Document Object Storage with MongoDB

Lecture BigData Analytics

Julian M. Kunkel

julian.kunkel@googlemail.com

University of Hamburg / German Climate Computing Center (DKRZ)

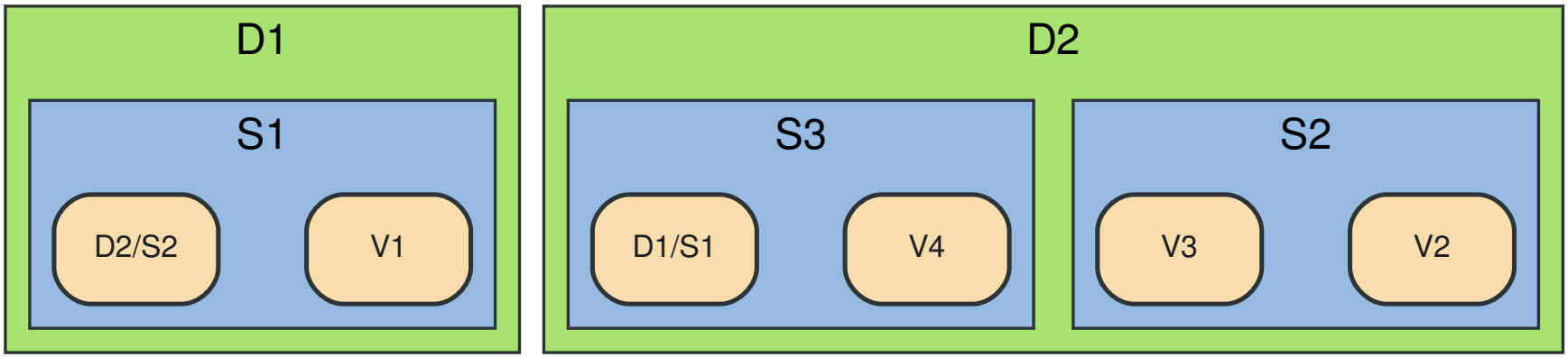
2017-12-15

Disclaimer: Big Data software is constantly updated, code samples may be outdated.

Outline

Data Model

- Documents contain semi-structured data (JSON, XML)
- Each document can contain data with other structures
- Addressing to lookup documents are implementation specific
  - e.g., bucket/document key, (sub) collections, hierarchical namespace
- References between documents are possible
- Example technology: **MongoDB**, Couchbase, DocumentDB



Source: Document Store. The Neo4j Manual v2.2.5 [33]

D=Document, S=Subdocument, V=Value, X/Y=reference to a subdocument in another document

MongoDB [11]

- Open-source document database
- High-performant and horizontally scalable for clusters
- Interfaces: interactive shell: mongo, REST, C, Python, ...
  - Connector for Hadoop for reading/writing to MongoDB

Data model

- Database: as usual, defines permissions
- Document: BSON object (binary JSON) – consisting of subdocuments
  - Primary key: _id field (manually set or automatically filled)

```
1  "_id" : ObjectId("43459bc2341bc14b1b41b124"),
2  "students" : [ # subdocument
3    { "name" : "Julian", "id" : 4711, "birth" : ISODate("2000-10-01")},
4    { "name" : "Hans",   "id" : 4712, "birth", ... }  ]
```

- Collection: like a table of documents
  - Addressing: collection name, document _id field (choose appropritely)
  - Documents can have individual schemas
  - Support for indexes on fields (and compound fields)
- Document references via object ids

Operations for the Data

- Documents: insert (create), find (read), update, delete (CRUD)
  - Sort, aggregate: use accumulators to aggregate fields
- Collections: create, drop removing of collections
  - Automatically created when first document is inserted
- Schemas via document validation
  - When creating a collection, a validator can be defined
  - It is checked upon insert / update
  - Triggers an action: warning or reject of changes

Semantics [14]

- The id field is always created, you can also define a (unique) id as string !
- Atomicity on document level: changes only one document at a time
  - All fields that must be updated together must be part of one doc
- Durability: flexibly; users can define a “write concern”
- Concurrency: read/write/exclusive locks are used internally
- Bulk operations are supported

Query Documents [13]

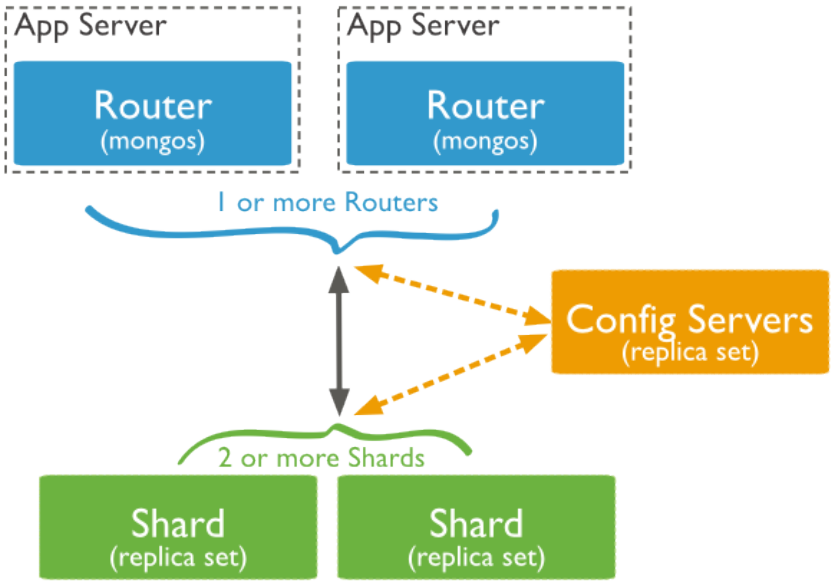
- Operations select the document to operate by defining documents
  - Example: lookup of a document using find(<query>, <projection>)
  - Query: defines the relevant documents (filtering)
  - Projection: The matched elements from JSON, e.g., return first array element
- Properties of a query (filter) document restrict the query:
  - Select all: {} (empty JSON)
  - Select documents with the key and value: { key : value }
  - Comparators: $eq, $lt, $ne (not equal)
  - Compare with sets: $in and $nin; { key : { $in: [value1, value2] } }
  - Logical: $and, $or, $not, $nor, $exists; { $or : [key : val, alt key, alt val] ; }
  - Text search: $text, $regex, $where (JavaScript expression)
  - Geospatial query operators: $geoWithin, $near, $minDistance, ...
- Subdocuments can be addressed using the dot notation
  - Example query document: { "students.age" : { $gt : 15} }

Architecture

- Shard: mongod server or *replica set* responsible for a set of data
- Replica set: server cluster impl. master-slave replication / failover

Components

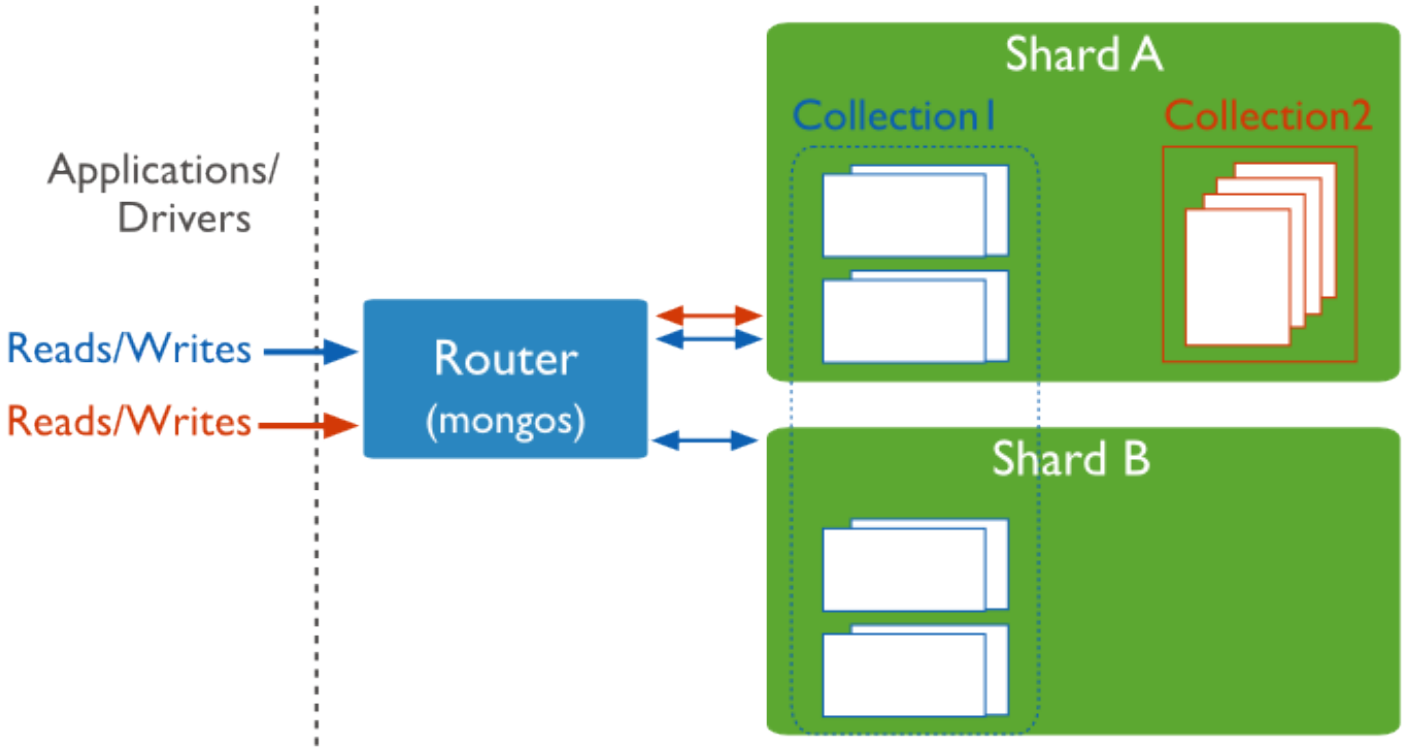
- Config server: replica set stores metadata and replication information
- Mongos: query router between client and replica set
- Mongod: MongoDB shard server providing storage space
- Balancer migrates data (chunks) between the servers



Source: [14]

Accessing Sharded Data [14]

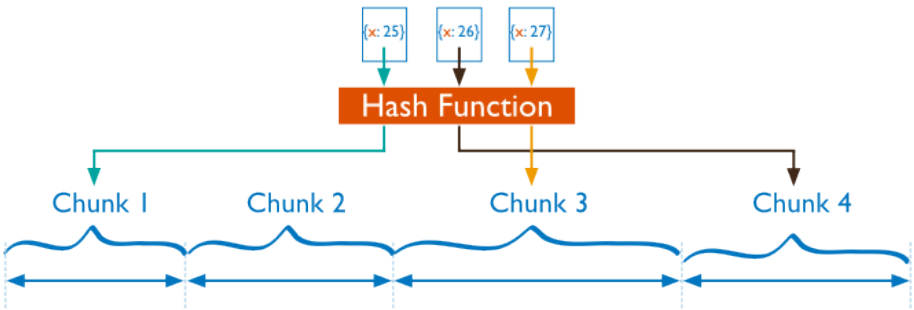
- Sharding (and options) are set on the collection level



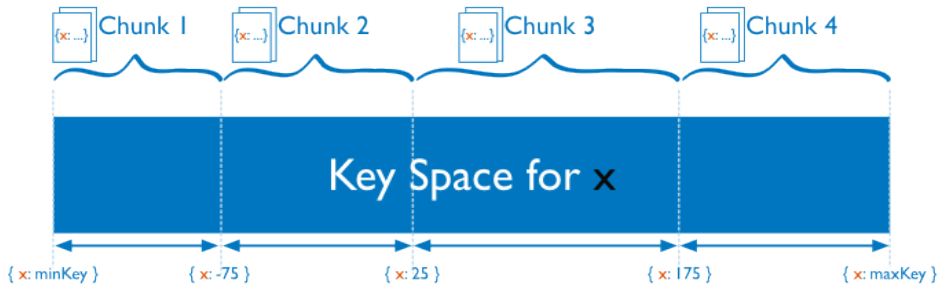
Source: [14]

Partitioning of Data (for one Collection) [14]

- Shard key: immutable field(s) in every collection document
  - Either by hashing of fields or by distributing ranges
  - Performance relevant: select an appropriate shard key
- Chunk: contiguous range of shard key values
  - Chunks are automatically splitted and migrated between shards



Hash sharding; Source: [14]

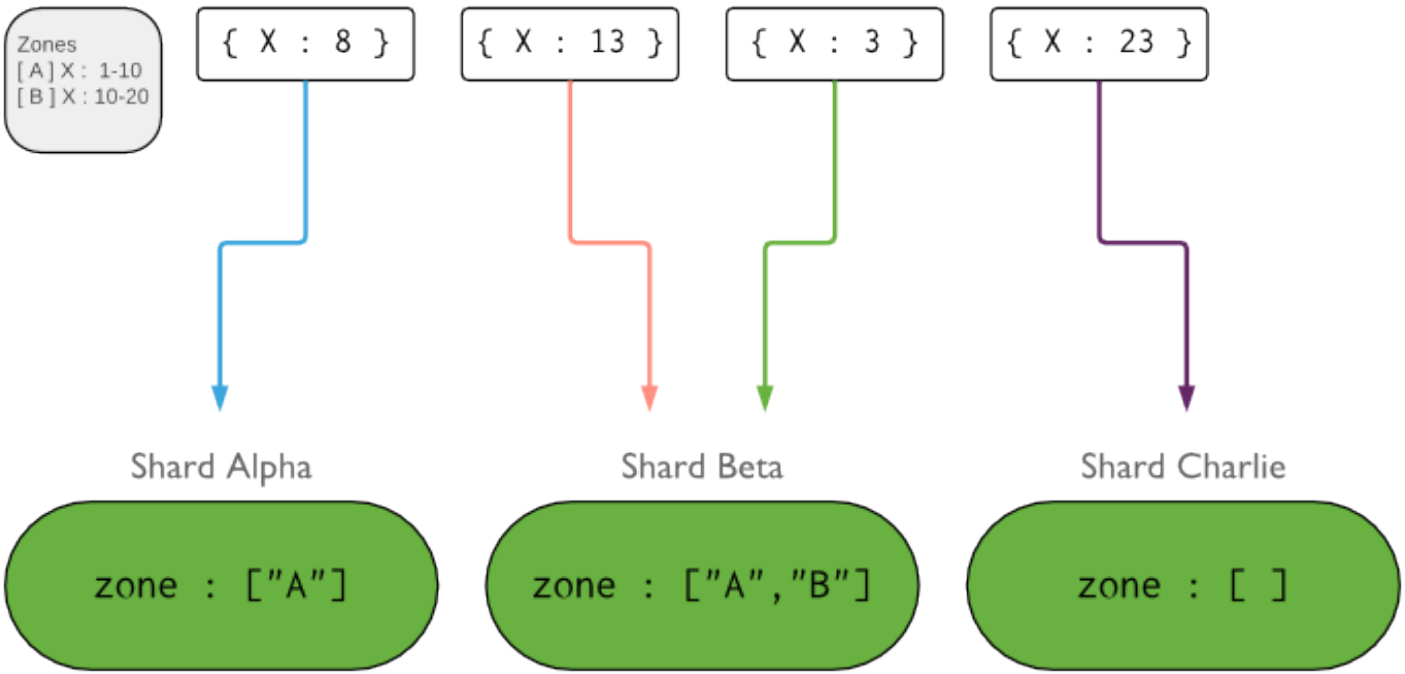


Ranged sharding; Source: [14]

- Internal processing of queries
  - Broadcast necessary if the query filter does not contain the shard key
  - If shard key is part of the query, only the subset of servers is contacted

Zones [14,18]

- Goals: improve locality of data, distribute data across data centers
- Zone: groups documents based on the value of shard key
- Create a tag for shards matching a key range
- A shard (server/replica set) may be assigned to multiple zones
- Migration of chunks is done only within its origin zone



Source: [14]

MongoDB Shell [12]

Start by invoking: `mongo`

Commands

- <X>.help: show help for obj X
- show collections: print the existing collections
- db.<COL>: access the collection COL
- Collection operations
  - find(query): search for an document with properties according to doc
  - insert(query): insert
  - update(query, update): update
  - remove(query): delete all matching documents
  - drop(): remove the collection discarding all data
  - createIndex(doc): create an index for all listed fields
  - sort(doc): sort documents based on the keys in the doc
  - aggregate(doc): use accumulators
  - explain(): describe the operations to perform

Examples

```
# Bulk insert some values into the collection uni (to be created)
var bulk = db.uni.initializeUnorderedBulkOp();
bulk.insert({"_id": "4711", "name": "Julian", "gender": "male", "major": "computer science", "birth": ISODate("2000-10-01")})
bulk.insert({"_id": "4712", "name": "Hans", "gender": "male", "major": "computer science", "birth": ISODate("2000-10-01")})
bulk.execute()
# BulkWriteResult({ "writeErrors" : [ ],  "writeConcernErrors" : [ ], "nInserted" : 2, "nUpserted" : 0,  "nMatched" : 0,
       ↪ "nModified" : 0, "nRemoved" : 0,"upserted" : [ ] })

# Create an index on the student's name
db.uni.createIndex( { "name": 1 } )

# Return the first 10 student names
db.uni.find( {}, {"name" : 1} ).limit(10)
#{ "_id" : "4711", "name" : "Julian" }
#{ "_id" : "4712", "name" : "Hans" }

# Return the student birth data where the name matches Hans
db.uni.find( { "name" : "Hans" }, {"birth" : 1} )
# { "_id" : "4712", "birth" : ISODate("2000-10-01T00:00:00Z") }

# Update the student, adding an address to all students with name Julian
db.uni.update ( {"name" : "Julian" }, {$set : { "address" : { "plz" : 4711, "city" : "Hamburg" } } }, {multi: true} )
# WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

# Aggregate to count the number of male and female computer science students
# The match stage filters the documents first
# The _id field indicates the field to use for grouping, here gender
db.uni.aggregate( [ { $match: { "major": "computer science"} },
                    { $group: { "_id": "$gender", "count": { $sum: 1 } } }   ] )
# Returns: { "_id" : "male", "count" : 2 }

db.uni.drop() # remove collection
```

Python [17]

```
1  import pymongo
2  from bson.objectid import ObjectId # internal object IDs
3
4  # Establish a connection
5  client = pymongo.MongoClient('localhost', 27017)
6  db = client.test # access test database
7
8  # print collections
9  db.collection_names(include_system_collections=False)
10 # ['uni']
11 uni = db.uni # access uni collection
12
13 print(uni.find_one({"name": "Julian"}))
14 # {'_id': '4711', 'name': 'Julian', 'gender': 'male', 'birth': datetime.datetime(2000,
       ↪ 10, 1, 0, 0), 'major': 'computer science'}
15
16 # Insert a student, we don't care about the id here
17 print(uni.insert_one({"name" : "Fritz"}).inserted_id)
18 # 58495ad0e91ebf67ae7f197d
19
20 # We can also use strings as the ID...
21 print(uni.insert_one({"_id": "Fritz", "name" : "Fritz"}).inserted_id)
```

Summary

- The document object model stores documents with subdocuments
  - Relations by embedding data as subdocument OR object reference
- MongoDB is a document object storage for JSON-like data
- Query filtering (SQL where clause) via JSON documents
- Scalable on a cluster via sharding of documents

Bibliography

11 https://docs.mongodb.com/getting-started/shell/introduction/

12 https://docs.mongodb.com/manual/reference/mongo-shell/

13 https://docs.mongodb.com/manual/tutorial/query-documents/

14 https://docs.mongodb.com/manual/sharding/

15 https://docs.mongodb.com/manual/core/data-model-operations/

16 SQL vs. MongoDB: https://docs.mongodb.com/v3.0/reference/sql-comparison/

17 http://api.mongodb.com/python/current/tutorial.html

18 https://docs.mongodb.com/manual/core/zone-sharding

33 The Neo4j Manual v2.2.5. http://neo4j.com/docs/stable/