MongoDB Patterns, Pitfalls, & Best Practices
Michael Carducci
@MichaelCarducci
3 reasons to use MongoDB

Note: This is precursor post to my talk at Mongo Boston on September 20th. It’s gonna be at Mi
COMpletely Awesome). If you haven’t signed up yet, stop being lame. It’s gonna be awesom
http://www.10gen.com/conferences/mongoboston2010

People have asked me why to use MongoDB. I used to answer with “it’s SO FREAKING
how “new” and “cool” and “hawt” it felt. I would wax on about how it felt like the first ti
That revolution it.

DIVIDED COMMUNITY
MongoDB is evil. It...

... loses data (sources: 1, 2)

... in fact, for a long time, ignored errors by default and assumed every single write

Why you should never, ever, ever use MongoDB

19 Jul 2015

joepie91's Ramblings

RSS
DIVIDED COMMUNITY
DIVIDED COMMUNITY:

F*** MongoDB, F*** Node.js, and F**

I hate MongoDB. I hate Node.js. I hate everything about NoSQL. While I'm hate Apple OS and Windows and Linux.

by Allen Coin  ·  Oct. 01, 12  ·  Database Zone  ·  Not set
MongoDB Pitfalls

USING THE RIGHT TOOL FOR THE JOB
WHEN TO USE MONGODB

- If your data structures are fluid or dynamic
- Storing hierarchical data
- Caching/presenting data
- Storing Documents
- High write volume that can tolerate loss
WHEN **NOT** TO USE MONGODB

- If your data is relational
- If durability and consistency are important
- If you need transactions
WHEN TO USE CAUTION

- Using either solution without a clear understanding of the problems you’re trying to solve
MONGO CAN BE PART OF A LARGER ARCHITECTURE
Patterns and Best Practices

DOCUMENT DESIGN
WHAT IS A DOCUMENT

{  
    _id: "123",
    title: "MongoDB: The Definitive Guide",
    authors: [
        { _id: "kchodorow", name: "Kristina Chodorow" },
        { _id: "mdirolf", name: "Mike Dirolf" }  
    ],
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English",
    thumbnail: BinData(0,"AREhMQ=="),
    publisher: {
        name: "O'Reilly Media",
        founded: 1980,
        locations: ["CA", "NY" ]
    }
}
MONGODB IS SCHEMALESS! WHO NEEDS DATA DESIGN?

“Your scientists were so preoccupied with whether they could that they didn't stop to think if they should.”

-Dr. Ian Malcolm (Jurassic Park)
GOOD DATA DESIGN GOES BEYOND THE DB

- Clear Structures
- Data structure harmonized with code
- Lessons Learned from RDBMS Databases
GOOD ARCHITECTURE IS FORWARD LOOKING

- Unknown future requirements
- Ability to scale economically
- Shorter Solution Lifecycles
MONGODB DOCUMENTS ARE FLEXIBLE

- TIMTOWTDI
- Document Design decisions involve trade-offs
DOCUMENT DESIGN BY EXAMPLE

Talent Agency Database
TALENT AGENCY

- Performers
- Clients
- Venues
- Bookings
PERFORMER COLLECTION

> db.performers.find({_id: 'Shawn'}).pretty()
{
   "_id": "Shawn",
   "FirstName": "Shawn",
   "LastName": "Popp",
   "Website": "http://openthetrapdoor.com",
   "PerformanceStyles": [
      "CloseUp magic",
      "Stage Magic",
      "Comedy Magic",
      "Mentalism"
   ]
}

> db.addresses.find({_id: 'Shawn'}).pretty()
{
   "id": "Shawn",
   "Street": "123 Magic Street",
   "City": "Centennial",
   "State": "CO",
   "Zip": 80123
}
PERFORMER + ADDRESS - THE MONGODB WAY

```javascript
> db.performers.find({ _id: 'Shawn' }).pretty()
{
  "_id": "Shawn",
  "firstName": "Shawn",
  "lastName": "Popp",
  "website": "http://openthetrapdoor.com",
  "performanceStyles": [
    "CloseUp magic",
    "Stage Magic",
    "Comedy Magic",
    "Mentalism"
  ],
  "address": {
    "street": "123 Magic Street",
    "city": "Centennial",
    "state": "CO",
    "zip": 80123
  }
}
```
TAKE WHAT YOU NEED - PROJECTION

```javascript
> db.performers.find({_id:'Shawn'}, {_id:0, 'firstName':1, 'lastName':1, 'address.city':1}).pretty();
{
    "firstName" : "Shawn",
    "lastName" : "Popp",
    "address" : {
        "city" : "Centennial"
    }
}
```
SUBSTRUCTURES WORK WELL WITH CODE

```javascript
let addr = db.performers.find({_id: 'Shawn'}, {'_id': 0, 'address': 1}).pretty();

// Pass the whole Map to this function:
doSomethingWithOneAddress(addr);

// Somewhere in the code this function is defined
doSomethingWithOneAddress(Map addr){
  // Look For State
  .
  .}
```
WHY DOES THIS MATTER?
DOCUMENT SHAPES CAN VARY
SUBSTRUCTURES WORK WELL WITH CODE

```javascript
> addr = db.performers.find({_id:'Michael'}, {_id:1, 'address':1}).pretty();
{
  "_id" : "Michael",
  "address" : {
    "street" : "3537 Stronghold Way",
    "city" : "Parker",
    "state" : "CO",
    "country" : "USA"
  }
}

// Pass the whole Map to this function:
doSomethingWithOneAddress(addr);

// Somewhere in the code this function is defined
doSomethingWithOneAddress(Map addr){
  // Look For State Or Optional Country
  ...
  
}
MODELING RELATIONSHIPS IN MONGODB
ONE-TO-ONE RELATIONSHIPS

- Belongs-to relationships are typically embedded
- Holistic view of the entity with embedded relationships and sub-structures
- Great Read Performance
- Keeps things simple
- Frees up time to tackle harder schema issues
ONE-TO-N BELONGS
MULTIPLE ADDRESSES

```javascript
> db.performers.find({_id:'Michael'}).pretty();
{
  "_id" : "Michael",
  "firstName" : "Michael",
  "lastName" : "Carducci",
  "website" : "http://trulymagic.com",
  "performanceStyles" : [
    "CloseUp magic",
    "Stage Magic",
    "Mentalism"
  ],
  "addresses" : [
    {
      "street" : "3537 Stronghold Way",
      "city" : "Parker",
      "state" : "CO",
      "country" : "USA"
    },
    {
      "street" : "123 S. Parker Rd STE 314",
      "city" : "Parker",
      "state" : "CO",
      "country" : "USA"
    }
  ]
}
```
MULTIPLE ADDRESSES

```javascript
> db.performers.find({_id:'Shawn'}).pretty()
{
   "_id" : "Shawn",
   "firstName" : "Shawn",
   "lastName" : "Popp",
   "website" : "http://openthetrapdoor.com",
   "performanceStyles" : [ 
      "CloseUp magic",
      "Stage Magic",
      "Comedy Magic",
      "Mentalism"
   ],
   "address" : {
      "street" : "123 Magic Street",
      "city" : "Centennial",
      "state" : "CO",
      "zip" : 80123
   }
}
```
MIGRATION OPTIONS

- Leave it alone
  - Write code that knows how to handle both address and addresses
- Migrate all documents when Schema Changes
- Migrate On Demand
  - As a record is loaded, make the change
  - Inactive performers never get updated
ONE-TO-MANY USING EMBEDDING
ONE-TO-MANY USING EMBEDDING

```javascript
> db.events.find().pretty();
{
  "_id" : "Party",
  "name" : "Corporate Holiday Party",
  "performers" : [
  {,
    "firstName" : "Michael",
    "lastName" : "Carducci"
  },
  {,
    "firstName" : "Shawn",
    "lastName" : "Popp"
  },
  "eventDate" : ISODate("2017-12-15T00:00:00Z"),
  "showTime" : "19:30"
}
ONE-TO-MANY USING EMBEDDING

- Optimized for read performance of events
- We accept data duplication
- An index on “performer.name” provides
  - efficient lookup for all events for a given performer
  - efficient way to find all performer names (distinct)
- Does not automatically mean there is no master performer collection (where data would be copied from)
ONE-TO-MANY LINKING

```javascript
> db.events.find().pretty();
{
    
    "_id" : "Party",
    "name" : "Corporate Holiday Party",
    "performers" : [
        
        {
            "performer_id" : "Michael"
        },
        
        {
            "performer_id" : "Shawn"
        }
    ],
    "eventDate" : ISODate("2017-12-15T00:00:00Z"),
    "showTime" : "19:30"
}
```
ONE-TO-MANY LINKING

```javascript
> db.performers.find({_id:{$in:['Michael','Shawn']}}).pretty();
{
  "_id" : "Michael",
  "firstName" : "Michael",
  "lastName" : "Carducci",
  "website" : "http://trulymagic.com",
  "performanceStyles" : [
    "CloseUp magic",
    "Stage Magic",
    "Mentalism"
  ],
  "addresses" : [
    {
      "street" : "3537 Stronghold Way",
      "city" : "Parker",
      "state" : "CO",
      "country" : "USA"
    },
    {
      "street" : "123 S. Parker Rd STE 314",
      "city" : "Parker",
      "state" : "CO",
      "country" : "USA"
    }
  ]
}``
ONE-TO-MANY LINKING

```javascript
> db.events.find({eventDate: {$gt: ISODate(2018-05-01)}}).forEach(function(event) {

[... //do whatever you need with the event document
[... //in addition, capture performer id
[... //use it as a key in an object (Map)
[... tmpm[event.performers.performer_id] = true;
[... });

> uniqueIds = Object.keys(tmpm);
> db.performers.find({'_id': {'$in': uniqueIds}});
```
ONE-TO-MANY LINKING

- Optimized for efficient management of mutable data
- Familiar way to organize basic entities
- Code is not used to assemble fetched material into other objects, not disassemble a single ResultSet
  - More complicated queries may be easier to code and maintain with assembly vs disassembly
CAUTIONS

- Avoid Using Large or unbounded Arrays
CAUTIONS

- Avoid Using Large Arrays
CAUTIONS

- Avoid Using Large Arrays
CAUTIONS

- Avoid Using Large Arrays
- Avoid Heavily Nested BSON documents
CAUTIONS

- Avoid Using Large Arrays
- Avoid Heavily Nested BSON documents
- Avoid collection over-normalization
UNBOUNDED ARRAYS

```javascript
// Find a client by ID
client = db.clients.find({_id: 'CarducciInc'}).pretty();
{
    "_id" : "CarducciInc",
    "name" : "Carducci Inc",
    "phone" : "555-1212",
    "events" : [
        "1",
        "2",
        "3"
    ]
}

// Find events associated with a client
events = db.events.find({_id: {$in: client.events}});
```
ALTERNATIVE - PARENT-REFERENCING

```javascript
> db.hosts.findOne()
{
    _id : ObjectId('AAAB'),
    name : 'goofy.example.com',
    ipaddr : '127.66.66.66'
}

> db.logmsg.findOne()
{
    time : ISODate("2014-03-28T09:42:41.382Z"),
    message : 'cpu is on fire!',
    host: ObjectId('AAAB') // Reference to the Host document
}
```
UNBOUNDED ARRAYS

```javascript
client = db.clients.find({_id: 'CarducciInc'}).pretty();
{
    "_id": "CarducciInc",
    "name": "Carducci Inc",
    "phone": "555-1212",
    "events": [
        "1",
        "2",
        "3"
    ]
}

events = db.events.find({_id: {$in: client.events}});
```
STRATEGIC DATA DUPLICATION

```javascript
> events = db.events.find().pretty();
{
  "_id": "Party",
  "name": "Corporate Holiday Party",
  "performers": [ 
    {
      "performer_id": "Michael",
      "firstName": "Michael",
      "lastName": "Carducci"
    },
    {
      "performer_id": "Shawn",
      "firstName": "Shawn",
      "lastName": "Popp"
    }
  ],
  "eventDate": ISODate("2017-12-15T00:00:00Z"),
  "showTime": "19:30"
}
```
EMBEDDING VS LINKING

- **Embedding**
  - Great for Read Performance
    - Pre-aggregated material
    - Complex Structures
  - Great for Insert-only/immutable Designs
  - Inserts might be slower than linking
  - Data integrity for not-belongs-to needs to be managed

- **Linking**
  - flexible
  - Data-integrity is built-in
  - Work is done during Reads
BEST PRACTICE: ASSIGN DYNAMIC STRUCTURE TO A KNOWN NAME
```javascript
> db.performers.find().pretty();
{
    "id" : "Shawn",
    "firstName" : "Shawn",
    "lastName" : "Popp",
    "website" : "http://openthetrapdoor.com",
    "performanceStyles" : [
        "CloseUp magic",
        "Stage Magic",
        "Comedy Magic",
        "Mentalism"
    ],
    "personalData" : {
        "birthday" : ISODate("1980-05-20T00:00:00Z")
    },
    "addresses" : [
        {
            "street" : "123 Magic Street",
            "city" : "Centennial",
            "state" : "CO",
            "zip" : 80123
        }
    ],
    "id" : "Michael",
    "firstName" : "Michael",
    "lastName" : "Carducci",
    "website" : "http://trulymagic.com",
    "performanceStyles" : [
        "CloseUp magic",
        "Stage Magic",
        "Mentalism"
    ],
    "personalData" : {
        "awards" : [
            {
                "name" : "Magic of the Mind",
                "year" : 2004
            }
        ]
    }
}
```
POLYMORPHISM

```javascript
> db.events.find()
{ type: "click", ts: ISODate("2015-03-03T12:34:56.789Z"),
data: { x: 123, y: 625, adId: "AE23A" } }

{ type: "click", ts: ISODate("2015-03-03T12:35:01.003Z"),
data: { x: 456, y: 611, adId: "FA213" } }

{ type: "view", ts: ISODate("2015-03-03T12:35:04.102Z"),
data: { scn: 2, reset: false, ... } }

{ type: "click", ts: ISODate("2015-03-03T12:35:05.312Z"),
data: { x: 23, y: 32, adId: "BB512" } }

{ type: "close", ts: ISODate("2015-03-03T12:35:08.774Z"),
data: { snc: 2, logout: true, mostRecent: [ ... ] } }

{ type: "click", ts: ISODate("2015-03-03T12:35:10.114Z"),
data: { x: 881, y: 913, adId: "F430" } }
```
SEARCHING BY ARRAY

```javascript
const performer = {
  "performanceStyles": [
    "CloseUp magic",
    "Stage Magic",
    "Mentalism"
  ],
  "personalData": {
    "awards": [
      { "name": "Magic of the Mind", "year": 2004 }
    ]
  },
  "addresses": [
    { "street": "3537 Stronghold Way", "city": "Parker", "state": "CO", "country": "USA" }
  ]
}

db.performers.ensureIndex({"performanceStyles":1});

db.performers.find({"performanceStyles":"Mentalism"});
```
DOCUMENT VALIDATION

> db.createCollection("books", { "validator": 
> { $and: [ 
> { "title": { $type: "string" } }, 
> { "publishDate": { $type: "date" } }, 
> { $or: [ 
> { "thumbnail": { $exists: False }}, 
> { "thumbnail": { $type: "binary"}} 
> ] 
> } 
> ] 
> });
SCHEMA SOFT VERSIONING

```javascript
> db.createCollection("books", { "validator":
  { $or: [
    { $and: [ { "v": 1},
      { "title": {${type: "string"}} } ]
    },
    { $and: [ { "v": 2},
      { "title": {${type: "string"}} },
      { "publishDate": {${type: "date"}} },
      { $or: [ ]
        { "thumbnail": {${exists: False}}},
        { "thumbnail": {${type: "binary"}}}
      ]
    }]
  ]
});
```
SUMMARY

- Physical Design is different in MongoDB
  - Basic data design principles stay the same
- Focus on how an application accesses/manipulates the data
- Seek out and capture belongs-to/1:1 relationships
- Use substructure to better align code objects
- Be polymorphic
- Evolve the schema to meet requirements as they change
DON’T LINK THE DOCUMENTS
SCHEMA DESIGN BEST PRACTICES

- favor embedding unless there is a compelling reason not to
favor embedding unless there is a compelling reason not to

needing to access an object on its own is a compelling reason not to embed it
SCHEMA DESIGN BEST PRACTICES

- favor embedding unless there is a compelling reason not to
- needing to access an object on its own is a compelling reason not to embed it
- Avoid unbounded arrays
SCHEMA DESIGN BEST PRACTICES

- favor embedding unless there is a compelling reason not to
- needing to access an object on its own is a compelling reason not to embed it
- Avoid unbounded arrays
- Don’t fear application-level joins
SCHEMA DESIGN BEST PRACTICES

- favor embedding unless there is a compelling reason not to
- needing to access an object on its own is a compelling reason not to embed it
- Avoid unbounded arrays
- Don’t fear application-level joins
- Consider the write/read ratio when denormalizing.
SCHEMA DESIGN BEST PRACTICES

- favor embedding unless there is a compelling reason not to
- needing to access an object on its own is a compelling reason not to embed it
- Avoid unbounded arrays
- Don’t fear application-level joins
- Consider the write/read ratio when denormalizing.
- Structure data based on anticipated data-access patterns
INDEXES IN MONGODB

Patterns & Best Practices
SELECTING WHAT TO INDEX

Optimize for workload

Write Performance  Read Performance
MONGODB INDEX LIMITATIONS

- A collection cannot have more than 64 indexes.
- Index entries cannot exceed 1024 bytes.
- The name of an index must not exceed 125 characters (including its namespace).
- In-memory sorting of data without an index is limited to 32MB. This operation is very CPU intensive, and in-memory sorts indicate an index should be created to optimize these queries.
MONGODB INDEX BUILDING TECHNIQUES

- Foreground Index Build
MONGODB INDEX BUILDING TECHNIQUES

- Foreground Index Build
- Background Index Build
MONGODB INDEX BUILDING TECHNIQUES

- Foreground Index Build
- Background Index Build
- Rolling Index Build
INDEX BEST PRACTICES
USE A COMPOUND INDEX, NOT INDEX INTERSECTION
USE A COMPOUND INDEX, NOT INDEX INTERSECTION
AVOID LOW SELECTIVITY INDEXES
USING REGULAR EXPRESSIONS
INEQUALITY QUERIES CAN BE INEFFICIENT
ELIMINATE UNNECESSARY INDEXES
USE PARTIAL INDEXES WHERE APPROPRIATE
Patterns, Pitfalls & best practices

PERFORMANCE

Updates - 8 co

Updates per second

Concurrent

@danveloper
AVOID GROWING DOCUMENTS
USE FIELD MODIFIERS
PAY ATTENTION TO BSON DATA TYPES
PREALLOCATE DOCUMENTS
PREALLOCATE DOCUMENTS
USE _ID FOR YOUR OWN PURPOSES
USE COVERED INDEXES
USE COLLECTIONS AND DATABASES TO YOUR ADVANTAGE
USE COLLECTIONS AND DATABASES TO YOUR ADVANTAGE
OPERATIONS
BEST
PRACTICES
DURABILITY
"..... If you were stupid enough to totally ignore durability just to get benchmarks, I suggest you pipe your data to /dev/null. It will be very fast."

–garl t
WRITE ACKNOWLEDGED
JOURNAL ACKNOWLEDGED
REPLICA ACKNOWLEDGED
MAJORITY ACKNOWLEDGED
CONSISTENCY
DESIGN YOUR SCHEMA FOR SCALE
OPTIMIZE YOUR INDEXES
OPTIMIZE YOUR HARDWARE
KNOW YOUR PERFORMANCE METRICS
SCALING MONGO (OUT)
“Shards are the secret ingredient in the web scale sauce. They just work.”

–gar / t
CHOOSE YOUR SHARD KEY WISELY
CHOOSING A SHARD KEY

- Cardinality
CHOOSING A SHARD KEY

- Cardinality
- Insert Scaling
CHOOSING A SHARD KEY

- Cardinality
- Insert Scaling
- Query Isolation
SHARDING PITFALLS
DON’T USE A MONOTONICALLY INCREASING SHARD KEY (LIKE OBJECTID)
DON'T USE A MONOTONICALLY INCREASING SHARD KEY (LIKE OBJECTID)
USE HASHED SHARDING
YOU CAN’T UPDATE YOUR SHARD KEY
YOU CAN’T UPDATE YOUR SHARD KEY
MISC SHARDING BEST PRACTICES

- Add capacity before it is needed.
MISC SHARDING BEST PRACTICES

- Add capacity before it is needed.
- Run 3+ configuration servers to provide redundancy
MISC SHARDING BEST PRACTICES

- Add capacity before it is needed.
- Run 3+ configuration servers to provide redundancy
- Use Replica Sets
MISC SHARDING BEST PRACTICES

- Add capacity before it is needed.
- Run 3+ configuration servers to provide redundancy
- Use Replica Sets
- Pre-split bulk inserts
PRACTICE PRINCIPLE OF LEAST PRIVILEGE
ENCRYPT DATA IN TRANSPORT
ENCRYPT DATA AT REST
ENCRYPT DATA AT REST

- The MongoDB Encrypted storage engine
- Certified database encryption solutions from MongoDB partners such as IBM and Vormetric
- Logic within the application itself
READ-ONLY, REDACTED VIEWS
MISC BEST PRACTICES
MISC BEST PRACTICES

- Always Use Replica Sets
MISC BEST PRACTICES

- Always Use Replica Sets
- Use the latest versions
MISC BEST PRACTICES

- Always Use Replica Sets
- Use the latest versions
- Turn on journaling by default
MISC BEST PRACTICES

- Always Use Replica Sets
- Use the latest versions
- Turn on journaling by default
- Your working set should fit in memory
Thank you
Michael Carducci
@MichaelCarducci
GREAT INDIAN DEVELOPER SUMMIT 2019

Conference: April 23-26, Bangalore

Register early and get the best discounts!

www.developersummit.com  @greatindiandev  bit.ly/gidslinkedin