MongoDB Schema Design

Demystifying document structures in MongoDB

Jon Tobin

@jontobs

MongoDB Overview

- NoSQL Document Oriented DB
- “Dynamic Schema”
- HA/Sharding Built In
  - Simple async replication setup
  - Automated elections
  - Sharding engine/router/balancer
- Aggregation Pipeline
- Map-Reduce
- “Developers Database”
  - Full driver library
  - Work outside of shell
- Easy to use
  - Read:“Easy to get started”

Terms: What Do They Mean?

MySQL	MongoDB
Database	Database
Table	Collection
Row	Document
Field	Key : value pairs

Practical Examples

Modeling Data - SQL

Tbl_Student

Student ID	First Name	Middle Name	Last Name
100	Jonathan	Eli	Tobin
101	Meathead	Rob	Lowe

@ each intersection is a single scalar value

Tbl_Grades

Student ID	Course ID	Grade
100	PHY101	B
101	PHY101	F
100	BUS101	B+

Tbl_Classes

Course ID	Course Name	Credits
PHY101	Physics 101	3
BUS101	Business 101	3

Modeling Data - SQL

Good

- Normalization gives guidelines
- Minimizes redundancy
- Efficient updates
- JOIN to get data (in database)
- Database is feature rich
- Schema enforces data intergrity

TLDR: great for consistency, updates & application simplicity

Bad

- Three queries for data
- Pre-defined schema constrains agility
- Complex relationships

TLDR: querying and inserting can (will) be inefficient, features may affect performance

WHY: It's all about (co)location of relevant data

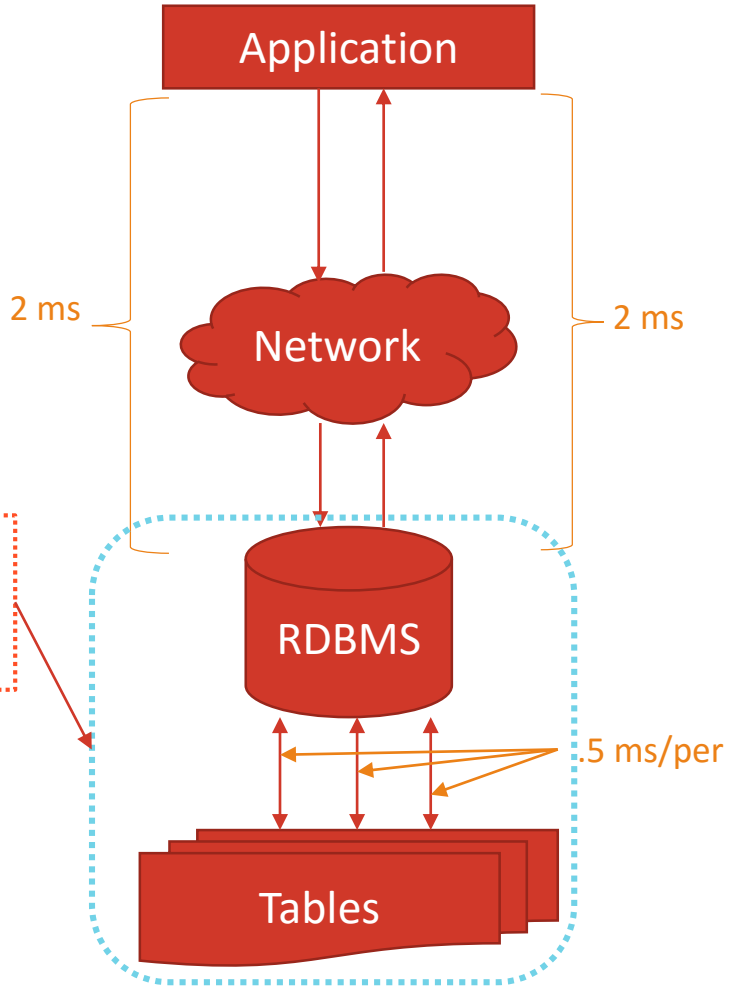
WHERE: At what level should feature be implemented for best performance

RDBMS JOINS

Assumptions

Network latency: 2 ms

Single table op: .5 ms



OPERATION

3 table JOIN operation

JOINS are handled by the DB

Time to App Response

2 + (3 * .5) + 2 =

5.5 ms

PERCONA

MongoDB Design Basics

- Known as a “Developers DB”
  - Meaning: “put it in the app!”
- “No” Joins
  - Joins are done in application
  - V3.2 = `$LOOKUP` =left outer join (no sharding)
- Dynamic Schemas
  - Fields (keys) can be added anytime
  - Keys don’t need to be added to all docs (rows)
  - Keys can have
    - Multiple values (arrays)
    - Multiple key:value pairs (sub-docs)
- (De)Normalization is up to you
  - What best fits your application
  - Could be a mix
- 16MB BSON limit on docs
- Atomicity within a single document
  - NO multi-doc transactions

JSON Types

- Number
- Text
- Boolean
- Array
- Object
- Null

Modeling Data - MongoDB

```
{
        "_id" : ObjectId("507f1f77bcf86cd799439011"),
        "studentID" : 100,
        "firstName" : "Jonathan",
        "middleName" : "Eli",
        "lastName" : "Tobin",
        "classes" : [
                {
                        "courseID" : "PHY101",
                        "grade" : "B",
                        "courseName" : "Physics 101",
                        "credits" : 3
                },
                {
                        "courseID"  :  "BUS101",
                        "grade"  :  "B+",
                        "courseName"  :  "Business 101",
                        "credits"  :  3
                }
        ]
```


PERCONA

QnD Doc Design Pointers

Embed

- Query performance priority
- Fields are fairly static
- Size of doc can be reasonably determined
- Eventual consistency acceptable

```
{
        "_id" : ObjectId("53d98f1…")
        "firstName" : "Jonathan",
        "lastName" : "Tobin",
        "year" : 3,
        "classes" : [
           {
               "class" : "Calc 101",
               "credits" : 3,
           },
           {
                       -etc-
}
```

Reference

- Insert performance priority
- Updates are common
- Immediate consistency necessary
- Field size can't be determined

```
{
        "_id" : ObjectId("53d98f1…")
        "firstName" : "Jonathan",
        "lastName" : "Tobin",
        "year" : 3,
        "classes" : [
           ObjectID(<of_class_1>),
           ObjectID(<of_class_2>),
           ObjectID(<of_class_3>),
           ]
}
```


PERCONA

Embedding

- Insert
  - Quick
  - Semi efficient
- Update
  - studentID
    - Quick
  - courseID
    - Complex
    - Inefficient
    - Inconsistent
- Query
  - Fast
  - Efficient

Be mindful: cache thrashing

```
Show collections
          college.students


//sample document
{
"_id" : ObjectId("507f1f77bcf86cd799439011"),
"studentID" : 100,
"firstName" : "Jonathan",
"middleName" : "Eli",
"lastName" : "Tobin",
"classes" : [
          {
          "courseID" : "PHY101",
          "grade" : "B",
          "courseName" : "Physics 101",
          "credits" : 3                    },
          {
          "courseID"  :  "BUS101",
          "grade"  :  "B+",
          "courseName"  :  "Business 101",
          "credits"  :  3
          }
]
}
```


PERCONA

Referencing

- Insert
  - Quick
  - Efficient
- Update
  - classes
    - Fairly quick
  - courseID
    - Efficient
    - Consistent
- Query
  - Fast
  - Efficient

Be mindful: join overhead

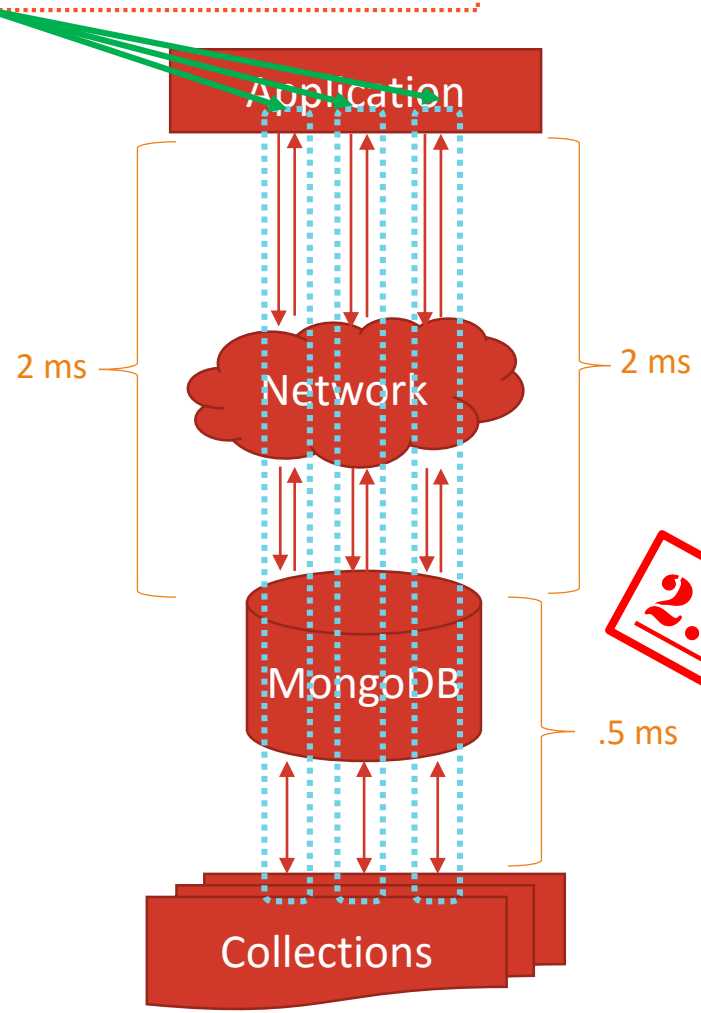
```
Show collections
        college.students
        college.courses
        college.grades


//sample document
{
"_id" : ObjectId("507f1f77bcf86cd799439011")
"firstName" : "Jonathan",
"lastName" : "Tobin",
"year" : 3,
"classes" : [
        ObjectID(<of_class_1>),
        ObjectID(<of_class_2>),
        ObjectID(<of_class_3>),
        ]
}
```

MongoDB Design - QnD

- DEPENDS: on use case
  - Embed
    - Efficient lookups
    - Infrequently changed data
    - Often queried data
    - Atomicity
  - Reference
    - Efficient writes
    - Oft excluded data (from queries)
    - Boundless additions
    - Doc size may approach 16MB limit

MongoDB “JOINs”

Each query is separate full stack operation



OPERATION
3 coll JOIN operation

Assumptions
Network latency: 2 ms
Single table op: .5 ms

Time to App Response
(2 + 2 + .5) *3 =
13.5 ms

2.5X SLOWER !!

PERCONA

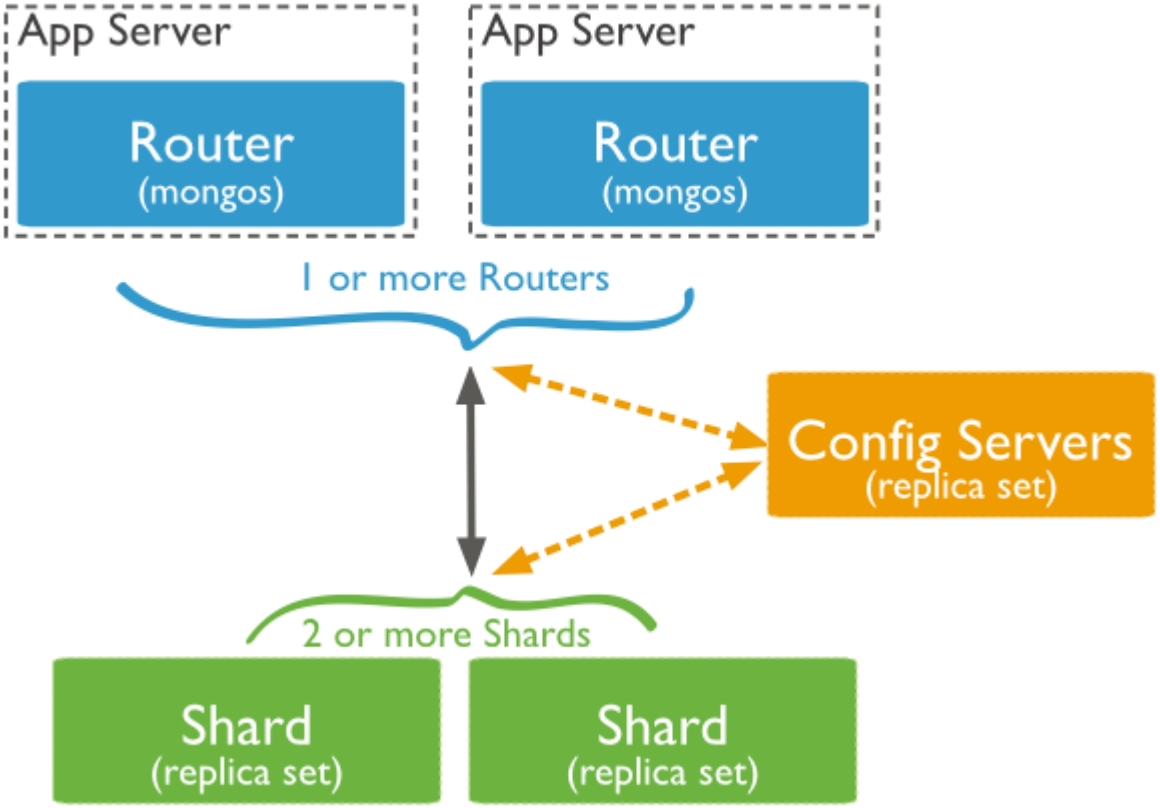
Finding Middle Ground

- Embed fields that are often fetched
  - If they don't grow boundlessly
- Limit growing keys to 1/per doc
  - Move to last key
- Reference fields that are volatile
  - Or are occasionally queried
- Atomicity can be achieved @ single doc level
  - Take care in design
- Index judiciously
  - Re-evaluate often
- Store relevant data
  - Archive old data (when possible)
  - Or delete
- **DON'T default to the RDBMS way**

Sharding

…an unfortunate name

Sharded Cluster



PERCONA

Sharding

- Mongo distributes data based on shard key
  - Indexed single key
  - Indexed compound key
- Data “chunked” by key space
  - Range based
  - Hash based
- **Shard key is immutable**
- Balancing happens in background
  - Inside each shard
  - Between shards

Two distinct possibilities:

- Range: data has low entropy (scatter) > *key1 & key2 are likely to be together*
- Hash: data has high entropy (scatter) > *key1 & key2 are unlikely to be together*

Shard Keys

- For insert speed: (avoid single shard bottleneck)
  - High-entropy shard key (mostly random).
  - Balances load across all shards.
  - Avoid migrations, can be expensive in MongoDB.
  - Range queries are scatter-gathers.
  - “Scatter” is **good**.
- For query speed: (avoid “scatter gather” queries)
  - Low-entropy shard key (mostly sequential).
  - Range queries should only hit 1 shard.
    - Queries should include shard key.
  - Indexing is still necessary
  - “Scatter” is **bad**.
- Data loading
  - Shard the collection(s)
  - Pre-split and distribute chucks
  - Removes balancer bottleneck

@ MongoDB World?

When: Thursday 6/30 8:00 PM

Where: Park Central Hotel

Why: Percona believes in "community." Without the entire community we all lose.

More Details & Reg: http://bit.ly/PerconaOH

PERCONA

Percona Live Amsterdam

- **Share Your Knowledge**
- **Learn from others**
- **Network with The Community**
- **Drink (free) Beer!!!**

Call for papers is open!!

https://www.percona.com/live/plam16/

PERCONA
LIVE EUROPE
AMSTERDAM

PERCONA

Useful Resources

Free Resources

- Understanding How Your MongoDB Schemas Affect Scaling – David Murphy
- MongoDB Administration for MySQL DBA – Alexander Rubin
- Optimizing MongoDB for High Performance Applications – David Murphy
- MongoDB University – FREE ONLINE TRAINING!

Books

- *MongoDB: The Definitive Guide* by Kristina Chodorow
  - Outdated, but still largely relevant as far as design goes
- *MongoDB Applied Design Patterns* by Rick Copeland
  - More up to date than “the definitive guide” in regards to functionality but is already dated in terms of storage engine

DATABASE PERFORMANCE MATTERS