MongoDB large scale data-centric architectures

QConSF 2012
Kenny Gorman
Founder, ObjectRocket

@objectrocket @kennygorman

MongoDB at scale

- Designing for scale
- Techniques to ease pain
- Things to avoid

What is scale?

- Scale; massive adoption/usage
- Scale; a very big, busy, or tricky system.
- Scale; I just want to sleep.
- Scale; The docs just seem silly now.
- Scale; Am I the only one with this problem?

Vintage playbook

- No joins, foreign keys, triggers, stored procs
- De-normalize until it hurts
- Split vertically, then horizontally.
  - Conventional wisdom. eBay an early pioneer.
- Many DBA's, Sysadmin's, storage engineers, etc
- Huge hardwarez
- You have your own datacenter or colo-location
- You realize your ORM has been screwing you
- You better have some clever folks on staff, shit gets weird at scale

Example:

```
while True:
    try:
        add_column()
        exit()
    exception e:
        print ("%s; crud") % e
```

Vintage scaling playbook

Scaling today

- Many persistence store options
- Horizontal scalability is expected
- Cloud based architectures prevalent
  - Hardware and data centers are abstracted from developers
- Focus on rapid development
- Mostly developers, maybe some devops
- Expectations that stuff just works
- Technologies are less mature, less tunables

ObjectRocket

Enter MongoDB

- Document based ~~NoSQL~~ database
- JSON/BSON (www.bson.org)
- Developers dream
- OPS nightmare (for now)
- Schema-less
- Built in horizontal scaling framework
- Built in replication
- ~65% deployments in the cloud

MongoDB challenges

- The lock scope
- Visibility
- Schema
- When bad things happen

A MongoDB document

```
{
  _id : ObjectId("4e77bb3b8a3e000000004f7a"),
  when : Date("2011-09-19T02:10:11.3Z",
  author : "alex",
  title : "No Free Lunch",
  text : "This is the text of the post",
  tags : [ "business", "ramblings" ],
  votes : 5,
  voters : [ "jane", "joe", "spencer" ],
 }
```

MongoDB keys for success at scale

- Design Matters!

Design for scale; macro level

- Keep it simple
- Break up workloads
- Tune your workloads
- NoORM; dump it
- Shard early
- Replicate
- Load test pre-production!

Your success is only as good as the thing you do a million times a second

Design for scale; specifics

- Embedded vs not
- Indexing
    - The right amount
    - Covered
- Atomic operations
- Use profiler and explain()

Example; document embedding

```
// yes, guaranteed 1 i/o
{userid: 100, post_id: 10, comments:["comment1","comment2"..]}
db.blog.find({"userid":100}).explain()
{ ..., "nscannedObjects" : 1, ... }

// no
{userid: 100, post_id: 10, comment: "hi, this is kewl"}
{userid: 100, post_id: 10, comment: "thats what you think"}
{userid: 100, post_id: 10, comment: "I am thirsty"}
db.blog.find({"userid":100}).explain()
{ ..., "nscannedObjects" : 3, ... }
```

Example; covered Indexes

```
mongos> db.foo.find({"foo":1},{_id:0,"foo":1}).explain()
{
    "cursor" : "BtreeCursor foo_-1", "isMultiKey" : false,
    "n" : 1,
    "nscannedObjects" : 1, "nscanned" : 1,
    "nscannedObjectsAllPlans" : 1,
    "nscannedAllPlans" : 1, "scanAndOrder" : false,
    "indexOnly" : true,
    "nYields" : 0, "nChunkSkips" : 0,
    "millis" : 0, "indexBounds" : { "foo" : [[1,1]]},
    "millis" : 0
}
```

Design for scale

- Shard keys
  - Tradeoffs
  - Local vs Scattered
  - Figure out at design time

Example; Shard Keys

- Tuning for writes
- Queries are scattered

```
{
    _id: ObjectId("4e77bb3b8a3e000000004f7a"),
    skey: md5(userid+date),  // shard key
    payload: {...}
}
```

Example; Shard Keys

- Tuning for reads
  - Localized queries
  - Writes reasonably distributed

```
{
    userid: 999,    // shard key
    post: {"userid":23343,
           "capt":"hey checkout my pic",
           "url":"http://www.lolcats.com"
           }
}
```

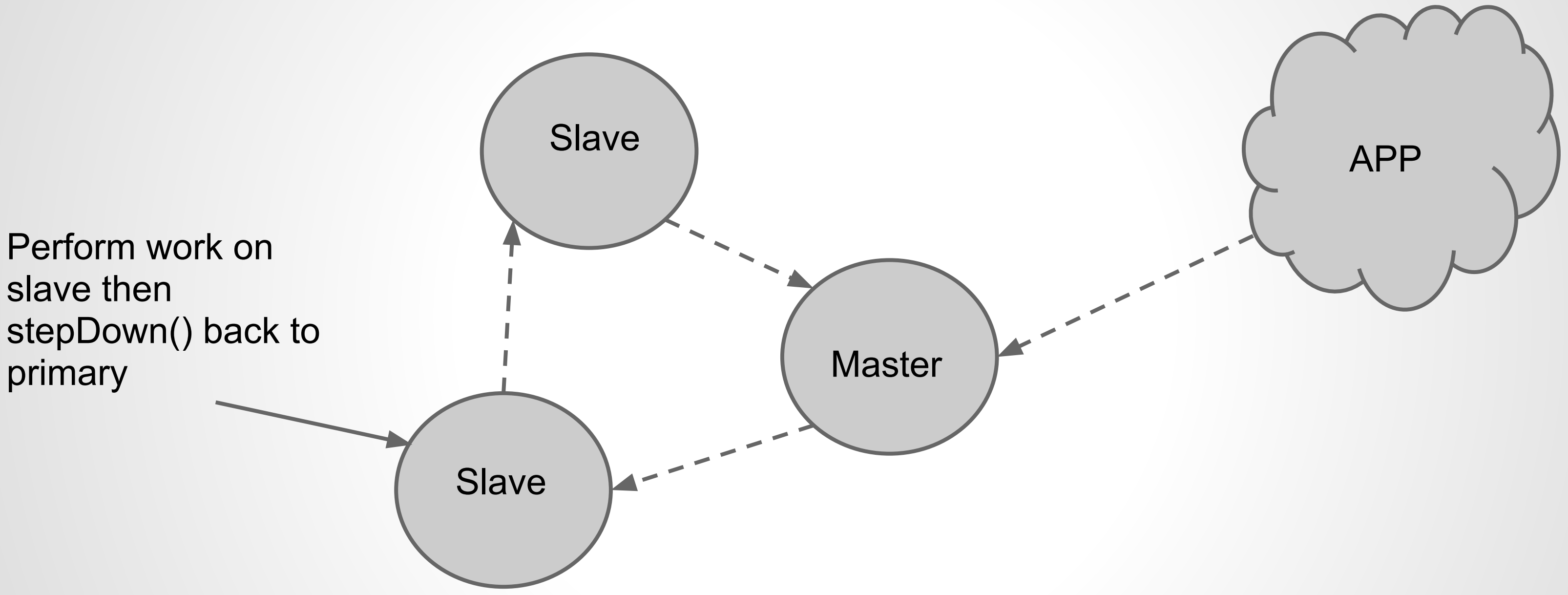
Design for scale; architecture

- Engage all processors
    - Single writer lock
    - Concurrency
- Replication
    - Understand elections, and fault zones
    - Understand the 'shell game', rebuilding slaves
        - Fragmentation
    - Client connections, getLastError
- Sharding
    - Pick good keys or die
    - Get enough I/O

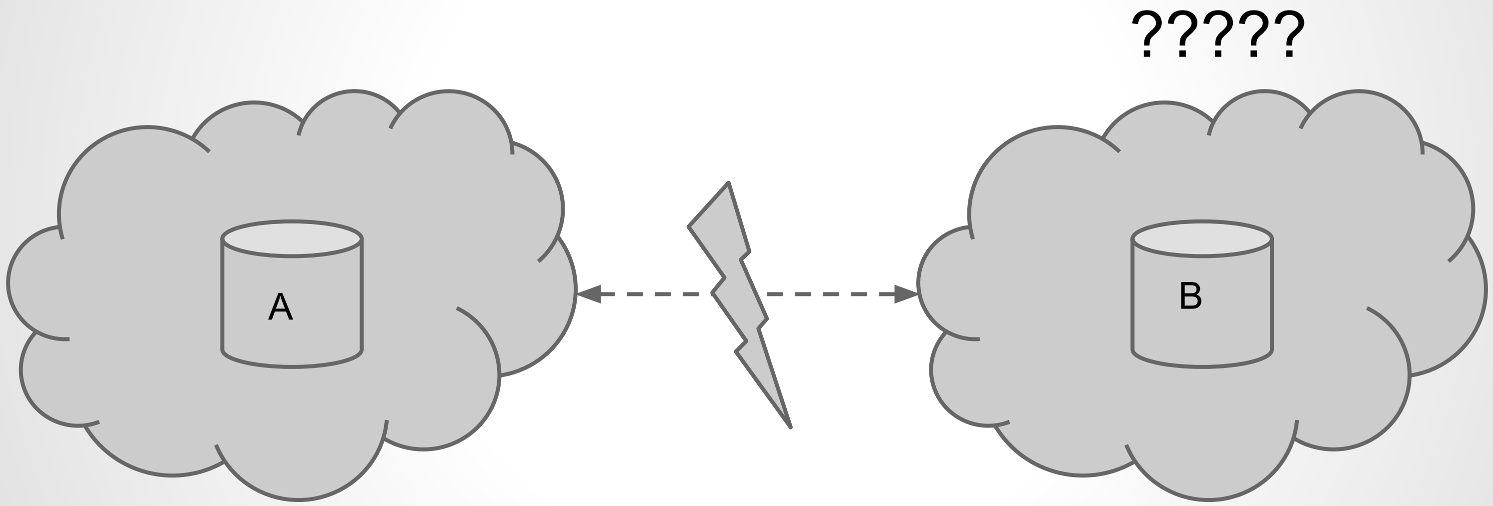
Design for scale; architecture

- I/O
  - You need it
  - Conventional wisdom is wrong
  - Maybe they don't have big databases?

Example; 'shell game'



Example; network partition



"replSet can't see a majority, will not try to elect self"

Example; write concern

```
// ensure data is in local journal

BasicDBObject doc = new BasicDBObject();
doc.put("payload","foo");
coll.insert(doc, WriteConcern.SAFE);
```

Random parting tips

- Monitor elections, and who is primary
- Write scripts to kill sessions > Nms or based on your architecture
- Automate or die
- Tools
  - Mongostat
  - Historical performance

Gotchas, risks, shit that will make you nuts

- Logical schema corruption
- That lock!
- Not enough I/O
- Engaging all processors
- Visibility
- Not understanding how MongoDB works
- FUD

Contact

@kennygorman
@objectrocket
kgorman@objectrocket.com

https://www.objectrocket.com
https://github.com/objectrocket/rocketstat