Big Data Analytics

B. Distributed Storage / B.3 NoSQL Databases

Lars Schmidt-Thieme

Information Systems and Machine Learning Lab (ISMLL)
Institute for Computer Science
University of Hildesheim, Germany

Syllabus

Tue.	9.4.	(1)	0. Introduction
			A. Parallel Computing
Tue.	16.4.	(2)	A.1 Threads
Tue.	23.4.	(3)	A.2 Message Passing Interface (MPI)
Tue.	30.4.	(4)	A.3 Graphical Processing Units (GPUs)
			B. Distributed Storage
Tue.	7.5.	(5)	B.1 Distributed File Systems
Tue.	14.5.	(6)	B.2 Partioning of Relational Databases
Tue.	21.5.	(7)	B.3 NoSQL Databases
			C. Distributed Computing Environments
Tue.	28.5.	(8)	C.1 Map-Reduce
Tue.	4.6.	—	— *Pentecoste Break* —
Tue.	11.6.	(9)	C.2 Resilient Distributed Datasets (Spark)
Tue.	18.6.	(10)	C.3 Computational Graphs (TensorFlow)
			D. Distributed Machine Learning Algorithms
Tue.	25.6.	(11)	D.1 Distributed Stochastic Gradient Descent
Tue.	2.7.	(12)	D.2 Distributed Matrix Factorization
Tue.	9.7.	(13)	Questions and Answers

Outline

1. Introduction

2. Key-Value Stores

3. Document Databases

4. Document Databases: Partitioning

5. Other NoSQL Databases

Outline

1. Introduction

2. Key-Value Stores

3. Document Databases

4. Document Databases: Partitioning

5. Other NoSQL Databases

Relational vs Big-Data Technologies

- **Structure:**
  - relational data bases: data is **structured**.
  - big data applications: data often is **raw**.
- **Process:**
  - relational data bases: initially created for **transactional processing**.
  - big data applications: **analytical processing**.
- **Entities:**
  - relational data bases: big if it has **many entities** (rows) of a kind.
  - big data applications: the number of entities is not necessary high, the amount of information that exist for entities may be large.
- **horizontal scaling:**
  - relational data bases: costs for adding new nodes is high.
  - big data applications: scaling should be accomplished inexpensively by **adding new nodes**.

NoSQL Databases

- A NoSQL or Not Only SQL database provides a mechanism for storage and retrieval of data that is modeled in means **other than the tabular relations** used in relational databases.
- Motivations for this approach include
  - **simplicity of design** and
  - **horizontal scaling**.
- The data structure differs from the RDBMS, and therefore
  - some operations are faster in NoSQL, and
  - some operations are faster in RDBMS.
- Most NoSQL stores lack true ACID transactions.
  - ACID: atomicity, consistency, isolation, durability

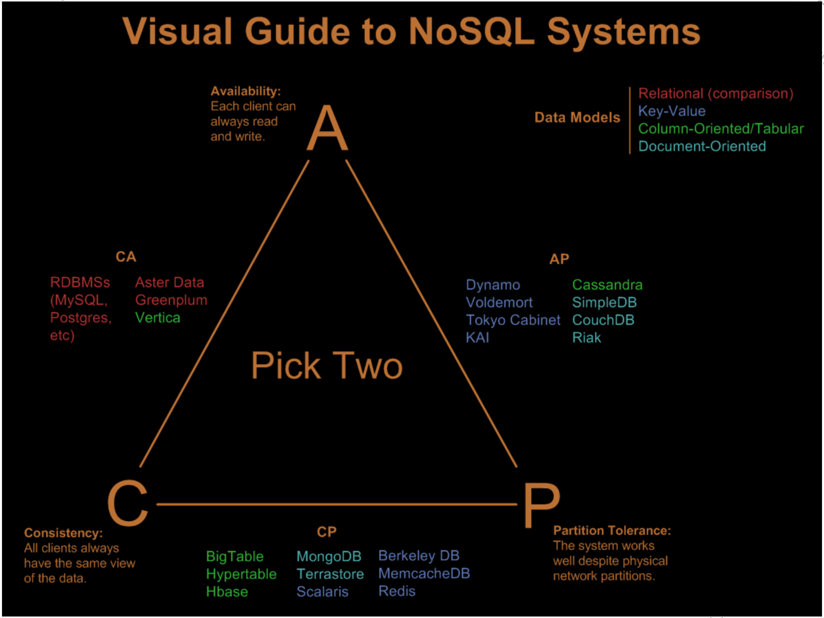
NoSQL Databases / Types and Implementations

- **Key-value:** Redis^{OS}, Amazon DynamoDB, Memcached^{OS}
- **Document:** MongoDB^{OS}, Amazon DynamoDB, Couchbase^{OS}
- **Graph:** Neo4J^{OS}, MS CosmosDB, OrientDB^{OS}
- **Column:** Cassandra^{OS}, HBase^{OS}, MS CosmosDB
- **Object:** Caché, Versant Object Database, ObjectStore, Db4o^{OS}

Note: Popularity ranking of DBMS by type: `https://db-engines.com/en/ranking/`;
^{OS}=open source

BASE vs ACID

- **CAP-theorem**: in the presence of **network partitions**, a DBMBs has to choose between
  - **Consistency**: all reads always return the last write — then clients cannot read/write during a network partition.
  - **Availability**: clients always can read/write — then what they receive might not be consistent at all times
- BASE design:
  - **Basically available**: the system aims for high availability.
  - **Soft state**: the state of the system may change over time, even without input. (due to the eventual consistency model).
  - **Eventual consistency**: the system may not be consistent at all times, but will become consistent over time.



http://blog.scottlogic.con

Outline

1. Introduction

2. Key-Value Stores

3. Document Databases

4. Document Databases: Partitioning

5. Other NoSQL Databases

Key–Value stores

- Key–Value stores use the **associative array** (dictionary) as their fundamental data model.
  - Data is represented as a collection of key–value pairs.
- One of the simplest non-trivial data models.

Example:

```
{
"Great Expectations": "John",
"Pride and Prejudice": "Alice",
"Wuthering Heights": "Alice"
}
```

Outline

1. Introduction

2. Key-Value Stores

3. Document Databases

4. Document Databases: Partitioning

5. Other NoSQL Databases

Document Databases

- Data abstraction:
  - relational databases: "relations" (= "tables").
  - document databases: "document".
- Documents are (possibly nested) **dictionaries**.
  - like Python dicts.
  - Documents are not required to have all the same fields (aka sections, slots, parts).
  - Documents are **schemaless**.
- Documents are identified by an ID, e.g.,
  - marked by a special type **ObjectId**,
  - stored by a special key **_id**.
- References are modeled by **foreign keys**.
  - or avoided by using **embedded documents**.

Example 1 / Schema-less

```
{
    _id: ObjectId(7df78ad8902c),
    FirstName: "Bob",
    Age: 35,
    Address: "5 Oak St.",
    Hobby: "sailing"
}
```

```
{
    _id: ObjectId(5df78ad8902c),
    FirstName: "Jonathan",
    Age: 37,
    Address: "15 Wanamassa Point Road",
    Languages: [ 'English', 'German' ]
}
```

Example 2 / Foreign Keys

```
{
    _id: ObjectId(5df78ad8902c),
    FirstName: "Jonathan",
    Age: 37,
    Address: "15 Wanamassa Point Road",
    Children: [ ObjectId(5df78ad89020), ObjectId(5df78ad89021),
                ObjectId(5df78ad89022), ObjectId(5df78ad89023) ]
},{
    _id: ObjectId(5df78ad89020),
    FirstName: "Michael",
    Age: 10,
},{
    _id: ObjectId(5df78ad89021),
    FirstName: " Jennifer ",
    Age: 8,
},{
    _id: ObjectId(5df78ad89022),
    FirstName: "Samantha",
    Age: 5,
},{
    _id: ObjectId(5df78ad89023),
    FirstName: "Elena",
    Age: 2,
}
```

Example 3 / Embedded Documents

```
{
    _id: ObjectId(5df78ad8902c),
    FirstName: "Jonathan",
    Age: 37,
    Address: "15 Wanamassa Point Road",
    Children: {
        { FirstName: "Michael", Age: 10 },
        { FirstName: " Jennifer ", Age: 8 },
        { FirstName: "Samantha", Age: 5 },
        { FirstName: "Elena", Age: 2}
    }
}
```

Organization

- Documents are addressed in the database via a **unique key**.
- Documents can be retrieved by their
  - key
  - content
- Documents are organized through
  - Collections
  - Tags
  - Non-visible Metadata
  - Directory hierarchies
  - Buckets

RDBMS vs. Document Database Terminology

relational database	document database
Database	Database
Table	Collection
Tuple/Row	Document
Column	Field
Primary key	Primary key (e.g., default key _id in mongodb)
Foreign key	Foreign key

Inserting Documents

- **insert** allows to insert a new document.
- **insertMany** allows to insert many documents.

```
db.inventory.insertMany([
   { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
   { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },
   { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
   { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },
   { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }
]);
```

Note: Most examples taken from MongoDB [2017].

Query Documents (1/3)

- all documents of a collection:

```
db.inventory.find()
```

```
{ "_id" : ObjectId("59116d5340229e45bb5eea9a"), "item" : "journal", "qty" : 25, "size" : { "h" : 14, "w"
{ "_id" : ObjectId("59116d5340229e45bb5eea9b"), "item" : "notebook", "qty" : 50, "size" : { "h" : 8.5, "
{ "_id" : ObjectId("59116d5340229e45bb5eea9c"), "item" : "paper", "qty" : 100, "size" : { "h" : 8.5, "w"
{ "_id" : ObjectId("59116d5340229e45bb5eea9d"), "item" : "planner", "qty" : 75, "size" : { "h" : 22.85, "
{ "_id" : ObjectId("59116d5340229e45bb5eea9e"), "item" : "postcard", "qty" : 45, "size" : { "h" : 10, "w
```

- all documents with a property:

```
db.inventory.find({ status: "D"})
```

```
{ "_id" : ObjectId("59116d5340229e45bb5eea9c"), "item" : "paper", "qty" : 100, "size" : { "h" : 8.5, "w"
{ "_id" : ObjectId("59116d5340229e45bb5eea9d"), "item" : "planner", "qty" : 75, "size" : { "h" : 22.85, "
```

- all documents with several properties:

```
db.inventory.find({ status: "D", qty: 100})
```

```
{ "_id" : ObjectId("59116d5340229e45bb5eea9c"), "item" : "paper", "qty" : 100, "size" : { "h" : 8.5, "w"
```

Query Documents (2/3)

- instead of querying for exact value matches, one can use query operators:
  - **$lt**, **$gt**, **$lte**, **$gte**: numerical comparison

```
db.inventory.find({ qty: { $gte: 75 } } )
```

```
{ "_id" : ObjectId("59116d5340229e45bb5eea9c"), "item" : "paper", "qty" : 100, "size" : { "h" : 8.5, "w"
{ "_id" : ObjectId("59116d5340229e45bb5eea9d"), "item" : "planner", "qty" : 75, "size" : { "h" : 22.85, '
```

```
db.inventory.find({ qty: { $lte: 75, $gt: 25 } } )
```

```
{ "_id" : ObjectId("59116d5340229e45bb5eea9b"), "item" : "notebook", "qty" : 50, "size" : { "h" : 8.5, "\
{ "_id" : ObjectId("59116d5340229e45bb5eea9d"), "item" : "planner", "qty" : 75, "size" : { "h" : 22.85, '
{ "_id" : ObjectId("59116d5340229e45bb5eea9e"), "item" : "postcard", "qty" : 45, "size" : { "h" : 10, "w
```

  - //: regular expressions.

```
db.inventory.find({ item: /er$/ } )
```

```
{ "_id" : ObjectId("59116d5340229e45bb5eea9c"), "item" : "paper", "qty" : 100, "size" : { "h" : 8.5, "w"
{ "_id" : ObjectId("59116d5340229e45bb5eea9d"), "item" : "planner", "qty" : 75, "size" : { "h" : 22.85, '
```

Query Documents (3/3)

- **$or** or queries:

```
db.inventory.find({ $or: [ { qty: { $lte: 25 } }, { qty: { $gte: 100 } } ] } )
```

```
{ "_id" : ObjectId("59116d5340229e45bb5eea9a"), "item" : "journal", "qty" : 25, "size" : { "h" : 14, "w"
{ "_id" : ObjectId("59116d5340229e45bb5eea9c"), "item" : "paper", "qty" : 100, "size" : { "h" : 8.5, "w"
```

Query Documents (4/4)

- fields of nested documents are queried by dot syntax:

```
db.inventory.find({ "size.h": { $gte: 14 } } )
```

```
{ "_id" : ObjectId("59116d5340229e45bb5eea9a"), "item" : "journal", "qty" : 25, "size" : { "h" : 14, "w"
{ "_id" : ObjectId("59116d5340229e45bb5eea9d"), "item" : "planner", "qty" : 75, "size" : { "h" : 22.85, '
```

Advantages and Disadvantages

- avoiding foreign keys by **embedding documents**:
  - + expensive join operations are not required.
  - – possibly redundant information.
- **schemaless**:
  - + one can add a new field at any time.
  - – one never can be sure a field actually has a value.

Big Data Document Databases

Document databases are useful for big data for two main reasons:

1. Document databases can be **horizontally partitioned** / **sharded**.
    - Documents are distributed over different nodes.
    - Using a **partition/sharding function**
        - e.g., a hash function.
    - works exactly the same way as for RDBMS.

Big Data Document Databases

Document databases are useful for big data for two main reasons:

1. Document databases can be **horizontally partitioned** / **sharded**.
    - Documents are distributed over different nodes.
    - Using a **partition/sharding function**
        - e.g., a hash function.
    - works exactly the same way as for RDBMS.

2. Documents are **sparse representations**,
    - only some fields/keys have values,

    while tuples/rows are **dense**.
    - all columns have values stored explicitely
        - also NULL is explicitely stored.

Outline

1. Introduction

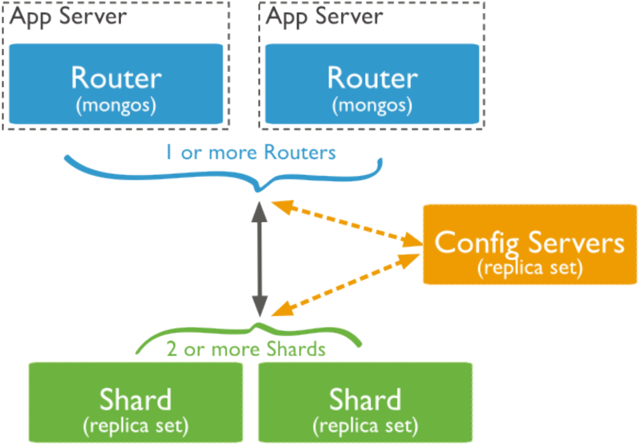
2. Key-Value Stores

3. Document Databases

4. Document Databases: Partitioning

5. Other NoSQL Databases

Sharded Document Database / Architecture



[source: https://docs.mongodb.com/manual/core/sharding-introduction/]

Mongo DB (from humongous – slang for enormous)

System Setup / Multiple Hosts

1. install mongodb on all hosts (h0, h1, h2, h3), e.g., for OpenSuSE:

```
zypper in mongodb
```

2. start a **config server** on one host (e.g., h0):

```
mongod --configsvr
```

   - default port is 27019

3. start a **query router** on one host (e.g., h0):

```
mongos --configdb localhost
```

   - default port is 27017

4. start a couple of **shard servers** on other hosts (e.g., h1, h2, h3):

```
mongod --shardsrv
```

   - default port is 27018

5. add shards:

```
mongo --host h0
sh.addShard("h1")
sh.addShard("h2")
```

System Setup / Single Hosts (for Testing)

1. install mongodb, e.g., for OpenSuSE:

```
zypper in mongodb
```

2. start a **config server**:

```
mongod --configsvr --dbpath db-configsrv/
```

   - config information will be stored in directory db-configsrv/.

3. start a **query router**:

```
mongos --configdb localhost:27019
```

   - query routers do not require any data path

4. start a couple of **shard servers**:

```
mongod --port 27021 --dbpath db-shardsrv1/
mongod --port 27022 --dbpath db-shardsrv2/
mongod --port 27023 --dbpath db-shardsrv3/
```

   - use different ports and data paths

5. add shards:

```
mongo
sh.addShard("localhost:27021")
sh.addShard("localhost:27022")
```

Database and Collection Setup

1. enable sharding per database

```
sh.enableSharding("mydb")
```

2. sharding a collection requires the sharding key to be indexed:

```
db.ijcnn1.createIndex( { _id: 1 } )
```

3. shard per collection:

```
sh.shardCollection("mydb.ijcnn1", { _id: 1 })
```

4. import data

```
mongoimport --db mydb --collection ijcnn1 --drop --file ijcnn1.json
```

   - data has to be in json format

- if a database is imported first and then sharded,
  it will be distributed across nodes automatically by the balancer.

Database and Collection Setup

- production databases also have to be replicated
  - both, data nodes and config nodes

Database and Collection Setup

- production databases also have to be replicated
  - both, data nodes and config nodes

- to work on sharded big data, one should
  - **not** query it from the central router nodes
    - rule of thumb: 1 PB moves through a 1 GB ethernet in ca. 100d.

Database and Collection Setup

- production databases also have to be replicated
  - both, data nodes and config nodes

- to work on sharded big data, one should
  - **not** query it from the central router nodes
    - rule of thumb: 1 PB moves through a 1 GB ethernet in ca. 100d.
  - **not** query it locally on the shard nodes
    - if data also is replicated, one would need to know which nodes to query
    - the balancer may move data during query time

Database and Collection Setup

- production databases also have to be replicated
  - both, data nodes and config nodes

- to work on sharded big data, one should
  - **not** query it from the central router nodes
    - rule of thumb: 1 PB moves through a 1 GB ethernet in ca. 100d.
  - **not** query it locally on the shard nodes
    - if data also is replicated, one would need to know which nodes to query
    - the balancer may move data during query time

- to work on sharded big data, one should
  use technologies provided by the database, e.g.,
  map-reduce.
  - for mongo: mappers and reducers have to be coded in javascript.

Outline

1. Introduction

2. Key-Value Stores

3. Document Databases

4. Document Databases: Partitioning

5. Other NoSQL Databases

Graph databases

A graph database is a database that uses graph structures with

- nodes,
- edges, and
- node/edge properties

to represent and store data.

Graph databases

- Nodes represent entities such as people, businesses, accounts, or any other item you might want to keep track of.
- Properties are relevant information that relate to nodes.
- Edges are the lines that connect nodes to nodes
- Most of the important information is often stored in the edges.
- Meaningful patterns emerge when one examines the connections and interconnections of nodes, properties, and edges.

Graph databases

- Compared with relational databases, graph databases are often faster for associative data sets
- They map more directly to the structure of object-oriented applications.
- As they depend less on a rigid schema, they are more suitable to manage ad hoc and changing data with evolving schemas.
- Graph databases are a powerful tool for graph-like queries.

Graph queries

- Reachability queries
- shortest path queries
- Pattern queries

Column Databases

- column databases store data column-wise,
  not row-wise
- also many relational data bases allow column-wise physical storage

Object Database

- An object database is a database management system in which information is represented in the form of objects as used in object-oriented programming.
- Most object databases also offer some kind of query language, allowing objects to be found using a declarative programming approach (OQL)
- Access to data can be faster because joins are often not needed.
- Many object databases offer support for versioning.
- They are specially suitable in applications with complex data.

Summary

- **NoSQL databases** denote several types of non-relational databases:
  - **key-value stores**: just store key/value pairs
  - **document databases**: store documents/(nested) dicts
  - **graph databases**: store and query information as a graph
- NoSQL databases are motivated by
  - **simplicity of design** and
  - **horizontal scaling**.
- NoSQL databases usually do not support ACID transactions, but **eventual concistency**
  - **BASE**: basically available, soft state, eventually consistent
  - **CAP theorem:** in presence of Network Partitions, a DBMS needs to choose between Consistency an Availability.

Summary (2/2)

- Document databases are **schemaless**
  - design decision between **foreign keys** and **embedded documents**
- Document databases are queried through **query documents**
  - key-value pairs
  - key-relation-value triples
  - logical aggregations (or)
- Document databases can partition document collections using a **partition/sharding function**.

Further Readings

- cap theorem: Brewer [2012]
- Mongo DB tutorial: MongoDB [2017]

References I

Eric Brewer. CAP Twelve years Later: How the. *Computer*, (2):23–29, 2012.

MongoDB. Mongodb tutorial — query documents, 2017. URL
`https://docs.mongodb.com/manual/tutorial/query-documents/`.