Deep Dive into MongoDB from a developer's point of view

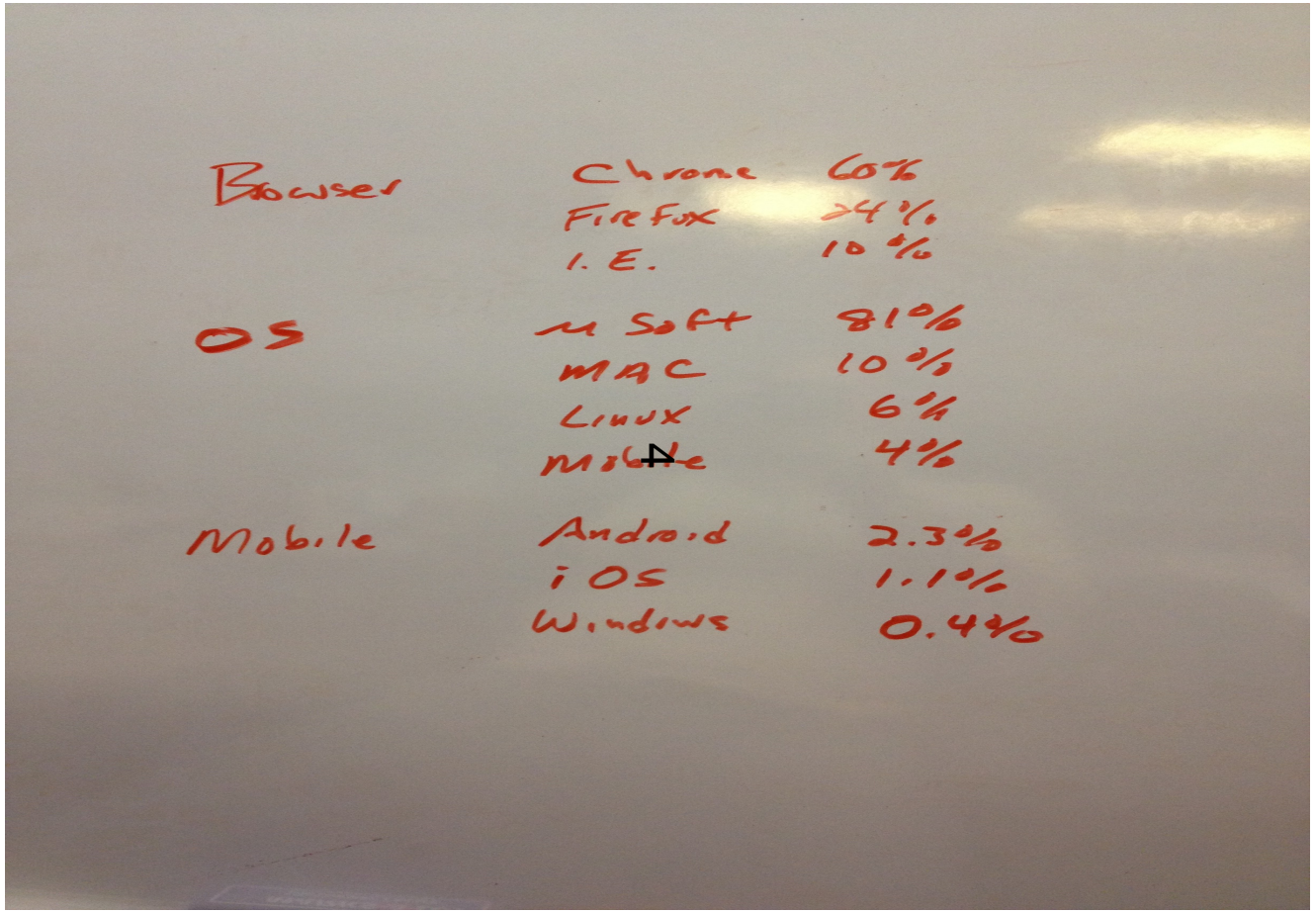
Craig Lawrence

- Software Developer/Lead/Manager in cable/telecom for over 25 years
- Bachelor and Master's degree in Electrical Engineering
- Worked at Ericsson on OpenStream and MDMS since 2008
- Using MongoDB as a developer since 2011

What's New in 2014?

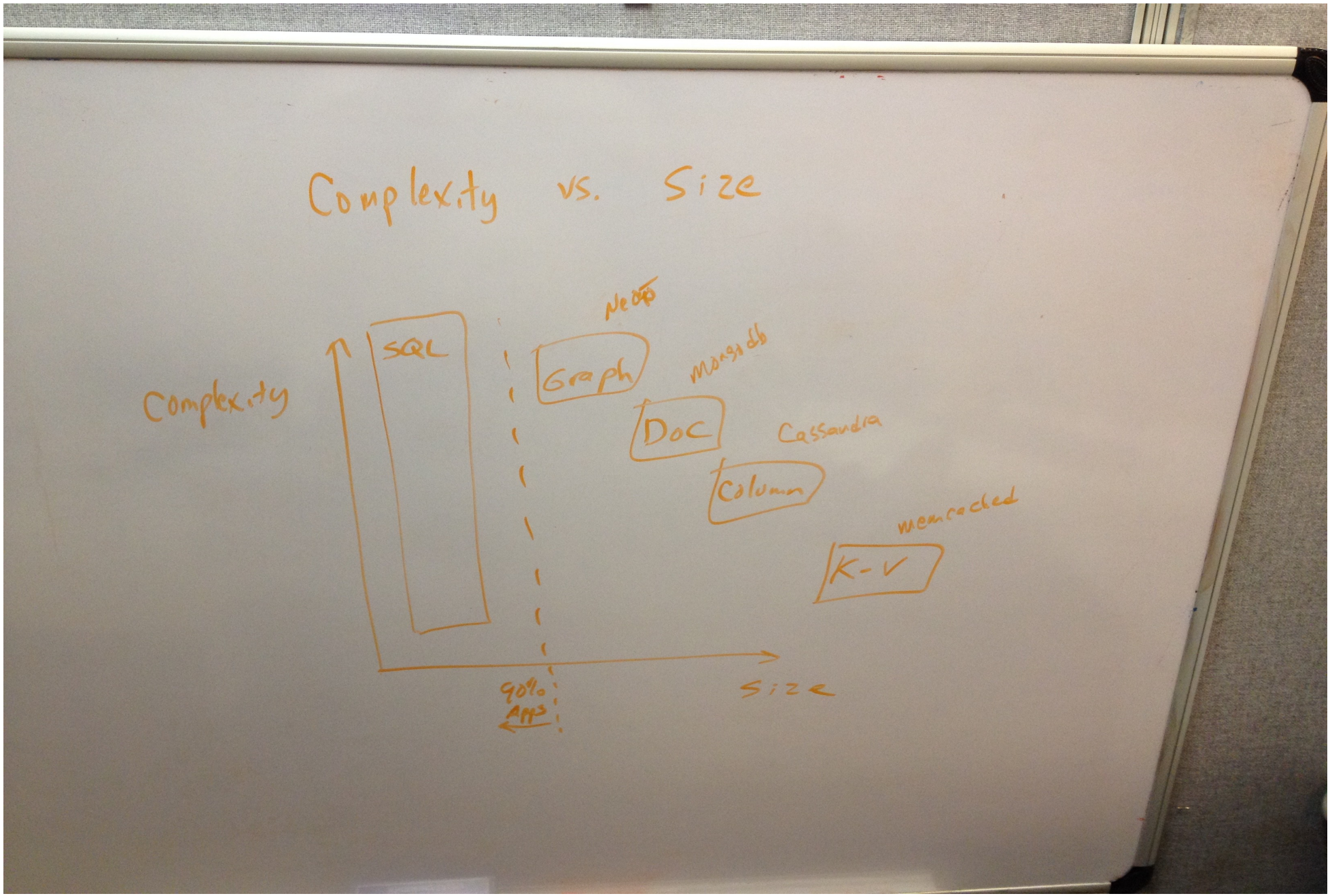
- Browser: Chrome
- OS: Microsoft
- Device: Android

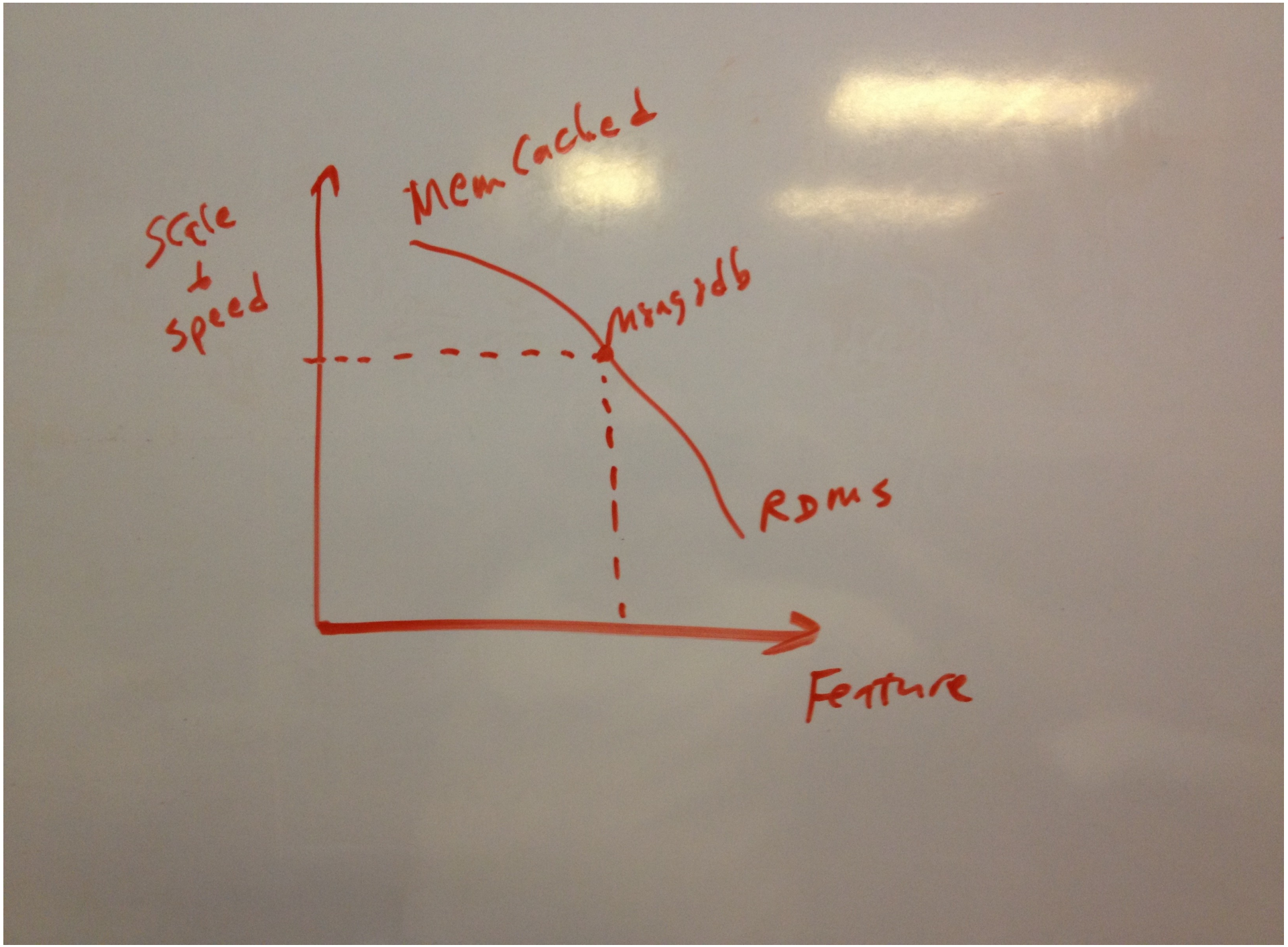
w3schools.com stats



What about Databases?

- Relational vs NoSQL
- MDMS uses MongoDB


Complexity vs. Size
Complexity
SQL
Neo4j
Graph
MongoDB
Doc
Cassandra
Column
memcached
K-V
90% Apps
Size


Scale
+
speed
MemCached
Mongodb
RDMS
Feature

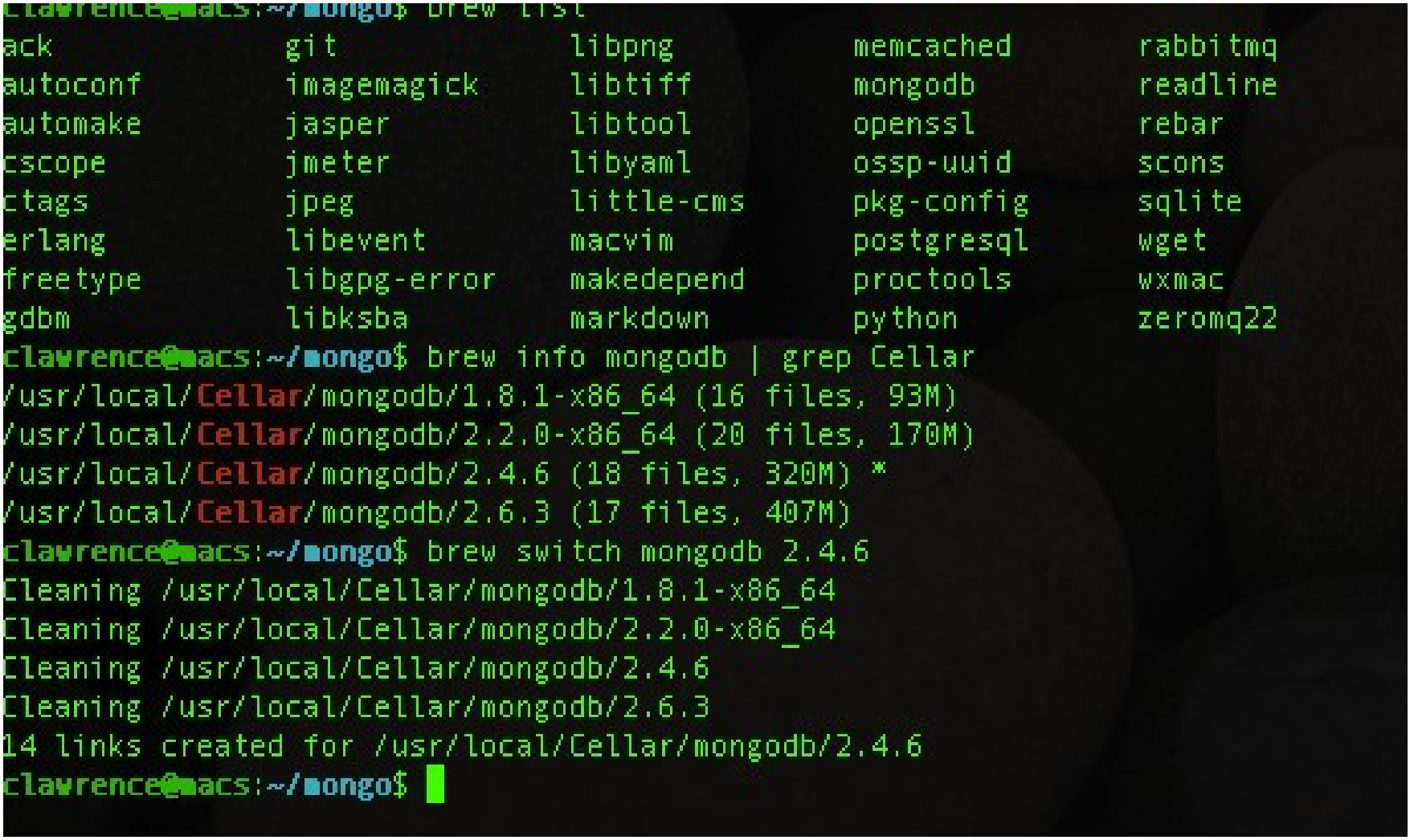
Target Audience

- Software Developers
- Already familiar with Mongodb basics
- Already familiar with Linux

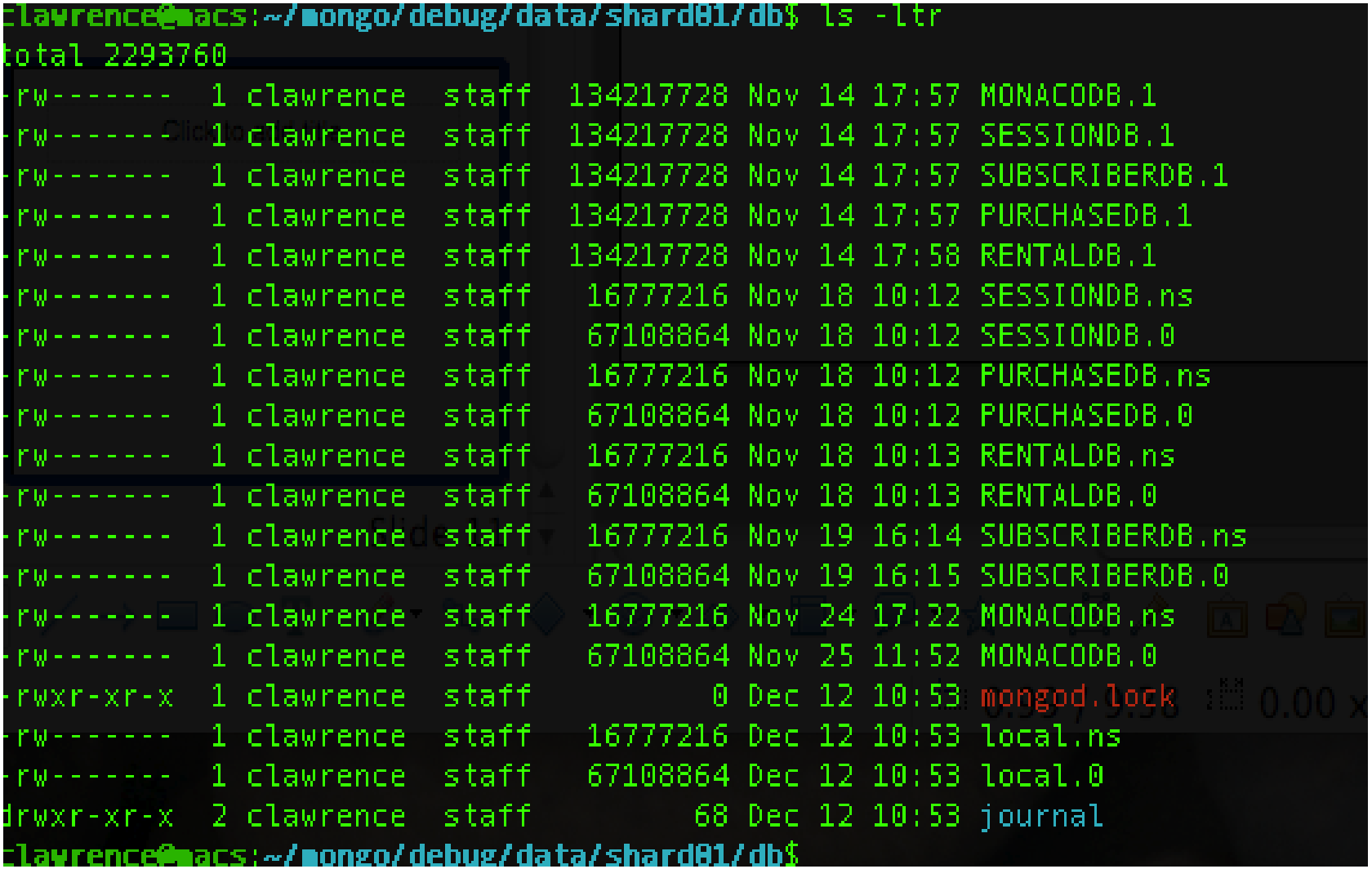
MongoDB basics

- Store Document as BSON
- Docs can reference or embed other Docs
- Javascript Shell
- Replication for High Availability
- Sharding for Scalability

Using Brew on Mac



MongoDB Files



Indexing a Collection

- Faster Read, Slower Writes
- Document move more expensive
- Build index before you have data is faster
- Covered index (explain, hint)
- Unique, sparse, TTL, text, geospacial

```
db.assets.find({ _type: "TermLimitation"}).hint({_id: 1}).explain()
{
        "cursor" : "BtreeCursor _id_",
        "isMultiKey" : false,
        "n" : 38,
        "nscannedObjects" : 256,
        "nscanned" : 256,
        "nscannedObjectsAllPlans" : 256,
        "nscannedAllPlans" : 256,
        "scanAndOrder" : false,
        "indexOnly" : false,
        "nYields" : 0,
        "nChunkSkips" : 0,
        "millis" : 1,
        "indexBounds" : {
                "_id" : [
                        [
                                {
                                        "$minElement" : 1
                                },
                                {
                                        "$maxElement" : 1
                                }
                        ]
                ]
        },
        "server" : "macs:27018",
        "millis" : 1
}
```

```
db.assets.find({ _type: "TermLimitation"})
.hint({_type: 1}).explain()

{
        "cursor" : "BtreeCursor _type_1",
        "isMultiKey" : false,
        "n" : 38,
        "nscannedObjects" : 38,
        "nscanned" : 38,
        "nscannedObjectsAllPlans" : 38,
        "nscannedAllPlans" : 38,
        "scanAndOrder" : false,
        "indexOnly" : false,
        "nYields" : 0,
        "nChunkSkips" : 0,
        "millis" : 0,
        "indexBounds" : {
                "_type" : [
                        [
                                "TermLimitation",
                                "TermLimitation"
                        ]
                ]
        },
        "server" : "macs:27018",
        "millis" : 0
}
~
```

MONGODB Memory Usage

- MongoDB runs fast when working set is in memory.
- Working set: data accessed most frequently (indexes & subset of data)
- Resident memory (includes file system cache & mongod process memory)
- Restarts and dropping caches

Determining Working Set

- Knowledge of data & indexes
- Average document size and index size
- Pages in memory & page Size
- Hard & Soft page faults under load

MONGODB General Info

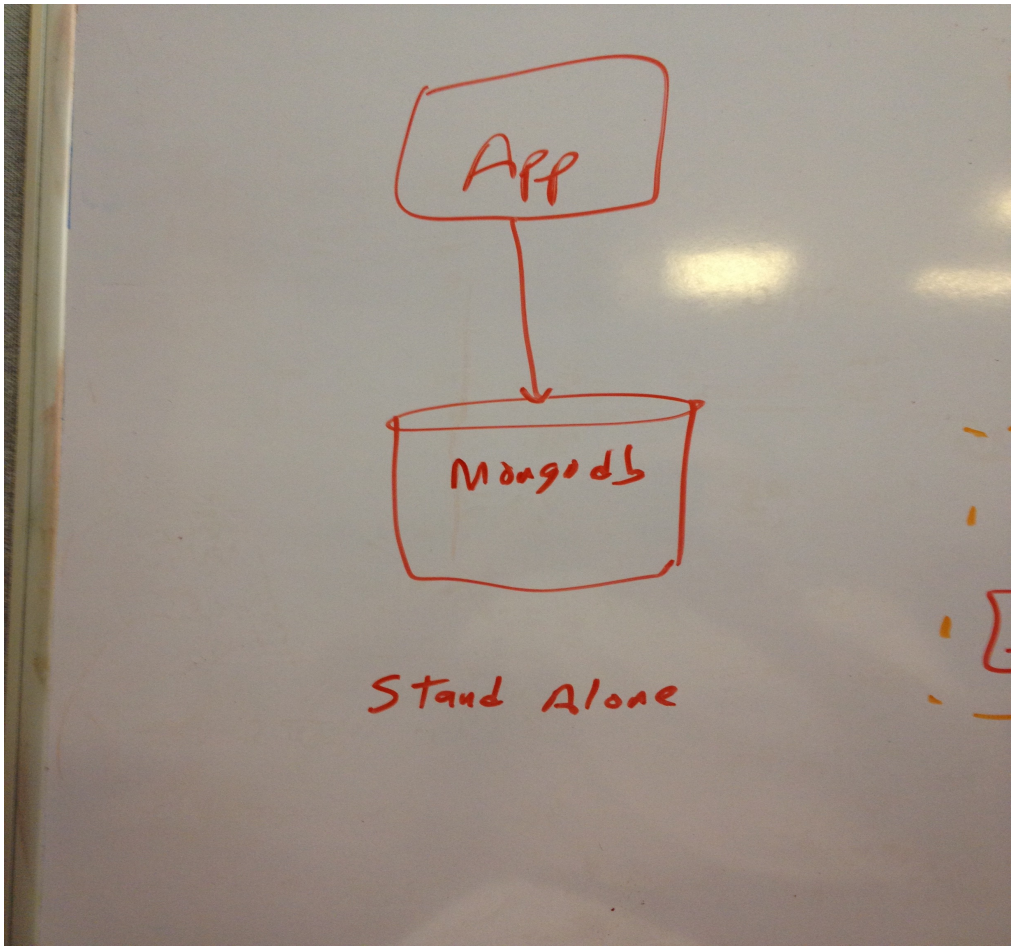
- Syncs Journal every 100msec
- Syncs Data every 60sec
- Preheating Data can be beneficial
- Follow file system/os config recommendations
- Does not track available disk space
- Mongodb is new and evolving

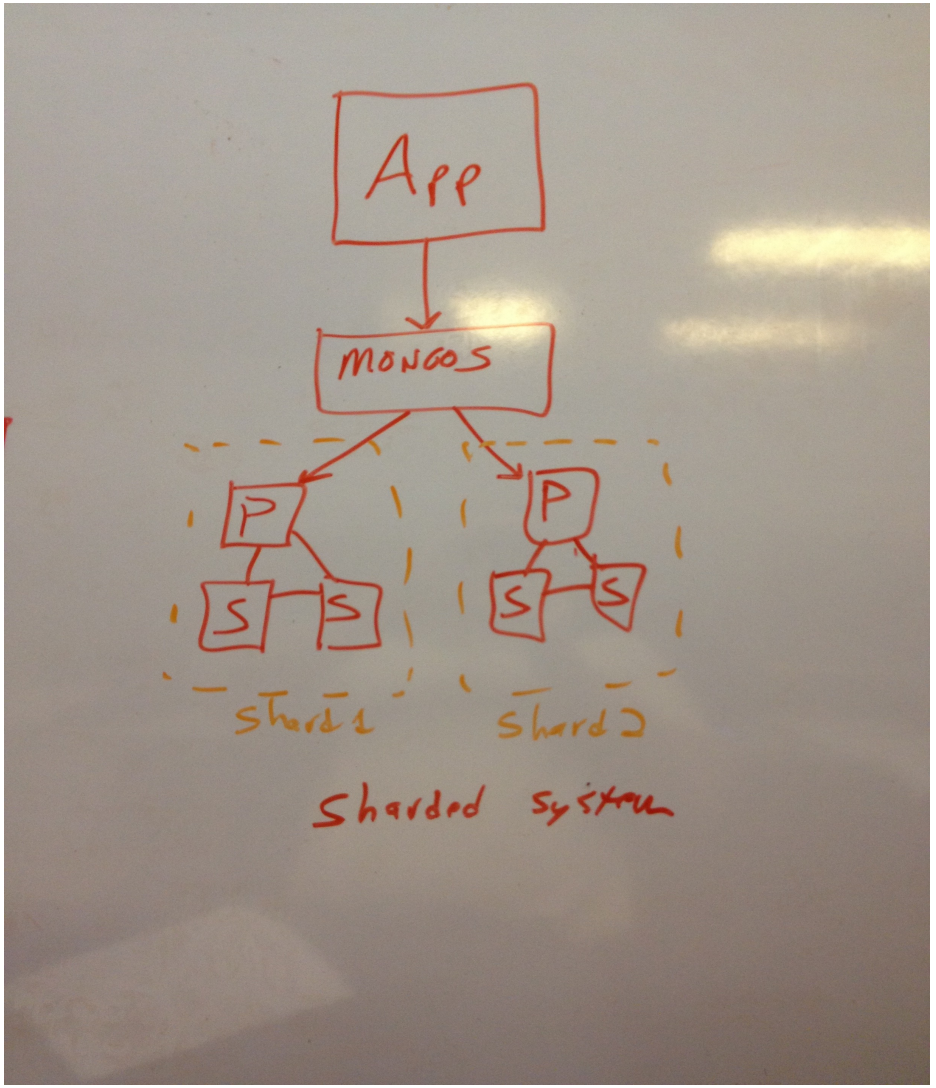
MONGODB Sizing Considerations

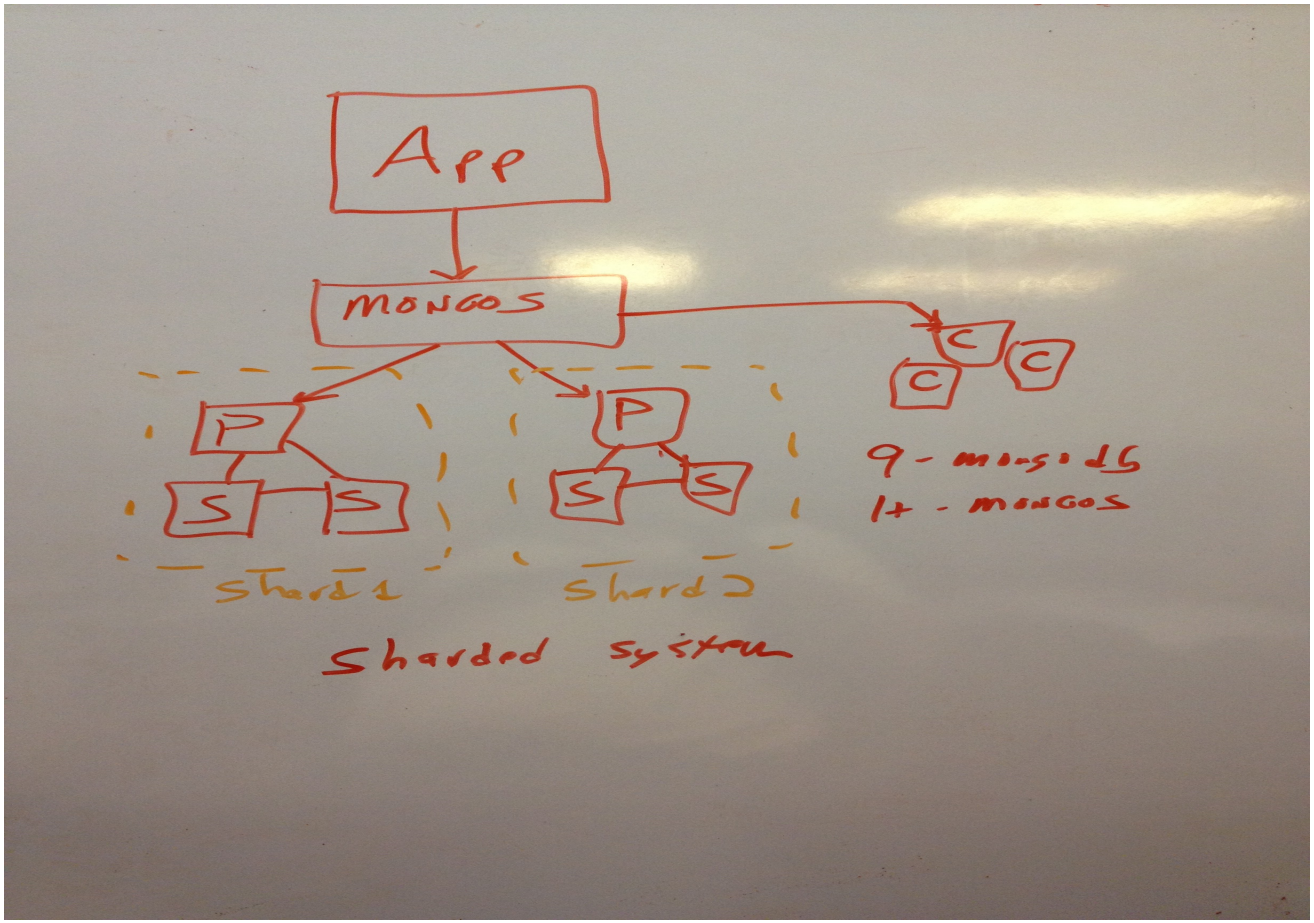
- Database size, number processors, I/O Benchmarking
- TCP connections, file descriptor limits
- Port limitations, connection churn, kernel connection parameters

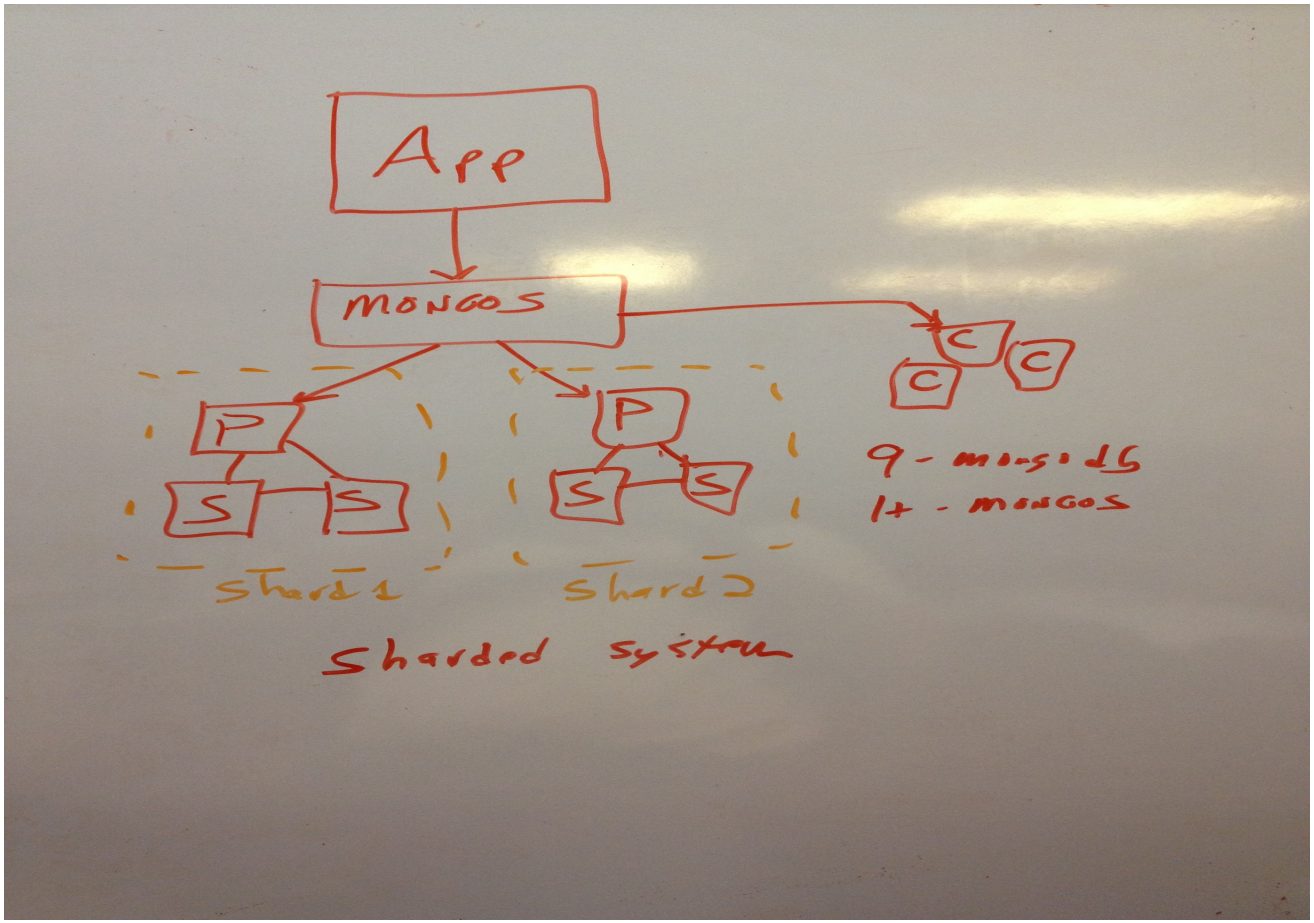
Development Environment

- Typical Standalone Mongod
- Could be 3 system replica set or a sharded system with mongos
- Likely running on a single dev box
- On Mac Brew and mlaunch are nice


App
Mongodb
Stand Alone

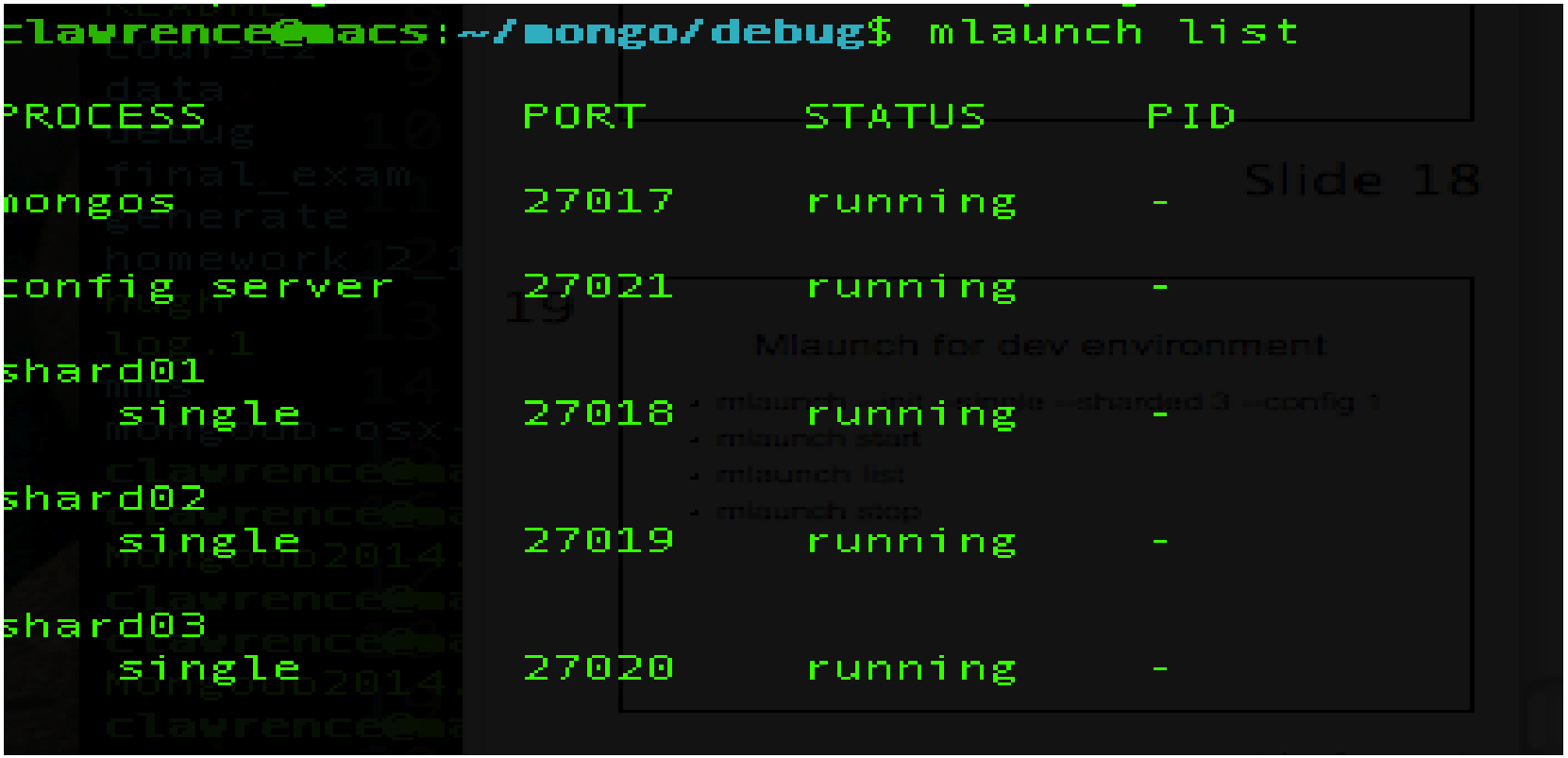

App
mongos
P
P
S
S
S
S
shard 1
shard 2
Sharded system


App
mongos
P
P
S
S
S
S
C
C
C
Shard 1
Shard 2
Sharded system


App
mongos
P
P
S
S
S
S
C
C
C
Shard 1
Shard 2
Sharded system

Mlaunch for dev environment

- mlaunch --init --single --sharded 3 --config 1
- mlaunch start
- mlaunch list
- mlaunch stop


clavrence@macs:~/mongo/debug$ mlaunch list
PROCESS PORT STATUS PID
mongos 27017 running -
config server 27021 running -
shard01
single 27018 running -
shard02
single 27019 running -
shard03
single 27020 running -

Production Deployment

- 3 member replica Set with voting
- Primary, Secondary, Secondary
- Primary, Secondary, Arbitor
- Maybe extras for analytics or backups
- Often distribute replicas across different sites
- Sharding used when necessary

Distributed Replica Sets

- Writes to a remote primary may have longer network delays
- Replica Set chaining can be used to help reduce network delays for updating secondaries

Sharding

- Each Shard normally a replica set
- Sharding requires 3 config servers (1 for dev)
- Sharded writes fail when a config server is down or unavailable
- Unsharded collection will live on one shard
- Sharded collections are split across shards using a shard key
- Shard keys need to be chosen carefully

Shard Balancing

- A load balancer moves chunks of data across all shards
- Chunks are a general abstraction. They are not data size nor number of documents nor load balancing.
- Balancing chunks is not the same as balancing load or data

Sharding Considerations

- Mongos has responsibility to combine queries that span different shards
- Jumbo chunks and empty chunks create problems
- Orphan chunks sometimes show up on secondary reads or direct mongod queries (see mongodb/support-tools orphanage.js)
- AllChunkInfo (https://gist.github.com/normgraham/8716338)

normgraham / allChunkInfo.js

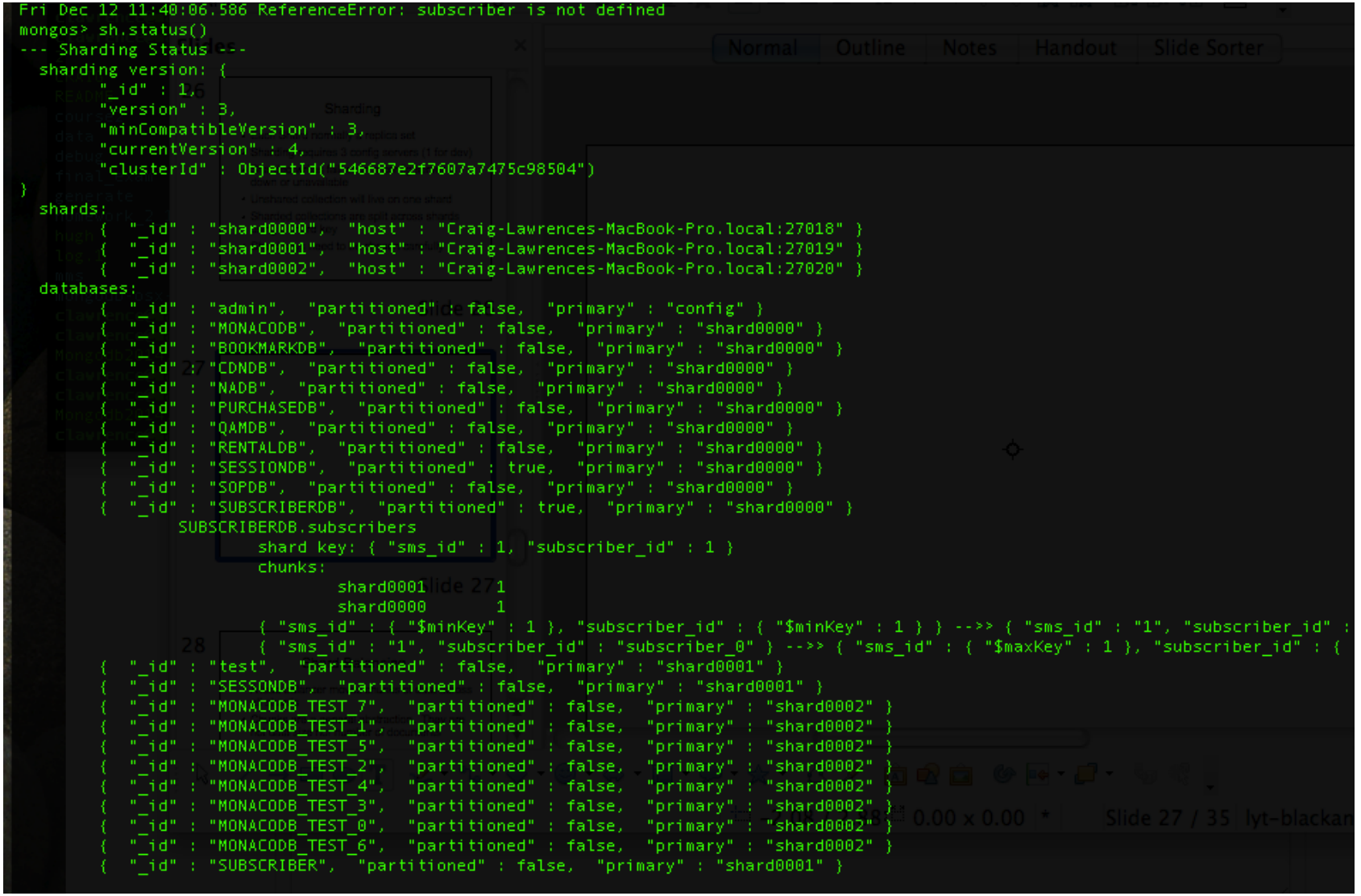
Created on Jan 30

Display detail and aggregate chunk info for a sharded collection. To run, load the following code for allChunkInfo() into a mongos process in your cluster and run allChunkInfo("dbname.collname").

allChunkInfo.js — Raw

```
var allChunkInfo = function(ns){
    var chunks = db.getSiblingDB("config").chunks.find({"ns" : ns}).sort({min:1}); //this will return all c
    //some counters for overall stats at the end
    var totalChunks = 0;
    var totalSize = 0;
    var totalEmpty = 0;
    print("ChunkID,Shard,ChunkSize,ObjectsInChunk"); // header row
    // iterate over all the chunks, print out info for each
    chunks.forEach(
        function printChunkInfo(chunk) {

        var db1 = db.getSiblingDB(chunk.ns.split(".")[0]); // get the database we will be running the comma
        var key = db.getSiblingDB("config").collections.findOne({_id:chunk.ns}).key; // will need this for
        // dataSize returns the info we need on the data, but using the estimate option to use counts is le
        var dataSizeResult = db1.runCommand({datasize:chunk.ns, keyPattern:key, min:chunk.min, max:chunk.ma
        // printjson(dataSizeResult); // uncomment to see how long it takes to run and status
        print(chunk._id+","+chunk.shard+","+dataSizeResult.size+","+dataSizeResult.numObjects);
        totalSize += dataSizeResult.size;
        totalChunks++;
        if (dataSizeResult.size == 0) { totalEmpty++ }; //count empty chunks for summary
        }
    )
    print("**********Summary Chunk Information**********");
    print("Total Chunks: "+totalChunks);
    print("Average Chunk Size (bytes): "+(totalSize/totalChunks));
    print("Empty Chunks: "+totalEmpty);
    print("Average Chunk Size (non-empty): "+(totalSize/(totalChunks-totalEmpty)));
}
```


Fri Dec 12 11:40:06.586 ReferenceError: subscriber is not defined
mongos> sh.status()
--- Sharding Status ---
sharding version: {
"_id" : 1,
"version" : 3,
"minCompatibleVersion" : 3,
"currentVersion" : 4,
"clusterId" : ObjectId("546687e2f7607a7475c98504")
}
shards:
{ "_id" : "shard0000", "host" : "Craig-Lawrences-MacBook-Pro.local:27018" }
{ "_id" : "shard0001", "host" : "Craig-Lawrences-MacBook-Pro.local:27019" }
{ "_id" : "shard0002", "host" : "Craig-Lawrences-MacBook-Pro.local:27020" }
databases:
{ "_id" : "admin", "partitioned" : false, "primary" : "config" }
{ "_id" : "MONACODB", "partitioned" : false, "primary" : "shard0000" }
{ "_id" : "BOOKMARKDB", "partitioned" : false, "primary" : "shard0000" }
{ "_id" : "CDNDB", "partitioned" : false, "primary" : "shard0000" }
{ "_id" : "NADB", "partitioned" : false, "primary" : "shard0000" }
{ "_id" : "PURCHASEDB", "partitioned" : false, "primary" : "shard0000" }
{ "_id" : "QAMDB", "partitioned" : false, "primary" : "shard0000" }
{ "_id" : "RENTALDB", "partitioned" : false, "primary" : "shard0000" }
{ "_id" : "SESSIONDB", "partitioned" : true, "primary" : "shard0000" }
{ "_id" : "SOPDB", "partitioned" : false, "primary" : "shard0000" }
{ "_id" : "SUBSCRIBERDB", "partitioned" : true, "primary" : "shard0000" }
SUBSCRIBERDB.subscribers
shard key: { "sms_id" : 1, "subscriber_id" : 1 }
chunks:
shard0001 1
shard0000 1
{ "sms_id" : { "$minKey" : 1 }, "subscriber_id" : { "$minKey" : 1 } } -->> { "sms_id" : "1", "subscriber_id" :
{ "sms_id" : "1", "subscriber_id" : "subscriber_0" } -->> { "sms_id" : { "$maxKey" : 1 }, "subscriber_id" : {
{ "_id" : "test", "partitioned" : false, "primary" : "shard0001" }
{ "_id" : "SESSONDB", "partitioned" : false, "primary" : "shard0001" }
{ "_id" : "MONACODB_TEST_7", "partitioned" : false, "primary" : "shard0002" }
{ "_id" : "MONACODB_TEST_1", "partitioned" : false, "primary" : "shard0002" }
{ "_id" : "MONACODB_TEST_5", "partitioned" : false, "primary" : "shard0002" }
{ "_id" : "MONACODB_TEST_2", "partitioned" : false, "primary" : "shard0002" }
{ "_id" : "MONACODB_TEST_4", "partitioned" : false, "primary" : "shard0002" }
{ "_id" : "MONACODB_TEST_3", "partitioned" : false, "primary" : "shard0002" }
{ "_id" : "MONACODB_TEST_0", "partitioned" : false, "primary" : "shard0002" }
{ "_id" : "MONACODB_TEST_6", "partitioned" : false, "primary" : "shard0002" }
{ "_id" : "SUBSCRIBER", "partitioned" : false, "primary" : "shard0001" }

Example AllChunkInfo

allChunkInfo("SUBSCRIBERDB.subscribers")
ChunkID,Shard,ChunkSize,ObjectsInChunk
SUBSCRIBERDB.subscribers-sms_id_MinKeysubscriber_id_MinKey,shard0001,432,1
SUBSCRIBERDB.subscribers-sms_id_"1"subscriber_id_"subscriber_0",shard0000,2512,8
**********Summary Chunk Information**********
Total Chunks: 2
Average Chunk Size (bytes): 1472
Empty Chunks: 0
Average Chunk Size (non-empty): 1472

Other Sharding Options

- Tag Based Sharding (pinning)
- Hash Based Sharding (scales writes, may slow queries)

MONGODB Driver Options

- Options vary with different drivers and with different versions of a driver
- Connection timeout, connections per host
- Connections use 1MB per connection and file descriptors too)
- TODO: Show mongoid.yml

Connection Strategy

- Primaries have greatest number of connections
- A 3 shard system may have 20 connection per mongos connection
- Consider maximum possible primary connections (1MB/per, ulimits, 20k typical mongod max)
- Consider limiting mongos and drivers to protect system from reaching overload

MONGODB Failover

- Journaling
- Oplog delays
- Write Concerns
- Read Concerns
- Triggering Rollback

Mongod Tools

- mlaunch (for dev use only)
- mloginfo (log file summary info, see blog.rueckstiess.com)
- mlogfilter
- mplotqueries
- (github.com/rueckstiess/mtools)
- pip install mtools (pymongo, matplotlib)

Example Data for mloginfo

- mloginfo mtools_example.log --connections --distinct --queries --restarts –rsstate
- mlogfilter mtools_example.log --from "2014 Jun 21 00:12:59" --to "+1s" --slow 1000 --operation remove --namespace "grilled.indigo.papaya"
- mplotqueries mtools_example.log --type scatter --group namespace --group-limit 10