ISEngineering

Technische Universität Berlin

Enterprise Computing: Exercise 4 – Dynamo, GFS, BigTable

Marco Peise

ISEngineering
Wirtschaftsinformatik –
Information Systems Engineering

Agenda

Lecture MongoDB

Exercise 4

- GFS
- BigTable

ISEngineering
Wirtschaftsinformatik –
Information Systems Engineering

Technische Universität Berlin

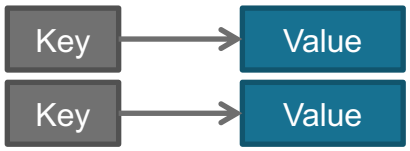
Lecture MongoDB

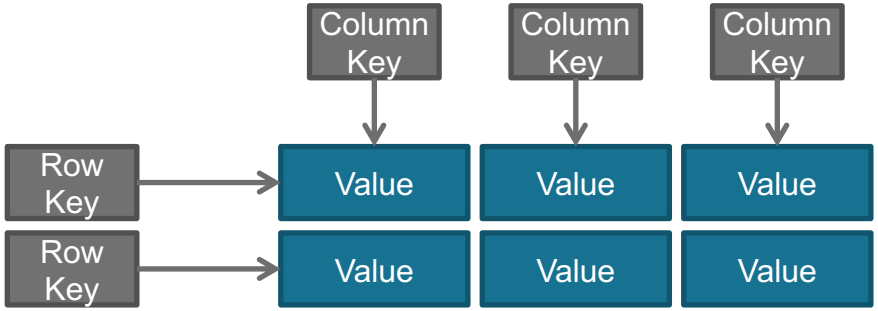
ISEngineering
Wirtschaftsinformatik –
Information Systems Engineering

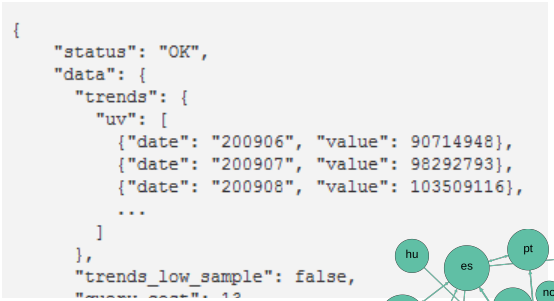
Examples of NoSQL Systems

Technische Universität Berlin

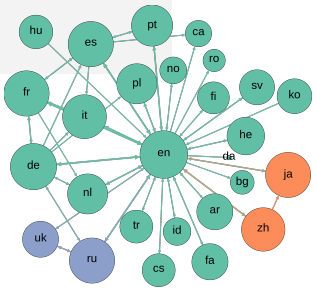
- Key-Value Stores
  - Riak
  - Redis
  - Voldemort
  - …
- Column Stores
  - Cassandra
  - HBase
  - …
- Document Stores
  - MongoDB
  - CouchDB
  - …
- Graph Stores
  - Neo4J







```
{
    "status": "OK",
    "data": {
      "trends": {
        "uv": [
          {"date": "200906", "value": 90714948},
          {"date": "200907", "value": 98292793},
          {"date": "200908", "value": 103509116},
          ...
        ]
      },
      "trends_low_sample": false,
```



ISEngineering
Wirtschaftsinformatik – Information Systems Engineering

Document Store Example: MongoDB
Learning objectives

Technische Universität Berlin

Cross-platform document-oriented database which supports ad hoc queries by field, range & regular expression.

High availability with replica sets.

Horizontal scaling through sharding.

Reference:
Kristina Chodorow
“MongoDB: The Definitive Guide, Second Edition”, O’Reilly, 2013

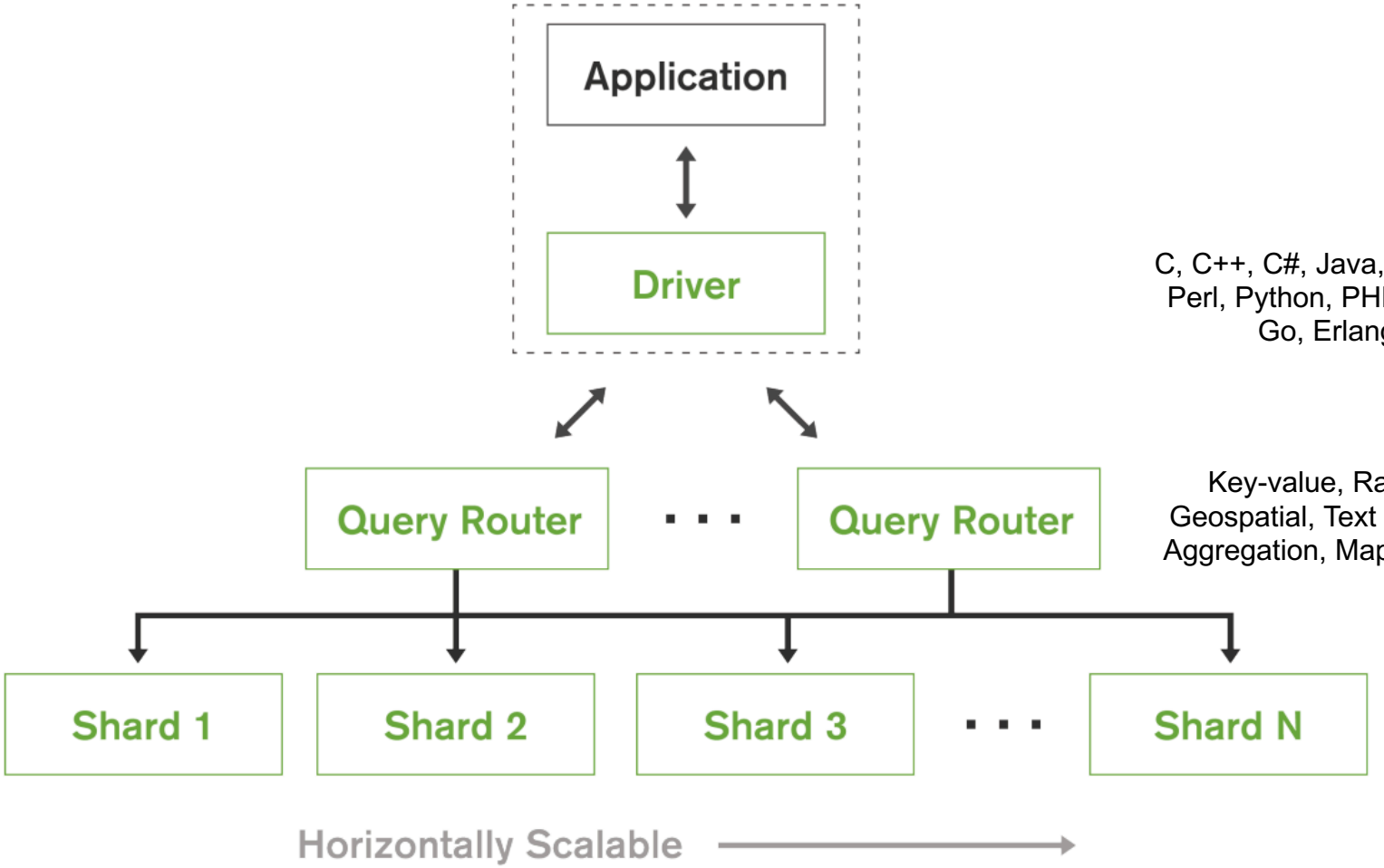
MongoDB

- written in C++ & developed in an open-source project (2007)
- supports:
  - Indexes & Secondary Indexes (unique, compound, array, ttl, geospatial, text ...)
  - Aggregation (count, distinct, group)
  - Special collection types
  - File storage
  - Deep query-ability
  - MapReduce
- does not support Joins & Transactions ($atomic flag)

best to use:

- deeply nested, complex data structures
- applications in javascript - stores in JSON-like format

MongoDB: Architecture



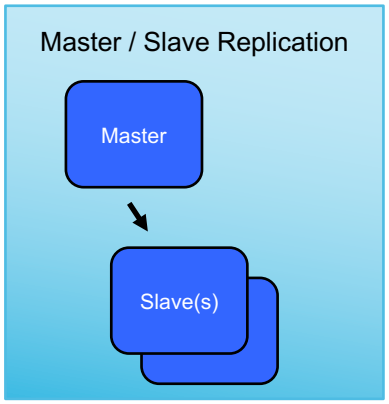
C, C++, C#, Java, Node.js, Perl, Python, PHP, Ruby, Go, Erlang

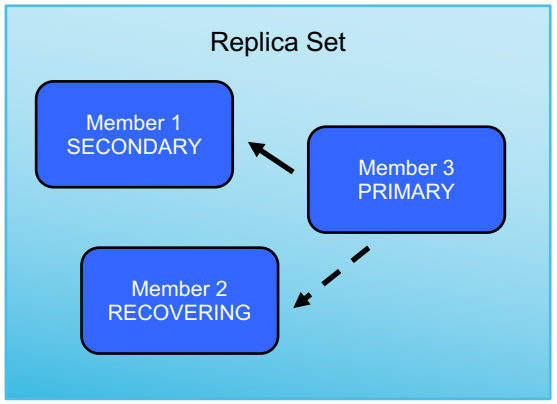
Key-value, Range, Geospatial, Text Search, Aggregation, MapReduce

Source: 2015 http://s3.amazonaws.com/info-mongodb-com/MongoDB_Architecture_Guide.pdf

MongoDB: Distribution Aspects

- Partitions called „Shard‘s“
  - Range-based
  - Hash-based
  - Location-aware





- asynchronous replication with primary server/node (+ hidden/delayed)
- failover causes elections

MongoDB

Technische Universität Berlin

Architecture:

- MongoDB is a schema-free document-oriented database system
- features master-slave replication with automated failover and built-in horizontal scaling via automated range-based partitioning
- version 3.0: WiredTiger storage engine provides optimistic concurrency control - only intent locks at global, db & collection level

Data Model & Programming:

- Normalized vs. Denormalized Data Models (Embedded vs. References)
- features secondary indexes, an expressive query language, atomic writes on a per-document level and fully-consistent reads
- client drivers for C, C++, C#, Java, Node.js, PHP, Python, Ruby etc.

Task 2 - Techniques used in Dynamo

Problem	Technique	Advantage
Partitioning	Consistent Hashing	Incremental Scalability
High Availability for writes	Vector clocks with reconciliation during reads	Version size is decoupled from update rates.
Handling temporary failures	Sloppy Quorum and hinted handoff	Provides high availability and durability guarantee when some of the replicas are not available.
Recovering from permanent failures	Anti-entropy using Merkle trees	Synchronizes divergent replicas in the background.
Membership and failure detection	Gossip-based membership protocol and failure detection.	Preserves symmetry and avoids having a centralized registry for storing membership and node liveness information.

Task 2 a) – MongoDB (Solution)

Partitioning

- Technique: Auto-sharding by shard key (range based partitioning or hash based partitioning)
- horizontal scaling (sharding)
  - Range based partitioning supports more efficient range queries. However, range based partitioning can result in an uneven distribution of data
  - Hash based partitioning ensures an even distribution of data at the expense of efficient range queries. Potentially lead to inefficient range queries
- location-aware sharding:
  - optimize physical location of documents for locating data in specific data centers, or for separating hot and cold data onto different tiers of storage

Task 2 b) – MongoDB (Solution)

Technische Universität Berlin

High Availability

- Technique: Production Cluster Architecture (Three Config Servers on separate machines, Two or More Replica Sets As Shards, One or More Query Routers (mongos))
- Advantages: A production cluster has no single point of failure.
- Disadvantages: configuration effort

Task 2 b) – MongoDB (Solution)

Handling temporary failures

- Technique: auto failover with replica sets - master/slave replication

  Possible failure scenarios for MongoDB deployments:
  - Application Servers or mongos Instances become unavailable
  - a Single mongo Becomes unavailable in a Shard
  - all Members of a Replica Set become unavailable
  - one or two Config Servers become unavailable
  - renaming Config Servers and Cluster availability
  - Shard Keys and Cluster availability

- Advantages: Provides high availability and durability guarantee when some of the replicas are not available.
- Disadvantages:

Task 2 b) – MongoDB (Solution)

Recovering from permanent failures

- Technique: depending where failure occurs
  config server: if you cannot recover the data on a config server, the cluster will be inoperable.
  shard: if a system is unrecoverable, replace it and create a new member of the replica set as soon as possible to replace the lost redundancy.
- Advantages: synchronizes divergent replicas in the background
- Disadvantages: highly resource-intensive process (CPU)

Task 2 b) – MongoDB (Solution)

Membership and failure detection

- Technique:

  - isMaster command
  - error from the driver if connection is dead
  - error if a query is send and response is „not master“
  - error if a server „steps down“ as primary because it can’t see a majority (closing its client connections)
  - Monitoring with Ops Manager and Cloud Manager to monitor database-specific metrics, including page faults, ops counters, queues, connections and replica set status

- Advantages: preserves symmetry and avoids having a centralized registry for storing membership and node liveness information

MongoDB

Pre Requisites:

- Introduction BSON
- requests: find(), findOne(), find().pretty(), find().sort()
- CRUD operations: insert, update, upsert, remove, save

Short Intro: BSON

element	::= "\x01" e_name double	64-bit binary floating point
	| "\x02" e_name string	UTF-8 string
	| "\x03" e_name document	Embedded document
	| "\x04" e_name document	Array
	| "\x05" e_name binary	Binary data
	| "\x06" e_name	Undefined (value) — *Deprecated*
	| "\x07" e_name (byte*12)	ObjectId
	| "\x08" e_name "\x00"	Boolean "false"
	| "\x08" e_name "\x01"	Boolean "true"
	| "\x09" e_name int64	UTC datetime
	| "\x0A" e_name	Null value
	| "\x0B" e_name cstring cstring	Regular expression - The first cstring is the regex pat stored in alphabetical order. Valid options are 'i' for c locale dependent, 's' for dotall mode ('.' matches ever
	| "\x0C" e_name string (byte*12)	DBPointer — *Deprecated*
	| "\x0D" e_name string	JavaScript code
	| "\x0E" e_name string	Deprecated
	| "\x0F" e_name code_w_s	JavaScript code w/ scope
	| "\x10" e_name int32	32-bit integer
	| "\x11" e_name int64	Timestamp
	| "\x12" e_name int64	64-bit integer
	| "\xFF" e_name	Min key
	| "\x7F" e_name	Max key

Short Intro: BSON

element ::=	"\x01" e_name double	64-bit binary floating point
|	"\x02" e_name string	UTF-8 string
|	"\x03" e_name document	Embedded document
|	"\x04" e_name document	Array
|	"\x05" e_name binary	Binary data
|	"\x06" e_name	Undefined (value) — *Deprecated*
|	"\x07" e_name (byte*12)	ObjectId
|	"\x08" e_name "\x00"	Boolean "false"
|	"\x08" e_name "\x01"	Boolean "true"
|	"\x09" e_name int64	UTC datetime
|	"\x0A" e_name	Null value
|	"\x0B" e_name cstring cstring	Regular expression - The first cstring is the regex pat stored in alphabetical order. Valid options are 'i' for c locale dependent, 's' for dotall mode ('.' matches ever
|	"\x0C" e_name string (byte*12)	DBPointer — *Deprecated*
|	"\x0D" e_name string	JavaScript code
|	"\x0E" e_name string	Deprecated
|	"\x0F" e_name code_w_s	JavaScript code w/ scope
|	"\x10" e_name int32	32-bit integer
|	"\x11" e_name int64	Timestamp
|	"\x12" e_name int64	64-bit integer
|	"\xFF" e_name	Min key
|	"\x7F" e_name	Max key

MongoDB query language

structured piece of data vs. a string

```
> db.accounts.find().pretty()
{
        "_id" : ObjectId("508465a1cb4cf4564b46a0b7"),
        "name" : "George",
        "favorites" : [
                "ice cream",
                "pretzels"
        ]
}
{
        "_id" : ObjectId("508465c6cb4cf4564b46a0b8"),
        "name" : "Howard",
        "favorites" : [
                "pretzels",
                "beer"
        ]
}
> db.accounts.find( { favorites : "pretzels" } );
```

MongoDB query language

What will the following query do?
db.scores.find({ score : { $gt : 50 }, score : { $lt : 60 } });

1. Find all documents with score between 50 and 60
2. Find all documents with score greater than 50
3. Find all documents with score less than 60
4. Explode like the Death Star
5. None of the above

Find all documents with score between 50 and 60:
db.scores.find({ score : { $gt : 50 , $lt : 60 } });

MongoDB query language

Technische Universität Berlin

Dot Notation

```
> db.ise.findOne();
{
    "_id" : ObjectId("562b5693e86f873b99671f4f"),
    "name" : "marco",
    "email" : {
            "work" :            "peise@tu-berlin.de",
            "personal" :        "peise@example.com"
    }
}
```

Question:

```
> db.ise.find( { email: { work : "peise@tu-berlin.de" } } );
```

MongoDB query language

Technische Universität Berlin

Dot Notation

```
> db.ise.findOne();
{
    "_id" : ObjectId("562b5693e86f873b99671f4f"),
    "name" : "marco",
    "email" : {
            "work" :            "peise@tu-berlin.de",
            "personal" :        "peise@example.com"
    }
}
```

Question:

```
> db.ise.find( { "email.work" : "peise@tu-berlin.de" } );
```

MongoDB query language

Technische Universität Berlin

Dot Notation

```
> db.ise.findOne();
{
    "_id" : ObjectId("562b5693e86f873b99671f4f"),
    "name" : "marco",
    "email" : {
            "work" :            "peise@tu-berlin.de",
            "personal" :        "peise@example.com"
    }
}
```

Question:

```
> db.ise.find( { "email.work" : "peise@tu-berlin.de" } );
{ "_id" : ObjectId("562b5693e86f873b99671f4f"), "name" : "marco", "email" : {
    "work" : "peise@tu-berlin.de", "personal" : "peise@example.com" } }
```

Technische Universität Berlin

Exercise 4

Task 1

a) Explain with your own words how distributed storage engines (like Dynamo) is handling temporary failures and give an example.

b) Give a reasons why one should use distributed caching and explain it in a use case.

c) Name three advantages of distributed caches and explain them in one sentence.

Task 2 - GFS

a) Which operations are supported by GFS? For which types of queries is GFS optimized?

b) State two main differences between Dynamo and GFS with regard to operation and data types?

c) Explain why GFS is particularly well suited as storage system for MapReduce job input and output data (why not Dynamo or a relational database?).

Task 3 - BigTable

a) Which operations are supported by GFS? For which types of queries is GFS optimized?

b) Explain why GFS is particularly well suited as storage system for MapReduce job input and output data (why not Dynamo or a relational database?).