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 LTP scheme from Datenbanksysteme I and an object of type Supply:

**How to incorporate the data bundled in the object Supply into the DB?**
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
History

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

• Data stored as objects (including data and methods)
• Avoidance 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
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
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 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

Levels of Consistency:

- Eventual Consistency
- Monotonic Read Consistency
  - M.R.C. + R.Y.O.W.
- Immediate Consistency
- Strong Consistency
  - Transactions
- Read-Your-Own-Writes

Big Data Management and Analytics
About the concepts behind NoSQL Databases

Levels of Consistency:

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

• **Monotononic 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

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

Brewer’s CAP Theorem:

- **C**onsistency
- **A**vailability
- **P**artition Tolerance

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

DB-Systems allowed 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

**ACID**

All clients always have the same view of the data

The system works well despite physical network partitions

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

AP-Systems

Each client can always read and write

CP-Systems

Big Data Management and Analytics
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 consistency* to *serializibility*

• Some systems support ordering of keys, which enables efficient querying, like range queries

• Some systems support in-memory data maintenance, some use disks

→ There are very heterogeneous systems
NoSQL Data Models

Key/Value Stores - Redis:

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

• Support of complex types like lists, sets, hashes, …

• Support of many \textit{atomic} operations

\begin{verbatim}
>> 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
\end{verbatim}
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:
NoSQL Data Models

Key/Value Stores – The Redis cluster model:

Master Nodes
- A
- B
- C

Hash slots
- 0
- 5000
- 5001
- 10000
- 10001
- 14522

Slave Nodes
- A'
- B'
- C'

Replicated Hash slots
- 0
- 5000
- 5001
- 10000
- 10001
- 14522

Hash slots 5001 – 10000 cannot be used anymore

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

CAP Theorem:

C
ACID
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

Key/Value Stores

AC-SYSTEMS
- RDBMSs (MySQL, Postgres, ...)

CP-SYSTEMS
- Redis

AP-SYSTEMS
- Dynamo

Key/Value Stores

ACID BASE
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

→ 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 wrt to 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** (Available): Each client can always read and write.
- **P** (Partition Tolerance): The system works well despite physical network partitions.

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

**AP Systems:**
- Dynamo
- CouchDB

**CP Systems:**
- Redis
- MongoDB

**Key/Value Stores:**
- Redis
- MongoDB

**Document Stores:**
- MongoDB

**Big Data Management and Analytics**

35
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

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

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

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

Column Family
### NoSQL Data Models

#### Wide Column Stores:

**Key Space „Edibles“**

**Column Family „Fruit“**

<table>
<thead>
<tr>
<th>Fruit</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>50</td>
<td>„sour“</td>
</tr>
<tr>
<td>Lemon</td>
<td>„yellow“</td>
<td>50</td>
<td>„Egypt“</td>
</tr>
</tbody>
</table>

**Column Family „Vegetable“**

<table>
<thead>
<tr>
<th>Carrot</th>
<th>2015-08-11</th>
<th>2015-08-12</th>
<th>…</th>
<th>2015-09-21</th>
</tr>
</thead>
<tbody>
<tr>
<td></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!
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 systems:
- AC-Systems
  - RDBMSs (MySQL, Postgres, …)
AC-Systems
- AP-Systems
  - Dynamo
  - CouchDB
  - Cassandra
AP-Systems
- CP-Systems
  - Redis
  - MongoDB
  - HBase
CP-Systems

Key/Value Stores:
- Redis
- Dynamo

Document Stores:
- CouchDB
- Cassandra

Wide Column Stores:
- MongoDB
- HBase
- RDBMSs (MySQL, Postgres, …)
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*)

![Graph Databases Diagram](image)
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“.

<table>
<thead>
<tr>
<th>Person</th>
<th>Person</th>
<th>Movie</th>
</tr>
</thead>
<tbody>
<tr>
<td>name = „Alice“</td>
<td>name = „Bob“</td>
<td>title = „Titanic“</td>
</tr>
</tbody>
</table>
NoSQL Data Models

Graph Databases:

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

![Graph database diagram](image-url)
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
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
- CouchDB
- Cassandra

**CP-Systems**

- Redis
- MongoDB
- HBase

**Key/Value Stores**

- Dynamo
- CouchDB
- Cassandra

**Document Stores**

- Redis
- MongoDB
- HBase

**Wide Column Stores**

- Redis
- MongoDB
- HBase

**Graph Databases**

- Neo4J