NoSQL Databases

Polytech Nice-Sophia, SI4

Pierre Monnin

pierre.monnin@inria.fr

Special thanks to **Dr. Sabeur Aridhi**, **Dr. Jörn Franke** and **Dr. Nicolas Travers** for providing their materials

Chapter 2:
Document Databases

Overview

- **Key idea:**
  - *Store/Load/Modify documents with arbitrary fields in a database*
  - *Simple text search functionality*

- **Difference from Column Storage:**
  - *You do not need to define any schema – it is defined by the data you insert*

Schema (1/2)

- Documents bring their own schema – they are self-describing
  - *Formats: XML, JSON etc.*
- No need to define data types – detected automatically
- Indexes are possible
- Documents are part of collections/databases
- Documents are versioned
  - *Keep in mind to clean them up regularly*

Schema (2/2)

- References („foreign keys“) are managed by the application
- Documents can share parts of their schema, but none of it as well
- Do not let documents grow too large to avoid swapping to disk
  → *normalization makes sense*
  → *think about what your application really needs*

Schema: Example /Json

–**Document 1**

–{

„username“ : „maxmustermann“,

„likes“ : [„soccer“, „photography“]

–}

–**Document 2**

–{

„username“ : „marthamusterfrau“,

„profession“: „CEO“,

„businesstrips“ : [„Bangkok“, „New York“, „Brisbane“]

„address“: {

street: „1569 Broadway“,

city: „New York“

}

–}

Type of Queries (1/2)

- Different from SQL
- Depends on document store
- Usually document-oriented
  - *Creating, Modifying, Retrieval (Simple Search)*

Type of Queries (2/2)

- Simple aggregations
- Java/Javscript syntax
- Additional functionality to implement any kind of query
  - *Map/Reduce Paradigm*

Distributed Persistence

- **Replication**
  - *–Usually Master/Slave paradigm*
  - *–Arbiter: elects new master in case of failure*
  - *–You can define read/write-quorums for each command you execute*
- **Sharding**
  - *–By a key in a document*
  - *–Range-Sharding: define a range which should be stored on a node*
    - *Example: All documents from 01.01.2014 – 31.12.2014*
    - *Good for data processing on specific nodes*
  - *–Hash-Sharding: ensures that data is evenly distributed across nodes*
    - *Good for heavy write operations*

Transactional Guarantees

- Operations on a single document are always atomic (MongoDB)

- Update document only if version number in update is the same as version number in document (CouchDB)
  - *Example: Update 1 {**Version: 1**, title: „this is a test“} can be applied Document {**Version: 1**, title „this is nothing“}*
  - *Similar to vector clock approach*

- Multiple documents:
  - *Create your own transaction mechanism using 2PC (2 Phase Commit) protocol*
  - *Alternatively: Have one document with nested documents (→ not always recommended)*

Technologies

- **MongoDB**
  - *Very popular database*
  - *Master/Slave Replication*
  - *GNU AGPL V3.0*

- **CouchDB**
  - *Similar functionality as MongoDB*
  - *Master/Master replication*
  - *Apache License*

Use Cases

- Web-Analytics
  - *Store all events on the web page (clicks, scroll, touch, …) - usually already JSON-objects*
- Catalogue/Shopping applications
  - *Products with different fields*
- Reviews / Tweets

Evaluation

- Document stores are very useful where a flexible schema is required
- Good for prototyping – no need to think much about schema beforehand
- Fixed schemas / data types can offer more performance for specific use cases
- Relations between documents have to be managed by the application

Document Databases - MongoDB Exercises

Overview

- MongoDB was initially released in 2009
- Open Source (GNU AGPL v3.0)
- Name comes from hu**mongo**us
- Typical document storage
- Schema is defined by the data you insert
- JavaScript is used to create more complex queries

Schema

- Collections (~ databases): Contain documents

- Documents
  - *Internal representation: JSON Format*
  - *Various formats can be imported*

- Further schema details are defined by the document you insert

Excursus: JSON-Format

- JavaScript Object Notation
- Relevant for NoSQL and Web applications
- You define data types implicitly
- Encoding: Often UTF-8
- Example

```
{
 "name": "Homer",
 "isPoet": true,
 "age": 35,
 "height": 180.1,
 "address": {
  "city": "Troy"
"country": "Turkey"
}
}
```

Queries

- Syntax similar to Javascript
- Aggregation calculation support several different type of aggregations for analytics
- Geospatial Queries
  - *Example: Give me all tweets near my location*
- You can use MapReduce for advanced calculations

Queries: CRUD

- CRUD = Create Read Update Delete
  - *Common term for basic data operations*

- **Create**
  - *db.collection.insert(document)*
  - *Example: db.test.insert({_id:10, type:"test"})*

- **Read** (can match multiple documents!)
  - *db.collection.find(criteria)*
  - *Example: db.collection.find({_id:10})*

- **Update** (can match multiple documents!)
  - *db.collection.update(criteria, update, options)*
  - *Example: db.collection.update({_id:10},{type:"final"}, {multi:false})*

- **Delete**
  - *db.collection.remove(criteria) (can remove multiple documents!)*
  - *Example: db.collection.remove({_id:10})*

Query criteria

- You can match fields exactly in a document
  - *Example ({type:“test“}) => all documents with the key „type“ and the corresponding value „test“*
- You can use wildcards/regular expressions
  - *Example ({type:{$regex:“te*“,$options:“i“}}) => all documents having a type starting with te*
- You can use > ($gt) or < ($lt) for numbers
  - *Example ({price:{$gt:100})*
- You can use OR conditions (default AND)
  - *Example ({$or: [{type:“test},{price:{$gt:100}}]})*

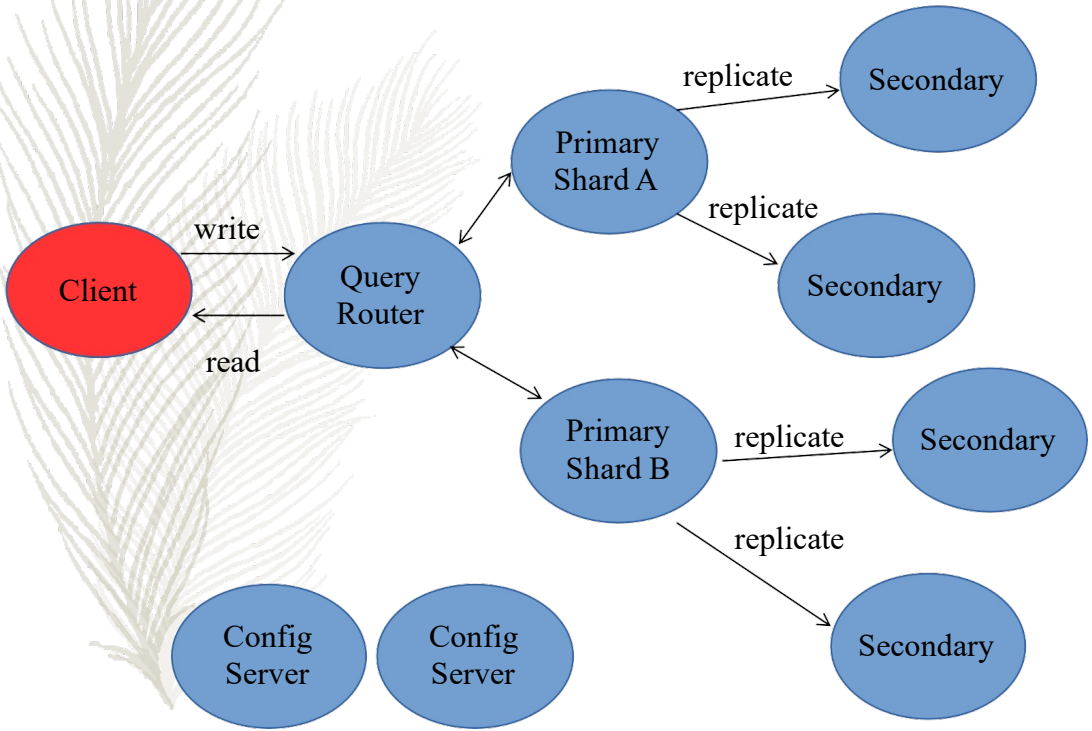
Queries: Aggregation

- **Simple**
  - *Count: Count all documents matching criteria*
    - *Example: db.collection.find(criteria).count()*
  - *Distinct: Returns all unique value for a given field*
    - *Example: db.collection.distinct(„type“)*
  - *Group (Warning: Not usable for sharded collections, limit of the documents returned)*
- **Aggregation Pipeline**
  - *You can define a pipeline of several aggregation commands (e.g. sum, avg, first, last, max, min, ...)*
  - *MongoDB optimizes execution*
- **MapReduce**
  - *Full flexibility, but more complex*

Transactions

- Only on document level
- Not across documents or collections
- As always: You can develop any transaction logic on top

Architecture



Distributed Persistence

- **Replication**
  - *Client writes to primary*
  - *Primary replicates to secondary*
  - *Client may read from secondaries for load balancing*
    - *May have outdated data*
  - *Primary fails → Config Servers support Secondary to elect a new primary among them*

- **Sharding**
  - *Query router routes the queries to the correct shard*
  - *You define an attribut of a document as a shard key*
    - *Hash-based shard vs. range-based shard*

Interfaces

- JDBC
- Libraries for various programing languages
- mongo – shell program

Excursus: Loading data into mongodb

- Mongo-importer (for documents in JSON/CSV)
- ETL Tools
  - *Pentaho Kettle/Spoon*
- Write your own import tool

Who uses it?

- LinkedIn: Learning Platform
- Expedia: Travel planing
- SAP: Enterprise Content Management
- MTV: Content Management System
- Springer: Real-time analytics for downloads of journals / book chapters
- Github: Reporting
- Nearley: user interaction and activities
- O2: Several applications
- E-Bay: Various Services