MongoDB: document-oriented database

Software Languages Team
University of Koblenz-Landau
Ralf Lämmel, Sebastian Jackel and Andrei Varanovich

Motivation

- Need for a flexible schema
- High availability
- Scalability

Use cases

- Event logging
- Document and Content Management Systems
- Analytics, e.g. financial statistics, data mining
- Handling of geospatial data, e.g. mobile applications, gaming

Key concepts

- Document: Basic unit of data in MongoDB (think RDBMS row but more flexible)
- Collection: A collection of documents
- Database: Houses collections
- Many databases per MongoDB instance
- JavaScript shell: Useful for administration & data manipulation
- MapReduce rather than joins

Document

```
{ "_id" : ObjectId("4fd1ef6716fc9783d9e779f0"),
"name" : "Ralf",
"address" : { "city" : "Koblenz", "country" : "Germany" },
"dept" : ObjectId("4fd1ea2942afa58847224864"),
"salary" : 1234 }
```

- Data encoded as BSON document
  - Binary encoded JSON variant
  - JavaScript data types + some useful additions
  - Documents can be nested

MongoDB Shell

```
$ mongo localhost/mydb
MongoDB shell version: 2.2.0
connecting to: localhost/mydb
>
```

- JavaScript shell for direct interaction with a database
  - Allows maintenance and manipulation of DB and clusters
  - Can be used to execute scripts

Basic operation: create

```
> db.employees.insert(
{
   "name" : "Erik",
   "salary" : 12345,
   "address" : {
      "city" : "Utrecht",
      "country" : "Netherlands"
   }
})
```

Basic operation: read

```
> db.employees.find()

{ "_id" : ObjectId("4fd2052816fc9783d9e779f5"), "name" : "Klaus", "salary" :
23456, "dept" : ObjectId("4fd1f7b616fc9783d9e779f3"), "address" : { "city" :
"Boston", "country" : "USA" } }
{ "_id" : ObjectId("4fd2073116fc9783d9e779f7"), "name" : "Karl", "salary" :
2345, "dept" : ObjectId("4fd2061816fc9783d9e779f6"), "address" : { "city" :
"Riga", "country" : "Latvia" } }
...
>
```

- _id field is automatically created
  - Can be manually overridden

Basic operation: update

```
> db.employees.update({“name“ : “Erik“},
  {$set : {"dept" : ObjectId("4fd1ea2942afa58847224864")}})
```

Set the value of an object property or add a new one

```
> db.employees.update({“name“ : “Ralf“},
  {$inc : {"salary" : 1000}})
```

Basic operation: delete

```
> db.employees.remove({“name“ : “Joe“})
```

Removes all where „name“ equals „Joe“

```
> db.employees.remove({})
```

Removes all objects in the collection

Querying

```
> db.employees.find({"salary" : {$gt : 200000}})

{ "_id" : ObjectId("4fd2024f16fc9783d9e779f4"), "name" : "Ray",
"salary" : 234567, "dept" : ObjectId("4fd1f78316fc9783d9e779f2"),
"address" : { "city" : "Redmond", "country" : "USA" } }
```

- Several filters are available
  - `$lt, $gt, $eq, $neq` - Arithmetic comparison
  - `$all, $in` - existential queries
  - `$and, $or` - boolean operators
  - ... plus a lot of special operators

MapReduce: Map

```
> var mapfunc = function() {
      emit("salary", this.salary);
  }
```

- Map function:
  - No parameter
  - Treated as method of each object instead

MapReduce: Reduce

```
> var reducefunc = function(key, values) {
      var result = 0;
      values.forEach(function(value) {
          result += value;
      });
      return result;
  }
```

- Reduce function:
  - Parameters: Key and Array of corresponding values
  - Returns one result

MapReduce: Execution

```
> db.employees.mapReduce(mapfunc, reducefunc,
    {"out" : {"inline" : 1}})
{
    "results" : [
        {
            "_id" : "salary",
            "value" : 399747
        }
    ],
    "timeMillis" : 50,
    "counts" : {
        "input" : 7,
        "emit" : 7,
        "reduce" : 1,
        "output" : 1
    },
    "ok" : 1,
}
```

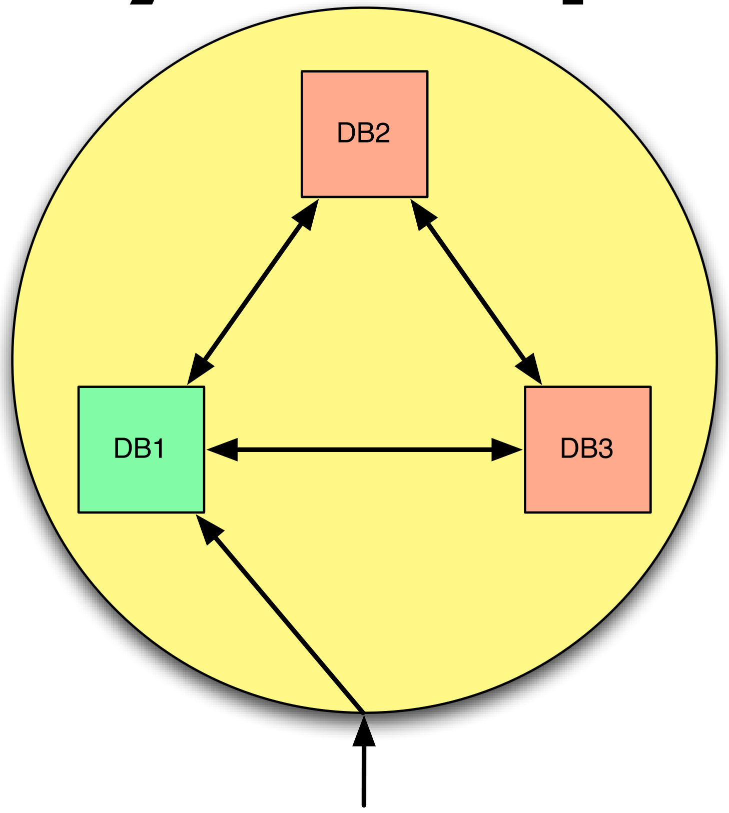
Output targets

- MapReduce results can be output in several ways
  - `{“replace“ : “collName“}` - replace an existing collection with results (default)
  - `{“merge“ :  “collName“}` - merge with collection. Existing keys will be overwritten
  - `{“reduce“ :  “collName“}` - recuce with content of existing collection
  - `{“inline“ : 1}` - don‘t create a collection, MapReduce will happen in RAM (use with caution!)

DEMO

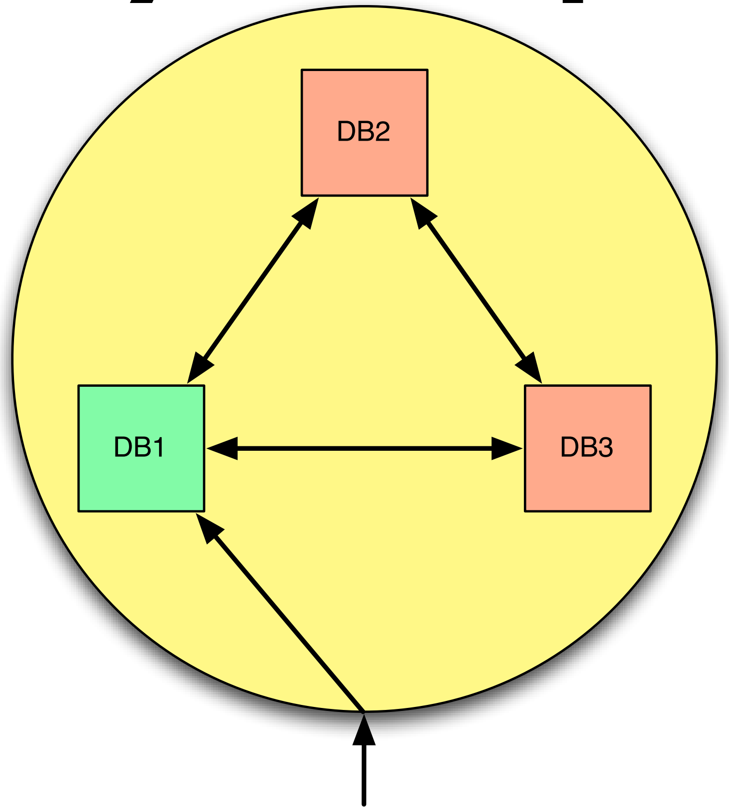
101companies:mongodb

Consistency & Availability: Replica Sets



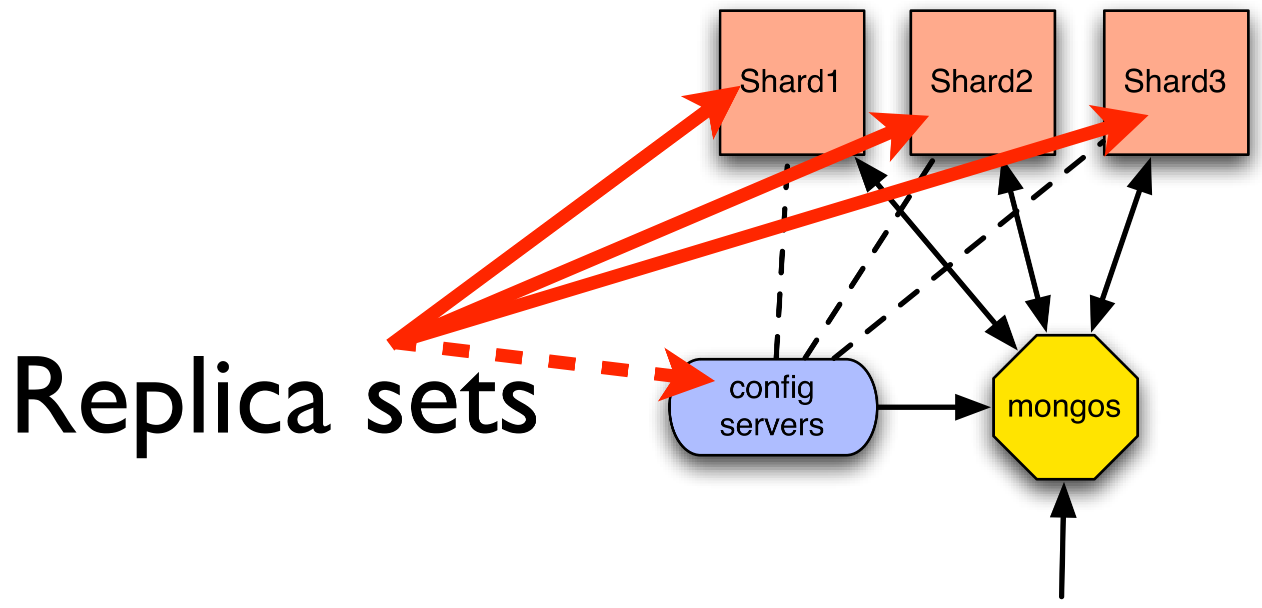
- Provide automated failover
- Distribute (read) load
- Disaster Recovery
- NOTE: Eventually consistent behaviour!

Consistency & Availability: Replica Sets



- 2 or more nodes which mirror each other
- One node is elected as PRIMARY
- PRIMARY will coordinate all reads/writes
- PRIMARY crashed? Elect new PRIMARY!

Scalability: Sharding



- Distribution of data
- Automatic load balancing
- Scaling out

Summary

You learned about ...

- Basic concepts of a document oriented DB
- MongoDB CRUD operations
- Safety features of MongoDB
- MongoDB Scalability

Resources

- Official MongoDB documentation:

  http://www.mongodb.org/display/DOCS/Introduction

- MongoDB on Java by Brendan McAdams (@rit) at MongoNYC 2010

  http://blip.tv/mongodb/java-development-with-mongodb-3720353

- Scaling MongoDB by Brendan McAdams at MongoUK 2011

  http://www.10gen.com/presentations/mongouk-2011/scaling-mongodb