Data Management in Large-Scale Distributed Systems
NoSQL Databases Fundamentals

Thomas Ropars

thomas.ropars@univ-grenoble-alpes.fr

http://tropars.github.io/

2021
References

• The lecture notes of V. Leroy
• The lecture notes of F. Zanon Boito
• Designing Data-Intensive Applications by Martin Kleppmann
  ▶ Chapters 2 and 7
In this lecture

- Motivations for NoSQL databases
- ACID properties and CAP Theorem
- A landscape of NoSQL databases
Introduction

Why NoSQL?

Transactions, ACID properties and CAP theorem

Data models
Common patterns of data accesses

Large-scale data processing

• Batch processing: Hadoop, Spark, etc.
• Perform some computation/ transformation over a full dataset
• Process all data

Selective query

• Access a specific part of the dataset
• Manipulate only the needed data
  ▶ 1 record among millions
• Main purpose of a database system
Processing / Database Link

- **Batch Job** (Hadoop, Spark)
  - Load data
  - Write results
  - Serve queries
  - Insert records

- **Stream Job** (Spark, Storm)
  - Write results
  - Serve queries
  - Insert records

**Applications**
- Application 1
- Application 2
- Application 3

**Database**

**Examples**
- e.g. sentiment analysis
- e.g. Twitter trends page
Different types of databases

• So far we used HDFS
  – A file system can be seen as a very basic database
  – Directories / files to organize data
  – Very simple queries (file system path)
  – Very good scalability, fault tolerance ...

• Other end of the spectrum: Relational Databases
  – SQL query language, very expressive
  – Limited scalability (generally 1 server)
Size / Complexity

Graph DB
Relational DB
Document DB
Column DB
Key/Value DB
Filesystem
The NoSQL Jungle

<table>
<thead>
<tr>
<th>Document Database</th>
<th>Graph Databases</th>
</tr>
</thead>
<tbody>
<tr>
<td>Couchbase</td>
<td>Neo4j</td>
</tr>
<tr>
<td>MarkLogic</td>
<td>InfiniteGraph</td>
</tr>
<tr>
<td>mongoDB</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Wide Column Stores</th>
<th>Key-Value Databases</th>
</tr>
</thead>
<tbody>
<tr>
<td>redis</td>
<td>HYPERTABLE</td>
</tr>
<tr>
<td>Aerospike</td>
<td>Accumulo</td>
</tr>
<tr>
<td>riak</td>
<td>Cassandra</td>
</tr>
<tr>
<td>Hbase</td>
<td>SimpleDB</td>
</tr>
</tbody>
</table>

© cloud4l.com  http://www.ariannara.com
Agenda

Introduction

Why NoSQL?

Transactions, ACID properties and CAP theorem

Data models
Relational databases

SQL

- Born in the 70’s – Still heavily used
- Data is organized into relations (in SQL: tables)
- Each relation is an unordered collection of tuples (rows)
- Foreign keys are used to define the relationships among the tables
About SQL

Advantages

• Separate the data from the code
  ▶ High-level language
  ▶ Space for optimization strategies

• Powerful query language
  ▶ Clean semantics
  ▶ Operations on sets

• Support for transactions
Motivations for alternative models

Some limitations of relational databases

- **Performance and scalability**
  - Difficult to partition the data (in general run on a single server)
  - Need to scale up to improve performance

- **Lack of flexibility**
  - Will to easily change the schema
  - Need to express different relations
  - Not all data are well structured

- **Few open source solutions**

- **Mismatch between the relational model and object-oriented programming model**

Illustration of the object-relational mismatch

Figure by M. Kleppmann

Figure: A CV in a relation database
Illustration of the object-relational mismatch
Figure by M. Kleppmann

```json
{
    "user_id" : 251,
    "first_name": "Bill",
    "last_name": "Gates",
    "summary": "Co—chair of the Bill & Melinda Gates; Active blogger.",
    "region_id": "us:91",
    "industry_id": 131,
    "photo_url": "/p/7/000/253/05b/308dd6e.jpg",
    "positions": [
        {
            "job_title": "Co—chair",
            "organization": "Bill & Melinda Gates Foundation"
        },
        {
            "job_title": "Co—founder, Chairman",
            "organization": "Microsoft"
        }
    ],
    "education": [
        {
            "school_name": "Harvard University",
            "start": 1973,
            "end": 1975
        },
        {
            "school_name": "Lakeside School, Seattle",
            "start": null,
            "end": null
        }
    ],
    "contact_info": {
        "blog": "http://thegatesnotes.com",
        "twitter": "http://twitter.com/BillGates"
    }
}
```

Figure: A CV in a JSON document
About NoSQL

What is NoSQL?

- A hashtag
  - NoSQL approaches were existing before the name became famous
- No SQL
- New SQL
- Not only SQL
  - Relational databases will continue to exist alongside non-relational datastores
About NoSQL

A variety of NoSQL solutions

• Key-Value (KV) stores
• Wide column stores (Column family stores)
• Document databases
• Graph databases

Difference with relational databases
There are several ways in which they differ from relational databases:

• Properties
• Data models
• Underlying architecture
Agenda

Introduction

Why NoSQL?

Transactions, ACID properties and CAP theorem

Data models
About transactions

The concept of transaction

- Groups several read and write operations into a logical unit
- A group of reads and writes are executed as one operation:
  - The entire transaction succeeds (commit)
  - or the entire transaction fails (abort, rollback)
- If a transaction fails, the application can safely retry

Why do we need transactions?

- Crashes may occur at any time
  - On the database side
  - On the application side
  - The network might not be reliable
- Several clients may write to the database at the same time
About transactions

The concept of transaction

• Groups several read and write operations into a logical unit
• A group of reads and writes are executed as one operation:
  ▶ The entire transaction succeeds (commit)
  ▶ or the entire transaction fails (abort, rollback)
• If a transaction fails, the application can safely retry

Why do we need transactions?

• Crashes may occur at any time
  ▶ On the database side
  ▶ On the application side
  ▶ The network might not be reliable
• Several clients may write to the database at the same time
ACID

ACID describes the set of safety guarantees provided by transactions

- Atomicity
- Consistency
- Isolation
- Durability

Having such properties make the life of developers easy, but:

- ACID properties are not the same in all databases
  - It is not even the same in all SQL databases

- NoSQL solutions tend to provide weaker safety guarantees
  - To have better performance, scalability, etc.
ACID: Atomicity

Description

- A transaction succeeds completely or fails completely
  - If a single operation in a transaction fails, the whole transaction should fail
  - If a transaction fails, the database is left unchanged
- It should be able to deal with any faults in the middle of a transaction
- If a transaction fails, a client can safely retry

In the NoSQL context:

- Atomicity is still ensured
ACID: Consistency

Description

• Ensures that the transaction brings the database from a valid state to another valid state
  ▶ Example: Credits and debits over all accounts must always be balanced

• It is a property of the application, not of the database
  ▶ The database cannot enforce application-specific invariant
    • It is the responsibility of the programmer to issue transactions that make sense.
  ▶ The database can check some specific invariant
    • A foreign key must be valid

In the NoSQL context:

• Consistency is (often) not discussed
ACID: Durability

Description

• Ensures that once a transaction has committed successfully, data will not be lost
  ▶ Even if a server crashes (flush to a storage device, replication)

In the NoSQL context:

• Durability is also ensured
ACID: Isolation

Description

• Concurrently executed transactions are isolated from each other
  ▶ We need to deal with concurrent transactions that access the same data

• Serializability
  ▶ High level of isolation where each transaction executes as if it was the only transaction applied on the database
    • As if the transactions are applied serially, one after the other
  ▶ Many SQL solutions provide a lower level of isolation
    • Example: Read committed (see next slides)

In the NoSQL context:

• What about the CAP theorem?
More about isolation
Alternative (weak) isolation levels

Read Committed

- **No dirty reads**: A read only sees data that has been committed
- **No dirty writes**: A write only overwrites data that has been committed.

Many SQL databases implement this level of isolation.
Example of allowed execution with Read Committed

Transaction to be executed (Increment a counter)

Begin transaction
Read count
count = count + 1
Write count
End transaction
Example of allowed execution with Read Committed

Transaction to be executed (Increment a counter)

Begin transaction
Read count
count = count + 1
Write count
End transaction

ClientA

DB

count=0
read count
write 1
commit

count=1

ClientB

read count
write 1
commit

No dirty writes does not prevent all inconsistencies.
Example of allowed execution with Read Committed

2 Transactions executed concurrently

On the accounts of Alice:

- initial state: $500 of the two accounts
- Transaction 1: Transfer $100 from account B to A
- Transaction 2: Check balance.
Example of allowed execution with Read Committed

2 Transactions executed concurrently

On the accounts of Alice:

- initial state: $500 of the two accounts
- Transaction 1: Transfer $100 from account B to A
- Transaction 2: Check balance.
Example of allowed execution with Read Committed

2 Transactions executed concurrently

On the accounts of Alice:

- initial state: $500 of the two accounts
- Transaction 1: Transfer $100 from account B to A
- Transaction 2: Check balance.

Some inconsistencies can also be observed with no dirty reads.
The CAP theorem

3 properties of databases

• Consistency
  ▶ What guarantees do we have on the value returned by a read operation?
  ▶ It strongly relates to Isolation in ACID (and not to consistency)

• Availability
  ▶ The system should always accept updates

• Partition tolerance
  ▶ The system should be able to deal with a partitioning of the network

Comments on CAP theorem

• Was introduced by E. Brewer in its lectures (beginning of years 2000)
• Goal: discussing trade-offs in database design
What does the CAP theorem says?

The theorem
It is impossible to have a system that provides Consistency, Availability, and partition tolerance.

How it should be understood:
• Partitions are unavoidable
  ▶ It is a fault, we have no control on it

• We need to choose between availability and consistency
  ▶ In the CAP theorem:
    • Consistency is meant as *linearizability* (the strongest consistency guarantee)
    • Availability is meant as *total availability*
  ▶ In practice, different trade-offs can be provided
The intuition behind CAP

- Let inconsistencies occur? (No C)
- Stop executing transactions? (No A)
The intuition behind CAP

- Let inconsistencies occur? (No C)
- Stop executing transactions? (No A)

Note that in a centralized system (non-partitioned relational database), no need for Partition tolerance

- We can have Consistency and Availability
The impact of CAP on ACID for NoSQL

source: E. Brewer

The main consequence

• No NoSQL database with strong Isolation

Discussion about other ACID properties

• Atomicity
  ▶ Each side should ensure atomicity

• Durability
  ▶ Should never be compromised
A vision of the NoSQL landscape

To be read with care:

- Solutions often provide a trade-off between CP and AP
- A single solution may offer a different trade-off depending on how is configured.
- We don’t pick two!
Agenda

Introduction

Why NoSQL?

Transactions, ACID properties and CAP theorem

Data models
Key-Value store

- Data are stored as key-value pairs
  - The value can be a data structure (e.g., a list)

- In general, only support single-object transactions
  - In this case, key-value pairs

- Examples:
  - Redis
  - Voldemort

- Use case:
  - Scalable (in-memory) cache for data
  - Some solutions ensure durability by writing data to disk
Key-value store

Image by J. Stolfi

![Diagram of a key-value store with keys and buckets](image)
Column family stores

• Data are organized in rows and columns (Tabular data store)
  ▶ The data are arranged based on the rows
  ▶ Column families are defined by users to improve performance
    • Group related columns together

• Only support single-object transactions
  ▶ In this case, a row

• Examples:
  ▶ BigTable/HBase
  ▶ Cassandra

• Use case:
  ▶ Data with some structure with the goal of achieving scalability and high throughput
  ▶ Provide more complex lookup operations than KV stores
Column family stores

Note that a row does not need to have an entry for all columns
Document databases

• Data are organized in Key-Document pairs
  ▶ A document is a nested structure with embedded metadata
  ▶ No definition of a global schema
    • The schema is implicit and not enforced by the database
    • *Schema-on-read*
  ▶ Popular formats: XML, JSON

• Only support single-object transactions
  ▶ In this case, a document or a field inside a document

• Examples:
  ▶ MongoDB
  ▶ CouchDB

• Use case:
  ▶ An alternative to relational databases for structured data
  ▶ Offer a richer set of operations compared to KV stores:
    Update, Find, etc.
A document can have one or more documents inside.

```json
{
  "_id": ObjectId("51c4218"),
  "name": "Claudia",
  "NumberKids": 3,
  "isActive": true,
  "interests": ["swimming", "tennis"]
  "favoriteCountries": [
    {
      "name": "France",
      "capital": "Paris"
    },
    {
      "name": "Japan"
    }
  ],
  "_id": 2,
  "name": "Rubby"
  "friends": 354,
  "favorite Country": {
    "name": "Italy",
    "capital": "Rome"
  }
}
```
Graph DB

• Represent data as graphs
  – Nodes / relationships with properties as K/V pairs
Graph DB: Neo4j

- Rich data format
  - Query language as pattern matching
  - Limited scalability
    - Replication to scale reads, writes need to be done to every replica

Cypher Query Language

```
MATCH (Person { name: "Dan"}) -[:KNOWS]-> (Person { name: "Ann"})
```

Node
- Label
- Property

Neo4j

32
On the Many-to-one relationship

Many-to-one relationship

- Many items may refer to the same item
- Example: Many people went to the same university

Relational vs Document DB
On the Many-to-one relationship

Many-to-one relationship

- Many items may refer to the same item
- Example: Many people went to the same university

Relational vs Document DB

- Relational databases use a foreign key
  - Consistency and low memory footprint (normalization)
  - Easy updates and support for joins
  - Difficult to scale

- Document databases duplicate data
  - Efficient read operations
  - Easy to scale
  - Higher memory footprint and updates are more difficult (risk of consistency issues)
    - Transactions on multiple objects could be very useful in this case
  - Some document databases support document references (equivalent to foreign keys)
Additional references

Suggested reading
