Lecture Notes to
Big Data Management and Analytics
Winter Term 2018/2019
NoSQL Databases

© Matthias Schubert, Matthias Renz, Felix Borutta, Evgeniy Faerman, Christian Frey, Klaus Arthur Schmid, Daniyal Kazempour, Julian Busch
© 2016-2018
NoSQL Database Systems

Outline

• History

• Concepts
  • ACID
  • BASE
  • CAP

• Data Models
  • Key-Value Stores
  • Document Databases
  • Wide Column Stores
  • Graph Databases
History

60s: IBM developed the Hierarchical Database Model

- Tree-like structure
- Data stored as records connected by links
- Support only one-to-one and one-to-many relationships

Mid 80’s: Rise of Relational Database Model

- Data stored in a collection of tables (rows and columns)
  → Strict relational scheme
- SQL became standard language (based on relational algebra)
  → Impedance Mismatch!
Given the following database scheme and an object of type Supply:

How to incorporate the data bundled in the object Supply into the relational DB?
History – Impedance Mismatch

Supply:
  Supplier:
    SNR: L1
    Sname: Meier
    status: 20
    location: Wetter
  Project:
    PNR: P2
    Pname: Pleite
    location: Bonn
Parts:
  ONR: T6
  Oname: screw
  color: rot
  weight: 03
  amount: 700

<table>
<thead>
<tr>
<th>SNR</th>
<th>Sname</th>
<th>Status</th>
<th>location</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PNR</th>
<th>Pname</th>
<th>location</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ONR</th>
<th>Oname</th>
<th>color</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SNR</th>
<th>PNR</th>
<th>ONR</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

INSERT INTO L VALUES (Supply.getSupplier().getLNR(), ...);

INSERT INTO P VALUES (Supply.getProject().getPNR(), ...);

...
History – Impedance Mismatch

- Object-oriented encapsulation vs. storing data distributed among several tables
  → Lots of data type maintenance by the programmer

```
INSERT INTO LTP VALUES (...);
```
History

Mid 90’s: Trend of the Object-Relational Database Model

• data stored as objects (including data and methods)
• avoids of object-relational mapping
  → Programmer-friendly
• but still Relational Databases prevailed in the 90’s

Mid 2000’s: Rise of Web 2.0

• Lots of user generated data through web applications
  → storage systems had to become scaled up
History

Approaches to scale up storage systems

• Two opportunities to solve the rising storage system:
  • Vertical scaling
    Enlarge a single machine
    – Limited in space
    – Expensive
  • Horizontal scaling
    Use many commodity machines and form *computer clusters or grids*
    – Cluster maintenance
Approaches to scale up storage systems

• Two opportunities to solve the rising storage system:
  • Vertical scaling
    Enlarge a single machine
    – Limited in space
    – Expensive
  • Horizontal scaling
    Use many commodity machines and form *computer clusters or grids*
    – Cluster maintenance
History

Mid 2000’s: Birth of the NoSQL Movement

• Problem of computer clusters:
  Relational databases do not scale well horizontally

→ 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 (>225)
Characteristics of NoSQL Databases

There is no unique definition but some characteristics for NoSQL Databases:

- Horizontal scalability (cluster-friendliness)
- Non-relational
- Distributed
- Schema-less
- Open-source (at least most of the systems)
About the concepts behind NoSQL Databases

ACID – The holy grail of RDBMSs:

• **Atomicity**: Transactions happen entirely or not at all. If a transaction fails (partly), the state of the database is unchanged.

• **Consistency**: Any transaction brings the database from one valid state to another and does not break one of the pre-defined rules (like constraints).

• **Isolation**: Concurrent execution of transactions results in a system state that would be obtained if transactions were executed serially.

• **Durability**: Once a transaction has been committed, it will remain so.
About the concepts behind NoSQL Databases

BASE – An artificial concept for NoSQL databases:

• **Basically Available:** The system is generally available, but some data might not be available 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 settled on the opposite site to ACID when considering a „consistency-availability spectrum“
About the concepts behind NoSQL Databases

The two types of consistency:

- **Logical consistency:** Data is consistent within itself (Data Integrity)

- **Replication consistency:** Data is consistent across multiple replicas (on multiple machines)
About the concepts behind NoSQL Databases

Levels of Consistency:

- Eventual Consistency
  - Monotonic Read Consistency
    - M.R.C. + R.Y.O.W.
  - Immediate Consistency
    - Strong Consistency
      - Transactions
    - Read-Your-Own-Writes
About the concepts behind NoSQL Databases

Levels of Consistency:

• Eventual Consistency: Write operations are not spread across all servers/partitions immediately

• Monotononinc Read Consistency: A client who read an object once will never read an older version of this object

• Read Your Own Writes: A client who wrote an object will never read an older version of this object

• Immediate Consistency: Updates are propagated immediately, but not atomic
About the concepts behind NoSQL Databases

Levels of Consistency:

- **Strong consistency:** Updates are propagated immediately + support of atomic operations on single data entities (usually on master nodes)

- **Transactions:** Full support of ACID transaction model
About the concepts behind NoSQL Databases

Brewer’s CAP Theorem:

Any networked shared-data system can have at most two of the three desired properties!
About the concepts behind NoSQL Databases

Possible DB-Systems by CAP Theorem:

- **CP-Systems**: Fully consistent and partitioned systems renounce availability. Only consistent nodes are available.

- **AP-Systems**: Fully available and partitioned systems renounce consistency. All nodes answer to queries all the time, even if answers are inconsistent.

- **AC-Systems**: Fully available and consistent systems renounce partitioning. Only possible if the system is not distributed.
Big Picture

CAP Theorem:

- **C**: All clients always have the same view of the data
- **A**: Each client can always read and write
- **P**: The system works well despite physical network partitions
Big Picture

CAP Theorem:

- **C**: All clients always have the same view of the data
- **A**: Each client can always read and write
- **P**: The system works well despite physical network partitions

**ACID**
- AC-Systems
  - RDBMSs (MySQL, Postgres, …)

**BASE**
- AP-Systems
- CP-Systems
NoSQL Data Models

The 4 Main NoSQL Data Models:

• Key/Value Stores
• Document Stores
• Wide Column Stores
• Graph Databases
NoSQL Data Models

Key/Value Stores:

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

Key/Value Stores:

• consistency models range from eventual consistent to serializable

• some systems support ordering keys which enables efficient query processing, e.g., for range queries

• some systems support in-memory data maintenance and others manage data based on hard drives and SSDs

→ Available Systems are very heterogeneous
NoSQL Data Models

Key/Value Stores - Redis:

- in-memory data structure store with built-in replication, transactions and different levels of on-disk persistence

- supports complex types like lists, sets, hashes, ...

- supports various *atomic* operations

```bash
>> SET val 1
>> GET val => 1
>> INCR val => 2
>> LPUSH my_list a (=> 'a')
>> LPUSH my_list b (=> 'b','a')
>> RPUSH my_list c (=> 'b','a','c')
>> LRANGE my_list 0 1 => b,a
```
NoSQL Data Models

Key/Value Stores – The Redis cluster model:

• data is automatically sharded across nodes

• some degree of availability, achieved by master-slave architecture (but cluster stops in the event of larger failures)

• easily extendable
NoSQL Data Models

Key/Value Stores – the redis cluster model:

- **Nodes**: A, B, C, D
- **Hash slots**:
  - A: 0-4000
  - B: 4001-8000
  - C: 8001-12000
  - D: 12001-14522

**Add node**:
- New node: D
- Hash slots: 14522

**Remove node**:
- Removed node: C
- Hash slots: 14522

---

Big Data Management and Analytics
NoSQL Data Models

Key/Value Stores – The Redis cluster model:

Master Nodes | Hash slots
---|---
A | 0
| 5000
B | 5001
| 10000
C | 10001
| 14522

slave node B’ is promoted as the new master and hash slots 5001 – 10000 are still available

Hash slots 5001 – 10000 cannot be used anymore
Big Picture

CAP Theorem:

- **C** (Consistency): all clients always have the same view of the data.
- **A** (Availability): each client can always read and write.
- **P** (Partition Tolerance): the system works well despite physical network partitions.

ACID Systems
- AC-Systems: RDBMSs (MySQL, Postgres, ...)

Key/Value Stores
- CP-Systems: Redis
- AP-Systems: Dynamo

Big Data Management and Analytics
NoSQL Data Models

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
NoSQL Data Models

Document Stores:

given following text: Max Mustermann
Musterstraße 12
D-12345 Musterstadt

<contact>
  <first_name>Max</first_name>
  <last_name>Mustermann</last_name>
  <street>Musterstraße 12</street>
  <city>Musterstadt</city>
  <zip>12345</zip>
  <country>D</country>
</contact>

→ find all <contact>s where <zip> is “12345”
NoSQL Data Models

Document Stores:  

- data stored as documents in binary representation (BSON)
- similarly structured documents are bundled in collections
- provides own ad-hoc query language
- supports ACID transactions on document level
NoSQL Data Models

Document Stores:

MongoDB Data Management:
- automatic data sharding
- automatic re-balancing

- multiple sharding policies:
  - **Hash-based sharding**: Documents are distributed according to an MD5 hash → uniform distribution
  - **Range-based sharding**: Documents with shard key values close to one another are likely to be co-located on the same shard → works well for range queries
  - **Location-based sharding**: Documents are partitioned w.r.t. a user-specified configuration that associates shard key ranges with specific shards and hardware
NoSQL Data Models

Document Stores:  

MongoDB Consistency & Availability:

- default: strong consistency (but configurable)

- increased availability through replication
  
  - replica sets consist of one primary and multiple secondary members
  
  - MongoDB applies writes on the primary and then records the operations on the primary’s oplog
Big Picture

CAP Theorem:

- **C** (Consistency): all clients always have the same view of the data.
- **A** (Availability): each client can always read and write.
- **P** (Partition Tolerance): the system works well despite physical network partitions.

ACID-Systems:
- AC-SYSTEMS: RDBMSs (MySQL, Postgres, ...)

AP-Systems:
- AP-SYSTEMS: Dynamo, CouchDB

CP-Systems:
- CP-SYSTEMS: Redis, MongoDB

Key/Value Stores:
- Redis
- MongoDB

Document Stores:
- MongoDB

Big Data Management and Analytics
NoSQL Data Models

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
NoSQL Data Models

Wide Column Stores:

Key Space

Column Family

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Column Name</th>
<th>Value</th>
<th>Column Name</th>
<th>Value</th>
<th>Column Name</th>
<th>Value</th>
</tr>
</thead>
</table>

Column

Column Family
### NoSQL Data Models

#### Wide Column Stores:

**Key Space „Edibles“**

**Column Family „Fruit“**

<table>
<thead>
<tr>
<th></th>
<th>color</th>
<th>weight</th>
<th>variety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple</td>
<td>„green“</td>
<td>95</td>
<td>„Granny Smith“</td>
</tr>
<tr>
<td>Cherry</td>
<td>„red“</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lemon</td>
<td>„yellow“</td>
<td>50</td>
<td>„Egypt“</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>„sour“</td>
</tr>
</tbody>
</table>

**Column Family „Vegetable“**

<table>
<thead>
<tr>
<th></th>
<th>2015-08-11</th>
<th>2015-08-12</th>
<th>…</th>
<th>2015-09-21</th>
</tr>
</thead>
<tbody>
<tr>
<td>Carrot</td>
<td>65</td>
<td>50</td>
<td>…</td>
<td>87</td>
</tr>
</tbody>
</table>
NoSQL Data Models

Wide Column Stores:

- developed by Facebook, Apache project since 2009
- cluster Architecture:
  - P2P system (ordered as rings)
  - Each node plays the same role (decentralized)
  - Each node accepts read/write operations
- user access through nodes via Cassandra Query Language (CQL)
NoSQL Data Models

Wide Column Stores:  

Consistency

• tunable Data Consistency (choosable per operation)

• read repair: if stale data is read, Cassandra issues a read repair → find most up-to-date data and update stale data

• generally: Eventually consistent

• main focus on availability!
Big Picture

**CAP Theorem:**

- **C** (Consistency)
- **A** (Availability)
- **P** (Partition Tolerance)

**ACID**

- All clients always have the same view of the data
- Each client can always read and write
- The system works well despite physical network partitions

**AC-Systems**
- RDBMSs (MySQL, Postgres, …)

**AP-Systems**
- Dynamo
- Cassandra

**CP-Systems**
- Redis
- MongoDB
- HBase

**Key/Value Stores**
- Document Stores
- Wide Column Stores
NoSQL Data Models

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)
NoSQL Data Models

Graph Databases:

Alice is a friend of Bob and vice versa. They both love the movie „Titanic“.

name = „Alice“

name = „Bob“

title = „Titanic“
NoSQL Data Models

Graph Databases:

Alice is a friend of Bob and vice versa. They both love the movie “Titanic”.

Person
name = “Alice”

Person
name = “Bob”

Movie
title = “Titanic”
Graph Databases:

Alice is a friend of Bob and vice versa. They both love the movie “Titanic”.

Person
name = “Alice”

is a friend of

Person
name = “Bob”

loves

Movie
title = “Titanic”
NoSQL Data Models

Graph Databases:

• master-slave replication (no partitioning!)

• consistency: eventual consistency (tunable to Immediate consistency)

• support of ACID Transactions

• cypher Query Language

• schema-optional
Big Picture

**CAP Theorem:**

- **C** (Consistency): all clients always have the same view of the data.
- **A** (Availability): each client can always read and write.
- **P** (Partition Tolerance): the system works well despite physical network partitions.

**ACID**

- AC-Systems (RDBMSs): MySQL, Postgres, ...
  - Neo4J
- AP-Systems: Dynamo, Cassandra
- CP-Systems: Redis, MongoDB, HBase

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

Big Data Management and Analytics