Humongous MongoDB

Sean Corfield
World Singles llc

Agenda

- Scaling MongoDB - Concepts
- Replica Sets & Sharding
- Read Preference, Write Concern, Etc
- Map/Reduce
- Aggregation

You Might Prefer...

- Queries and Looping and Grouping
- Enterprise JavaScript Workflows
- Using PhoneGap to Build Mobile Apps
  - Deep Dive - two hours, halfway thru!
- Caching in ColdFusion 10

Me

- Functional Programming in the 80's
- Object-Oriented Programming in the 90's
- Web / Dynamic Programming in the 00's
- Mostly Clojure today

Me & MongoDB

- Using MongoDB in production (2 years)
- Took 10gen "MongoDB for Developers" course (Python + MongoDB)
- Lead maintainer for Clojure's MongoDB wrapper "CongoMongo"

Agenda

- **Scaling MongoDB - Concepts**
- Replica Sets & Sharding
- Read Preference, Write Concern, Etc
- Map/Reduce
- Aggregation

Scaling Concepts

- Master / slave for traditional DBs
  - One master
  - One or more slaves
- Usually scale "up" not "out"

Scaling Concepts

- MongoDB replaces "master / slave" with "replica set" for failover & load distribution
- MongoDB adds sharded clusters to support very large data sets - horizontal scale out

Replica Set Concepts

- A replica set contains any number of (mostly) identical nodes
- A subset of these vote to elect a primary
- All remaining nodes are then secondaries
- Writes go to the primary node and replicate to the secondary nodes

Replica Set Concepts

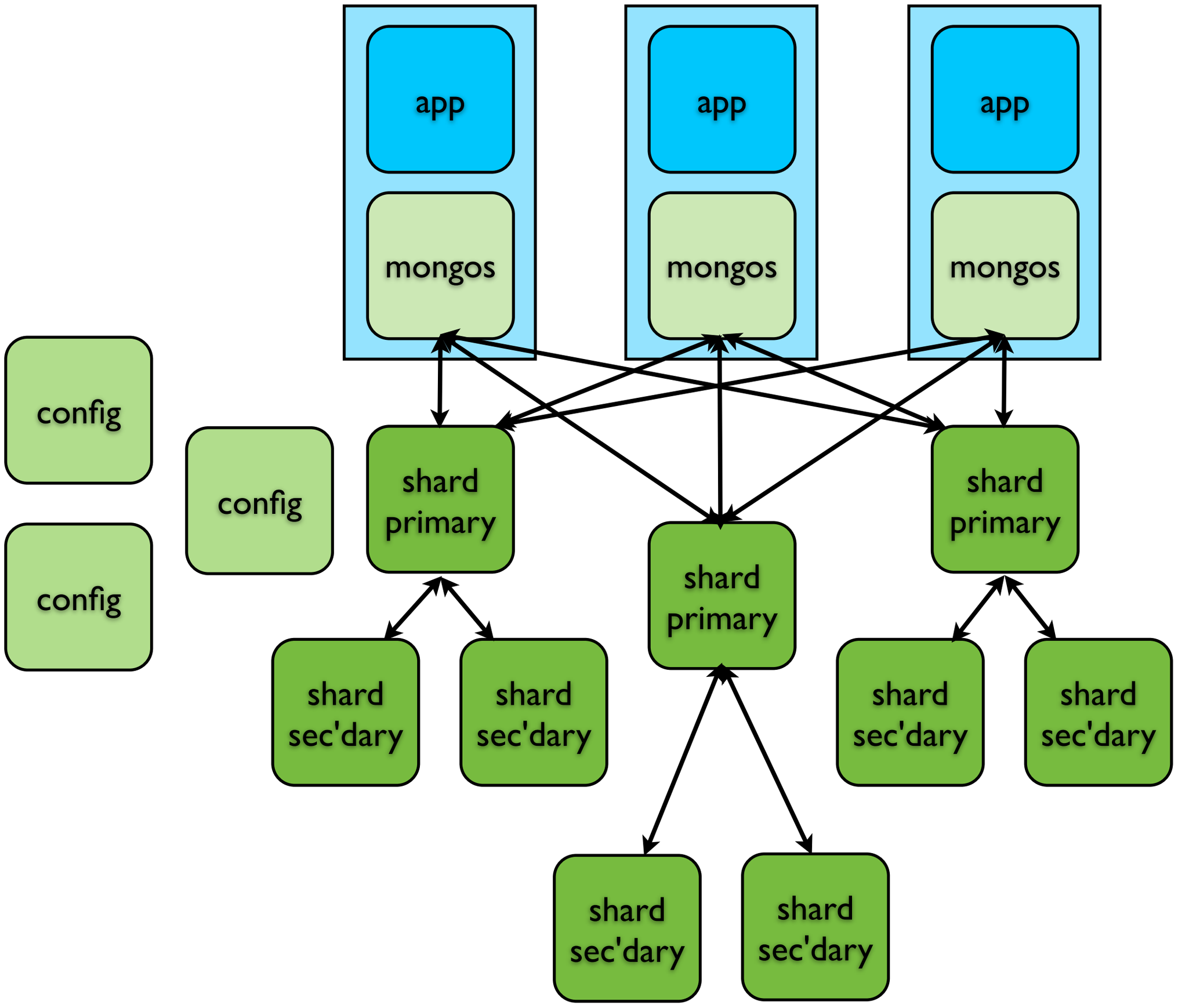
- Reads generally go to the primary node but can be performed against secondaries
  - Per connection or per operation
  - Can also be targeted to specific nodes
    - Tagged of nodes and reads

Replica Set Concepts

- Nodes may also be
  - Secondary only
  - Hidden
  - Arbiter
  - Non-voting
  - Delayed

Sharding Concepts

- Multiple "shards"
  - Servers or clusters (replica sets)
- Configuration server (or a cluster)
- Shard server proxy (mongos process)
  - Lightweight, can have one per app server


app
app
app
mongos
mongos
mongos
config
config
config
shard
primary
shard
primary
shard
primary
shard
sec'dary
shard
sec'dary
shard
sec'dary
shard
sec'dary
shard
sec'dary
shard
sec'dary

Sharding Concepts

- A collection is split across the shards
  - Automatic, based on a key column
  - Automatically balanced across shards
  - Reads directed to appropriate shards
    - May run on all shards, then aggregate

Agenda

- Scaling MongoDB - Concepts
- **Replica Sets & Sharding**
- Read Preference, Write Concern, Etc
- Map/Reduce
- Aggregation

Replica Set Setup

- Start MongoDB servers as replica nodes
  - add --replSet {rsName}
  - specify unique ports and --dbpath folders!

Replica Set Setup

- Connect via mongo shell to one server
- initiate the replica set
- add the other servers to that

Replica Sets

- Either:

```
conf = { _id: "rsName",
    members: [ { _id: 0,
                 host: "server-x:27017" } ] }
rs.initiate( conf )
```

Replica Set Setup

- Or:
  rs.initiate()
- Creates default rs.conf()
- Now add the others:
  rs.add("server-y:27017")
  rs.add("server-z:27017")

Replica Set Demo

- Let's see a real replica set running locally!
  - Must use local machine name
  - Must use different ports for each instance

Sharding Setup

- Start config server(s)
  - mongod --configsvr --dbpath /data/cfg
- Start mongos process(es)
  - mongos --configdb server-c:27019
- Start server (or replica set) for each shard
  - mongod --dbpath /data/sh1

Sharding Setup

- Add each shard to configuration
  - sh.addShard("server-s:27100")
- Enable sharding for the database
  - sh.enableSharding("mydb")
- Enable sharding for a collection
  - sh.shardCollection("mydb.coll",{thekey:1})

Sharding In Use

- Connect to the mongos server (or cluster)
- Interact with the database as usual
- Data automatically moved between shards
- Reads automatically routed based on key

Sharding In Use

- If a query includes the shard key, it will be routed directly to the appropriate server
- If a query does not include the shard key (or uses a range), the query will be sent to multiple servers and the results merged in the mongos process

Sharding Demo

- Let's see it running!

Agenda

- Scaling MongoDB - Concepts
- Replica Sets & Sharding
- **Read Preference, Write Concern, Etc**
- Map/Reduce
- Aggregation

Read Preference

- Normally all reads go to the primary
  - Just like writes
- As we saw, you cannot normally read from a secondary unless you explicitly allow it
- Sometimes you want to spread the read load or read from nearby servers (in a geographically distributed cluster)

Read Preference

- Available options:
  - primary - default
  - primaryPreferred
  - secondary
  - secondaryPreferred
  - nearest

Read Preference

- Secondaries can return stale data!
- Can specify preference
  - Per connection
  - Per collection
  - Per operation
- Not currently supported by cfmongodb

Write Concern

- By default, receipt of a write command is acknowledged by the server (but may not yet have been written to disk)
- You can control what operations you wait for before a write command returns

Write Concern

- Errors Ignored - do not use!
- Unacknowledged
- Fire and forget
- Network errors are detected
- Used to be the default

Write Concern

- Acknowledged
  - Current default (as of late 2012)
  - Network errors, duplicate keys etc
- Journaled
  - The update is written to local journal
  - Durable - will survive shutdown / crash

Write Concern

- Replica Acknowledged
  - The update is written to one or more secondaries in a replica set
  - Can specify number or "majority"
  - Specifying number will block until that many secondaries have the write (and therefore it can block "forever"!)

Write Concern

- cfmongodb supports per-connection only
  - Not per-operation
- mongoClientOptions arg to MongoConfig
  - writeConcern field of that struct

WriteConcern

- Retrieve from a Java class
  - mongoFactory.getObject(
    "com.mongodb.WriteConcern"
).UNACKNOWLEDGED
- http://api.mongodb.org/java/2.10.1/com/mongodb/WriteConcern.html

Agenda

- Scaling MongoDB - Concepts
- Replica Sets & Sharding
- Read Preference, Write Concern, Etc
- **Map/Reduce**
- Aggregation

Map/Reduce

- Intended for complex data processing
- Batch operation, not real time!
- You provide map, reduce, finalize functions written in JavaScript (as strings!)

Map/Reduce

- people = mongo.getDBCollection("people");
  people.mapReduce(
  map = "function(){ ...}",
  reduce = "function(key,values){ ... }",
  outputTarget = ...);
- For finalize you must use a DB command
- Examples in cfmongodb aggregation folder

Agenda

- Scaling MongoDB - Concepts
- Replica Sets & Sharding
- Read Preference, Write Concern, Etc
- Map/Reduce
- **Aggregation**

Aggregation Framework

- Added in MongoDB 2.2
- Native, pipeline-based functions
  - project (SELECT), match (WHERE), group (GROUP BY), sort (ORDER BY), unwind, skip, limit, geoNear (new in 2.4)
- Simple aggregate() function takes each operation as an argument in order

Aggregation Framework

- result = musicians.aggregate(
  { "$group" : { "_id" : "$status",
                 "total" : { "$sum" = 1 } } },
  { "$project" : { "status" : "$_id",
   "numberOfMusicians" : "$total",
                         "_id" : 0 } }
);

Aggregation Framework

- Equivalent to
  - SELECT COUNT(*) AS total, status
    FROM musicians
    GROUP BY status
- More examples in cfmongodb aggregation folder

Summary

- MongoDB
  - Supports simple, robust clustering with automatic failover
  - Supports data sharding to provide automatic horizontal scalability
  - Provides plenty of control over reading and writing in a clustered environment

Summary

- cfmongodb supports
  - Robust, scalable clustering
  - Big Data manipulation through map/reduce and the aggregation framework
  - Write Concern (partially)
- It's open source - contribute!

Resources

- http://cfmongodb.riaforge.org
- http://mongodb.org
- http://docs.mongodb.org
- http://www.10gen.com

Q & A?

- @seancorfield
- http://corfield.org
- sean@corfield.org