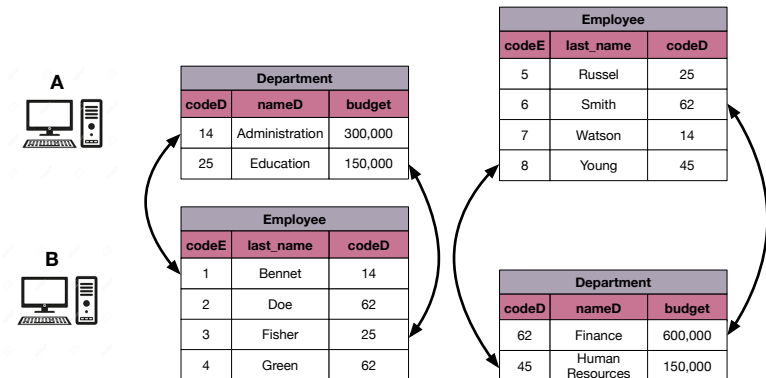
- Tuples partitioned into balanced **shards**. Shards distributed across the nodes.

+ Load balance.

+ No consistency problems.

    – Loss of data when node fail.

    – Join across different nodes.

    – Updates entail changes in shards.

Combining Replication and Sharding
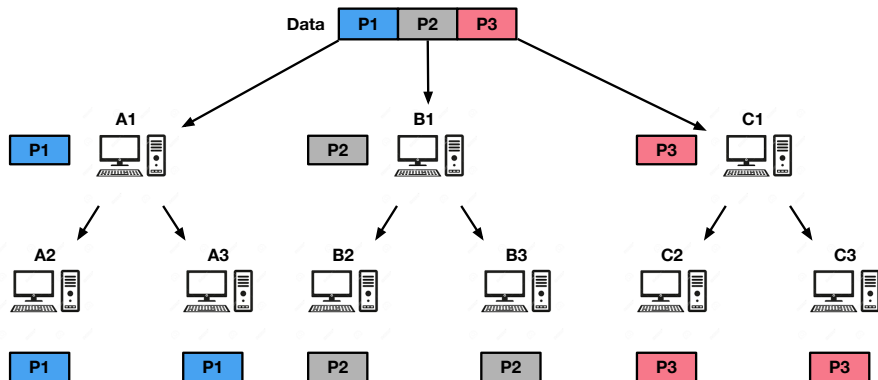
The CAP Theorem

- **Consistency.** "Equivalent to having a single up-to-date copy of the data" (Brewer).
- **Availability.** A database can perform read/write operations even when some nodes fail.
- **Partition tolerance.** The database can operate when a network partition occurs as if the partition did not happen.

Theorem (CAP, Brewer 1999)

Given the three properties of consistency, availability and partition tolerance, a networked shared-data system can have at most two of these properties.

The CAP Theorem

Sketch of the proof (Gilbert & Lynch, 2002).

- Suppose that a database is partition tolerant. Two cases when a network partition occurs.
    - Write operations are allowed.
        - Consistency cannot be guaranteed.
        - AP database.
    - Write operations are not allowed.
        - Database unavailable (write operations forbidden on reachable nodes).
        - CP database.
- Consistency and availability only when no network partition occurs.
    - Assuming absence of network partitions means that a database is not partition tolerant.

The CAP Theorem

- The theorem has been largely misunderstood for years.
- Common interpretation:
    - Network partitions occur.
    - Therefore, the CAP theorem reduces to the choice of consistency over availability.
- However, network partitions are not that frequent.
    - Makes no sense to give up either consistency or availability.
- A distributed database should detect network partitions and operate in a *partition mode* with:
    - Reduced availability (some operations forbidden), or
    - Reduced consistency.
- Partition recovery to resolve the inconsistencies.

BASE Consistency Model

- ACID transactions are a pessimistic consistency model.
- An optimistic model to consistency, used in distributed databases, is BASE.
- **Basic Availability** (BA). The database appears to work most of the time.
- **Soft state** (S). Write and read inconsistencies can occur.
- **Eventually consistent** (E). The database is consistent at some time.
  - Update propagation.

NoSQL Databases

Limitations of the Relational Model

Relational DBMSs have two major limitations:

- **Impedance mismatch**.
  - The DBMS always models data as tables.
  - An application models data in different ways, depending on their nature.

Example

- The DBMS models a graph as a collection of tables.
- An application models a graph as an adjacency list.

- Conceived when data distribution was not a concern.
  - Consistency at any cost: not always a good option.
  - Problems when sharding: joins across nodes.
- NoSQL databases conceived to address these concerns.
  - First NoSQL databases: Amazon Dynamo (2007), Google BigTable (2008).

Characteristics of NoSQL databases

- **NoSQL** means *Not Only SQL* (term coined in 2011).
- NoSQL databases are not based on the relational model.
- NoSQL databases are generally **open-source**.
- NoSQL databases are **cluster-oriented**.
- NoSQL databases tend to privilege **availability** over consistency.
- NoSQL databases are **schemaless**.
- NoSQL databases are classified into four families:
    - **Key-value** databases.
    - **Document** databases.
    - **Column-family** databases.
    - **Graph** databases.
- The first three databases are based on the notion of **aggregate**.

Aggregate

- **Aggregate**: data structure containing the description of an entity.
- All data in the same aggregate must stay in the same shard.
- Operations within the same aggregate are atomic.
- Schema is flexible and non-normalized.

Aggregate

```
{                                    {
  "codeE": 1,                          "codeE": 2,
  "first": "Joseph",                   "first": "John",
  "last": "Bennett",                   "last": "Doe",
  "position": "Office assistant",
  "salary": 55,000,                    "salary": 45,000
  "department": [                      "department": [
    {                                    {
      "codeD": 14,                         "codeD": 14,
      "nameD": "Administration",           "nameD": "Administration",
      "budget": 30000                      "budget": 30000
    }                                    }
  ]                                    ]
}
```
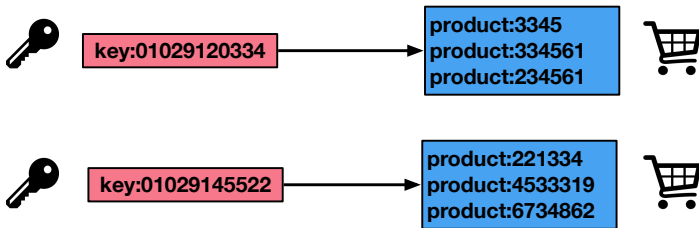
Aggregate

- Different ways to model the data.

Aggregate

```
{
  "codeD": 14,
  "name": "Administration",
  "budget": 30000,
  "employees": [
    {
      "codeE": 1,
      "first": "Joseph",
      ....
    },
    {
      "codeE": 7,
      "first": "Michael",
      ....
    }
  ]
}
```

Key-value Data Model

- Data are modeled as **key-value pairs**.
    - **Key**: alphanumeric string auto-generated by the database.
    - **Value**: an aggregate.
    - **Query**: get a value given its key.
- Fast read/write operations.
- Little to no checks on integrity constraints.
- Example: shopping cart.
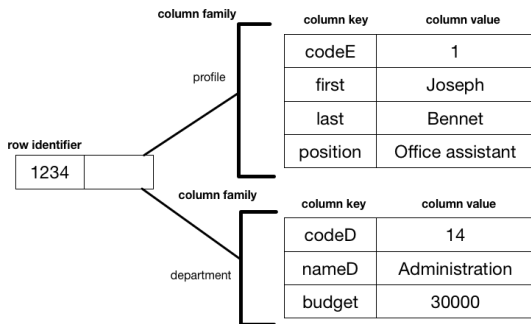- Key-value databases: Amazon Dynamo, Voldermort, Riak, Memcached DB.

Document Data Model

- Data are modeled as **key-value pairs**.
    - **Key**: alphanumeric string auto-generated by the database.
    - **Value**: an aggregate (called **document**).
    - **Query**: get documents by key and by the values of their properties.
- Examples: MongoDB, CouchDB, DocumentDB, OrientDB

Column-family Data Model

- Data are modeled as **key-value pairs**.
  - **Key**: row identifier.
  - **Value**: an aggregate, composed of one or more **column families**.
  - **Query**: get a row given its key and the values of its columns.
- **Sharding unit**: a row.
- **Storage unit**: a column family.
- Column-family databases: BigTable, HBase, Cassandra.

column key	column value
codeE	1
first	Joseph
last	Bennet
position	Office assistant

column family — profile

row identifier

| 1234 |

column key	column value
codeD	14
nameD	Administration
budget	30000

column family — department

Column-family Data Model

- Definition of a small number of column families.
- As many columns as we need.
- The value of a column can be an aggregate.
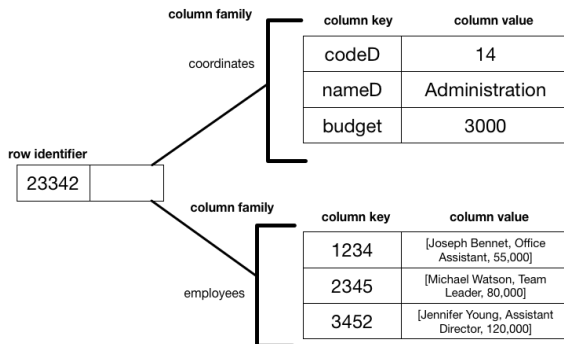
Graph Databases

- DBMS specifically thought to manage and process graphs.
- Two components:
    - **Storage engine**: dictates how the graph is stored.
    - **Processing engine**: dictates how the graph is processed.
- Native storage engine. Storage is tailored to graphs.
- Native processing engine. Operations optimized for graphs.
- Example in details: Neo4j.