MongoDB Cool Features

Brendan W. McAdams

10gen, Inc.

Dec. 3, 2010 @ MongoSV

mongoDB

Cool Features?

- There are lots of cool features in MongoDB. We're going to discuss just a few.
  - MapReduce
  - Stored JavaScript
  - GeoSpatial Indexes
  - GridFS
- Sharding, Replica Sets and many other things *are* cool. . . but not part of this talk.

mongoDB

MongoDB MapReduce

- MongoDB's Aggregation Functionality
- Write functions in JavaScript
- Reads from *one* collection, writes to *one* collection.
- Single Threaded per mongod...
  - In a single mongod / replica set environment: No parallelization
  - In sharded environments, one map/reduce is run per shard and re-reduced to combine all results (idempotence)

MongoDB MapReduce
Output Behavior

- Before 1.7.3: MapReduce creates a temporary collection. Can specify permanent collection via ‘out’. Contents of ‘out’ are overwritten after job is finished. Temp collections cleaned up when connection closes.
- Since 1.7.3: Specify ‘outType‘ parameter.
  - ‘normal’ is current behavior.
  - ‘merge’ merges old collection and new results, clobbering any existing keys.
  - ‘reduce’ runs a reduce operation if both new and old contain the same key.

mongoDB

- Sample Data: US Treasury Bond historical Bid Curves since January 1990, to calculate an annual average for the 10 year Treasury.
  - A sample of our dataset:

code/sample_treasury.js

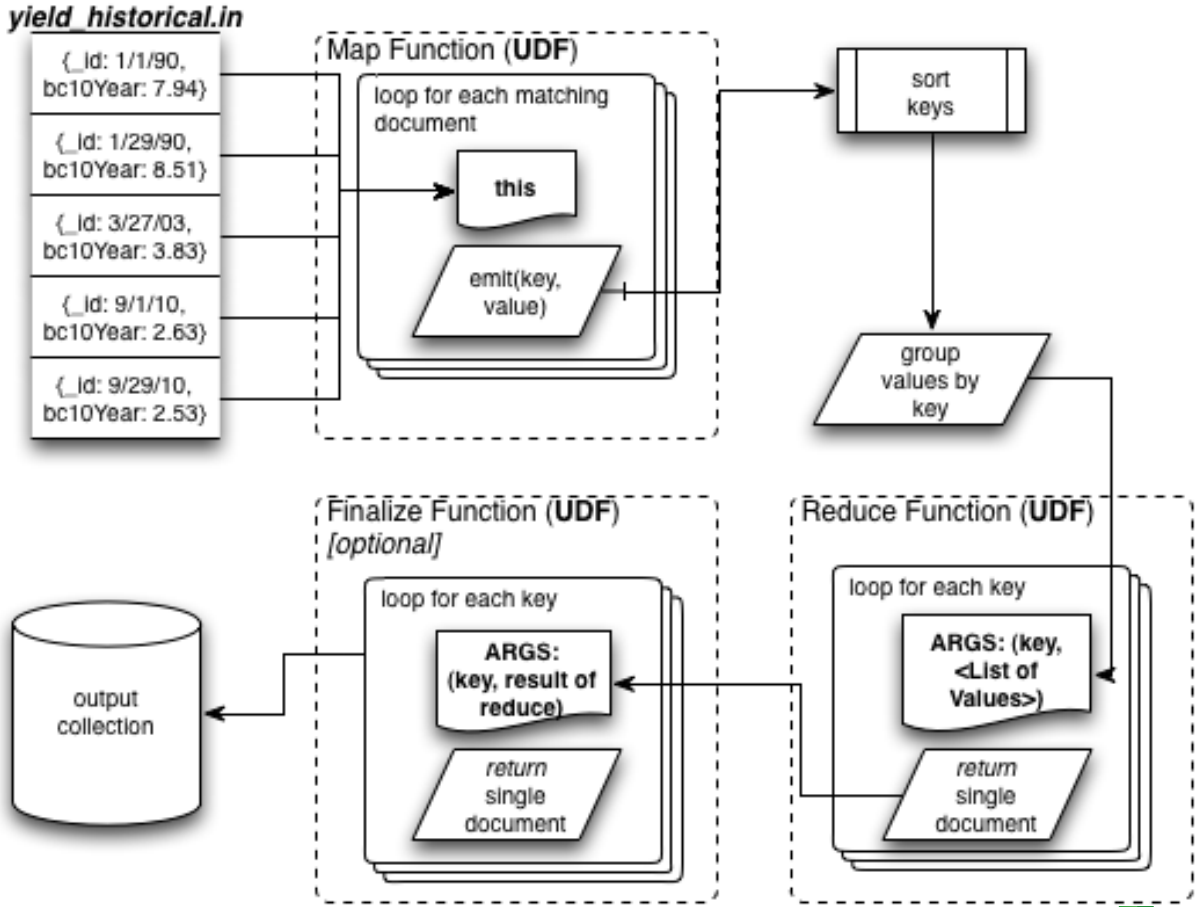
```
{ "_id" : { "$date" : 631238400000 }, "dayOfWeek" :
    "TUESDAY", "bc3Year" : 7.9, "bc5Year" : 7.87,
    "bc10Year" : 7.94, "bc20Year" : null, "bc1Month" :
    null, "bc2Year" : 7.87, "bc3Month" : 7.83, "bc30Year" :
     8, "bc1Year" : 7.81, "bc7Year" : 7.98, "bc6Month" :
    7.89 }
{ "_id" : { "$date" : 631324800000 }, "dayOfWeek" :
    "WEDNESDAY", "bc3Year" : 7.96, "bc5Year" : 7.92,
    "bc10Year" : 7.99, "bc20Year" : null, "bc1Month" :
    null, "bc2Year" : 7.94, "bc3Month" : 7.89, "bc30Year" :
     8.039999999999999, "bc1Year" : 7.85, "bc7Year" :
    8.039999999999999, "bc6Month" : 7.94 }
```

MongoDB MapReduce II
Running a MapReduce

```
{ "_id" : { "$date" : 631411200000 }, "dayOfWeek" :
    "THURSDAY", "bc3Year" : 7.93, "bc5Year" : 7.91,
    "bc10Year" : 7.98, "bc20Year" : null, "bc1Month" :
    null, "bc2Year" : 7.92, "bc3Month" : 7.84, "bc30Year" :
     8.039999999999999, "bc1Year" : 7.82, "bc7Year" : 8.02,
     "bc6Month" : 7.9 }
{ "_id" : { "$date" : 631497600000 }, "dayOfWeek" :
    "FRIDAY", "bc3Year" : 7.94, "bc5Year" : 7.92,
    "bc10Year" : 7.99, "bc20Year" : null, "bc1Month" :
    null, "bc2Year" : 7.9, "bc3Month" : 7.79, "bc30Year" :
    8.06, "bc1Year" : 7.79, "bc7Year" : 8.029999999999999,
    "bc6Month" : 7.85 }
```

MongoDB MapReduce III
Running a MapReduce

- Job Anatomy (Single Server):



mongoDB

MongoDB MapReduce IV
Running a MapReduce

- The MongoDB JavaScript mapReduce:

code/mongo_treasury_mr.js

```
function m() {
    emit( this._id.getYear(), { count: 1, sum: this.bc10Year
     })
}

function r( year, values ) {
    var n = { count: 0,  sum: 0 }
    for ( var i = 0; i < values.length; i++ ){
        n.sum += values[i].sum;
        n.count += values[i].count;
    }

    return n;
}

function f( year, value ){
    value.avg = value.sum / value.count;
```

MongoDB MapReduce V
Running a MapReduce

```
    return value.avg;
}

result = db.yield_historical.in.mapReduce( m , r, {
    finalize: f });
```

MongoDB MapReduce VI
Running a MapReduce

- Job Output:

code/mongo_mr_out.js

```
{
    "result" : "tmp.mr.mapreduce_1291414680_16",
    "timeMillis" : 524,
    "counts" : {
        "input" : 5193,
        "emit" : 5193,
        "output" : 21
    },
    "ok" : 1,
}
```

MongoDB MapReduce VII
Running a MapReduce

- A bit more verbosely:

code/mongo_mr_out_verbose.js

```
> db.yield_historical.in.mapReduce( m , r, { finalize: f });
{
    "result" : "tmp.mr.mapreduce_1291414803_17",
    "timeMillis" : 389,
    "timing" : {
        "mapTime" : NumberLong(275),
        "emitLoop" : 345,
        "total" : 389
    },
    "counts" : {
        "input" : 5193,
        "emit" : 5193,
        "output" : 21
    },
    "ok" : 1,
}
```

MongoDB MapReduce VIII
Running a MapReduce

- Read the collection for your results:

code/mongo_mr_out_read.js

```
> db.tmp.mr.mapreduce_1291414803_17.find()
{ "_id" : 90, "value" : 8.552400000000002 }
{ "_id" : 91, "value" : 7.862360000000025 }
{ "_id" : 92, "value" : 7.008844621513946 }
{ "_id" : 93, "value" : 5.866279999999999 }
{ "_id" : 94, "value" : 7.085180722891565 }
{ "_id" : 95, "value" : 6.573920000000002 }
{ "_id" : 96, "value" : 6.443531746031743 }
{ "_id" : 97, "value" : 6.353959999999992 }
{ "_id" : 98, "value" : 5.262879999999994 }
{ "_id" : 99, "value" : 5.646135458167332 }
{ "_id" : 100, "value" : 6.030278884462145 }
{ "_id" : 101, "value" : 5.020685483870969 }
{ "_id" : 102, "value" : 4.61308 }
{ "_id" : 103, "value" : 4.013879999999999 }
{ "_id" : 104, "value" : 4.271320000000004 }
{ "_id" : 105, "value" : 4.288880000000001 }
```

MongoDB MapReduce IX
Running a MapReduce

```
{ "_id" : 106, "value" : 4.794999999999955 }
{ "_id" : 107, "value" : 4.634661354581674 }
{ "_id" : 108, "value" : 3.664262948207174 }
{ "_id" : 109, "value" : 3.264120000000037 }
has more
```

- It's possible to specify a query, sort and limit as well, to limit your input.

MongoDB's Stored JavaScript I

- Each Database has a system collection, 'system.js' which can store JavaScript routines
- '_id' is set to the function name, 'value' to the function body.
- Stored Functions are unique per database and can be accessed in scope from any JavaScript (But not the raw JS Shell)
- Useful for commonly used routines in MapReduce

code/stored_func.js

```
// START: CreateStoredCheckFunc
> var _checkTime = function(date, hour, minuteStart, minuteEnd)
    {
...     var hourOk = date.getHours() == hour;
...     var minuteOk = true;
...     if (minuteStart != null && minuteEnd != null)
...         minuteOk = date.getMinutes() >= minuteStart &&
...                    date.getMinutes() <= minuteEnd;
...     else if (minuteStart != null)
```

MongoDB's Stored JavaScript II

```
...         minuteOk = date.getMinutes() == minuteStart;
...     return hourOk && minuteOk;
... }
> var test = new Date("Mon Aug 16 2010 14:25:11 GMT-0400 (EDT)")
> test
"Mon Aug 16 2010 14:25:11 GMT-0400 (EDT)"
> _checkTime(test, 9)
false
> _checkTime(test, 14)
true
> _checkTime(test, 14, 25)
true
> _checkTime(test, 14, 15, 45)
true
> db.system.js.insert({_id: "checkTime", value: _checkTime})
> db.system.js.find({_id: "checkTime"})
{ "_id" : "checkTime",
  "value" : function cf__1__f_(date, hour, minuteStart,
    minuteEnd) {
    var hourOk = date.getHours() == hour;
    var minuteOk = true;
    if (minuteStart != null && minuteEnd != null) {
```

MongoDB's Stored JavaScript III

```
        minuteOk = date.getMinutes() >= minuteStart &&
            date.getMinutes() <= minuteEnd;
    } else if (minuteStart != null) {
        minuteOk = date.getMinutes() == minuteStart;
    }
    return hourOk && minuteOk;
} }



> db.orders.find({date:
...   {$gte: midnight, $lt: tomorrow},
...   $where: function() {
...     return checkTime(this.date, 14, 0, 30);
...   }})
{
        "_id" : ObjectId("4c69f7ed94e047532497d174"),
        "product" : {
                "book" : "JavaScript: The Good Parts",
                "author" : "Douglas Crockford"
        },
        "quantity" : 1,
```

MongoDB's Stored JavaScript IV

```
        "price" : {
                "currency" : "USD",
                "msrp": 29.99,
                "amount" : 19.99,
                "tax": 1.77,
                "shipping": 3.99,
                "total": 25.75
        },
        "date" : "Mon Aug 16 2010 14:25:11 GMT-0400 (EDT)"
}
```

GeoSpatial Indexing I

- Search by Geospatial proximity with MongoDB...
- One Geoindex allowed per database
- Index can be created on an array or a subdocument
- You must be consistent across all documents (e.g. same key names or order in array)
- I loaded the publicly available GTFS data for Caltrain and BART
- Quick & Dirty Python script to create the index:

GeoSpatial Indexing II

code/geospatial_idx.py

```
import pymongo
from pymongo import Connection

if float(pymongo.version) < 1.6:
    raise Exception("ERROR: This script requires PyMongo Version
     1.6 or greater.")

connection = Connection()
db = connection['transit']
print "Indexing the Stops Data."
for row in db.stops.find():
    row['stop_geo'] = {'lat': row['stop_lat'], 'lon': row[
import pymongo
from pymongo import Connection

if float(pymongo.version) < 1.6:
    raise Exception("ERROR: This script requires PyMongo Version
     1.6 or greater.")

connection = Connection()
```

GeoSpatial Indexing III

```
db = connection['transit']
print "Indexing the Stops Data."
for row in db.stops.find():
    row['stop_geo'] = {'lat': row['stop_lat'], 'lon':
    row['stop_lon']}
    db.stops.save(row)

db.stops.ensure_index([('stop_geo', pymongo.GEO2D)])
print "Reindexed stops with Geospatial data."

print "Indexing the Shapes data"
for row in db.shapes.find():
    row['shape_pt_geo'] = {'lat': row['shape_pt_lat'], 'lon':
    row['shape_pt_lon']}
    db.shapes.save(row)

db.shapes.ensure_index([('shape_pt_geo', pymongo.GEO2D)])
print "Reindexed shapes with Geospatial data."

print "Done."
```

GeoSpatial Indexing IV

- What are the 5 nearest BART or Caltrain stops to our current location ('37.41331, -122.07098')?

code/geospatial.js

```
> db.stops.find({"stop_geo": {$near: [37.41331, -122.07098]}},
    {"stop_name": 1, "stop_desc": 1}).limit(10)
{ "_id" : ObjectId("4cf976fb0dc7bf0c1d9eeef8"), "stop_desc" :
    "600 W. Evelyn Avenue, Mountain View", "stop_name" :
    "Mountain View Caltrain" }
{ "_id" : ObjectId("4cf976fb0dc7bf0c1d9eeefb"), "stop_desc" :
    "190 Showers Drive, Mountain View", "stop_name" : "San
    Antonio Caltrain" }
{ "_id" : ObjectId("4cf976fb0dc7bf0c1d9eef04"), "stop_desc" :
    "121 W. Evelyn Avenue, Sunnyvale", "stop_name" : "Sunnyvale
    Caltrain" }
{ "_id" : ObjectId("4cf976fb0dc7bf0c1d9eeeee"), "stop_desc" :
    "101 California Avenue,Palo Alto", "stop_name" : "California
     Ave Caltrain" }
{ "_id" : ObjectId("4cf976fb0dc7bf0c1d9eeef4"), "stop_desc" :
    "137 San Zeno Way, Sunnyvale", "stop_name" : "Lawrence
    Caltrain" }
```

GeoSpatial Indexing V

```
{ "_id" : ObjectId("4cf976fb0dc7bf0c1d9eeef9"), "stop_desc" :
    "95 University Avenue,Palo Alto", "stop_name" : "Palo Alto
    Caltrain" }
{ "_id" : ObjectId("4cf976fb0dc7bf0c1d9eeef5"), "stop_desc" :
    "1120 Merrill Street,Menlo Park", "stop_name" : "Menlo Park
    Caltrain" }
{ "_id" : ObjectId("4cf976fb0dc7bf0c1d9eeee8"), "stop_desc" : "1
     Dinkelspiel Station Lane, Atherton", "stop_name" :
    "Atherton Caltrain" }
{ "_id" : ObjectId("4cf976fb0dc7bf0c1d9eef02"), "stop_desc" :
    "1001 Railroad Avenue,Santa Clara", "stop_name" : "Santa
    Clara Caltrain" }
{ "_id" : ObjectId("4cf976fb0dc7bf0c1d9eeef0"), "stop_desc" :
    "780 Stockton Avenue,San Jose", "stop_name" : "College Park
    Caltrain" }
```

- In production use at Foursquare & Wordsquared (Formerly Scrabb.ly)

GridFS: Scalable MongoDB File Storage I

- Specification for storing large files in MongoDB, supported in all official drivers as reference implementation.
- Works around 4MB BSON limit by breaking files into chunks.
- Two collections: ‘fs.files’ for metadata, ‘fs.chunks’ stores the individual file chunks.
- Sharding: Individual file chunks don’t shard but the files themselves will (e.g. File A goes on Server 1, File B goes on Server 2 but no chunks of A will be on 2)
- Experimental modules for Lighttpd and Nginx to serve static files directly from GridFS
- A Unit Test from Casbah (Scala Driver):

GridFS: Scalable MongoDB File Storage II

code/gridfs_spec.scala

```
package com.mongodb.casbah
package test

import com.mongodb.casbah.gridfs.Imports._

import java.security.MessageDigest
import java.io._

import org.specs._
import org.specs.specification.PendingUntilFixed

class GridFSSpec extends Specification with PendingUntilFixed {
  val logo_md5 = "479977b85391a88bbc1da1e9f5175239"
  val digest = MessageDigest.getInstance("MD5")


  "Casbah's GridFS Implementations" should {
    shareVariables()
    implicit val mongo = MongoConnection()("casbah_test")
    mongo.dropDatabase()
```

GridFS: Scalable MongoDB File Storage III

```
val logo = new
FileInputStream("casbah-gridfs/src/test/resources/powered_by_mon
val gridfs = GridFS(mongo)

"Correctly save a file to GridFS" in {
  gridfs must notBeNull
  logo must notBeNull

  gridfs(logo) { fh =>
    fh.filename = "powered_by_mongo.png"
    fh.contentType = "image/png"
  }

}

"Find the file in GridFS later" in {
  val file = gridfs.findOne("powered_by_mongo.png")
  file must notBeNull
  file must haveSuperClass[GridFSDBFile]
  file.md5 must beEqualTo(logo_md5)
  println(file.md5)
}
```

GridFS: Scalable MongoDB File Storage IV

```
    }
}

// vim: set ts=2 sw=2 sts=2 et:
```

- See the GridFS Spec...`http://www.mongodb.org/display/DOCS/GridFS+Specification`

Questions?

- Twitter: **@rit** | mongodb: **@mongodb** | 10gen: **@10gen**
- email: **brendan@10gen.com**
- Pressing Questions?
  - IRC - freenode.net **#mongodb**
  - MongoDB Users List - `http://groups.google.com/group/mongodb-user`
- 10gen is *hiring!* We need smart engineers in both NY and Bay Area: `http://10gen.com/jobs`
- Up Next: “MongoDB Project Roadmap” with Dwight Merriman and Eliot Horowitz. Simulcast to all rooms so stay put!

mongoDB