# Understanding MongoDB

Brendan W. McAdams

10gen, Inc.

Mar. 23, 2011 @ PGEast


mongoDB

# What Is MongoDB? I

- “Document-Oriented” Database, with feature of both Key-Value Stores & RDBMS’
  - Rich Query Interface
  - Works with JSON-like documents
  - Favors embedding data over “foreign key” relationships

# What Is MongoDB? II

- Open Source & Free: Server is licensed under A-GPL, Official language drivers under Apache 2
- Focused on native APIs for MongoDB interaction to adapt to the host language's native idioms (rather than vice versa)
  - Official Drivers for. . . C, C#, C++, Java, JavaScript, Perl, PHP, Python, Ruby, Scala, Haskell, Erlang
  - Community Supported drivers for. . . Clojure, F#, Go, Haskell, Lua, Objective C, Smalltalk and more. . .

mongoDB

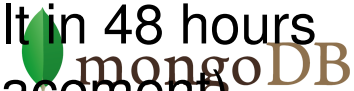
# What Is MongoDB? III

- Cursor-based query results
- ServerSide JavaScript
  - Stored JavaScript functions server-side
  - Powerful aggregation via Map/Reduce & Group Commands
  - JavaScript statements in queries (No indexes, though)
- Indexing system much like RDBMS', includes Geospatial support
- Scalable file storage with GridFS
- Data scalability with Replica Sets & Sharding

mongoDB

# But is anyone actually *using* it?!?

- MongoDB is deployed in production at companies which include. . .
  - Foursquare
  - Shutterfly
  - Bit.ly
  - Intuit
  - Wordnik
    - Over 12 billion documents in MongoDB
    - Multiple nodes with  3TB per node
    - Handling loads of 2m requests/hour, writing 50k documents/second (at peak)
  - Sourceforge
  - Etsy
  - The New York Times
  - Justin.tv
  - Github
  - Wordsquared (formerly 'Scrabb.ly' - Scrabble MMO built in 48 hours using MongoDB geospatial indexes to determine tile placement)



# Core Concepts I

- MongoDB's equivalent to "tables" are referred to as "collections", which contain "documents" (individual pieces of data)

```
> db.inventory.findOne({"title": /^The Prag/})
{
    "_id" : ObjectId("4d59b5a6cad49870530000ec"),
    "author" : "Andrew Hunt and David Thomas",
    "isbn" : "020161622X",
    "price" : {
        "discount" : 35.28,
        "msrp" : 49.99
    },
    "publicationYear" : 2000,
    "publisher" : "Addison-Wesley",
    "quantity" : NumberLong(50),
    "tags" : [
        "programming",
        "software development",
        "agile development",
        "best practices",
        "computer science"
```

# Core Concepts II

```
    ],
    "title" : "The Pragmatic Programmer: From Journeyman to
    Master"
}
```

mongoDB

# Core Concepts III

- DBs & Collections are lazy - they are created when first written to
- MongoDB's wire format/internal representation is **BSON** - Binary JSON
  - Binary optimized flavor of **JSON**; corrects several shortfalls.
  - Binary efficient string encoding (**JSON uses Base64**)
  - Supports other features such as Regular Expressions, Byte Arrays, DateTimes & Timestamps and JavaScript code blocks & functions.
  - Implemented in separate packages for official drivers
  - Createive Commons Licensed, available at `http://bsonspec.org`
- Java & Scala drivers ( represents **BSON** objects with a map-like **DBObject**; many dynamic languages (Perl, Python, etc.) use native dictionary objects.

mongoDB

# The basics of Querying I

- Find a single row with *findOne()*; returns the first document found (by natural order).
- You can find all documents matching your query with *find()*. No query means you get the entire collection back.
- Queries are specified as **BSON** documents to match against.

# The basics of Querying II

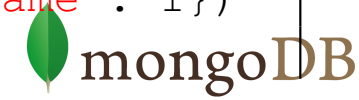
- The *find()* and *findOne()* methods can take an optional second **DBObject** specifying the fields to return.
- If you have an embedded object (for example, an address object) you can retrieve it with dot notation in the fields list (e.g. *"address.city"* retrieves just the city value).
- Use *limit()*, *skip()* and *sort()* on result objects (**DBCursor** in Java-driver land) to adjust your results. These all return a new cursor.
- *distinct()* can be used (on **DBCollection** to find all distinct values for a given key; it returns a list of values.

mongoDB

# The basics of Querying III

```
> db.routes.findOne({"route_short_name": "E"})
{
    "_id" : ObjectId("4d8a1ccbe289ae2897caf547"),
    "route_id" : "E",
    "agency_id" : "MTA NYCT",
    "route_short_name" : "E",
    "route_long_name" : "8 Avenue Local",
    "route_desc" : "Trains operate between Jamaica Center
    (Parsons/Archer), Queens, and World Trade Center,
    Manhattan, at all times.",
    "route_type" : 1,
    "route_url" :
    "http://www.mta.info/nyct/service/pdf/tecur.pdf",
    "route_color" : "0039A6",
    "route_text_color" : "FFFFFF"
}


> db.routes.find({"route_long_name": /Local$/},
...               {"route_short_name": 1, "route_long_name": 1})
```



# The basics of Querying IV

```
{ "_id" : ObjectId("4d8a1ccbe289ae2897caf539"),
    "route_short_name" : 1, "route_long_name" : "Broadway - 7
    Avenue Local" }
{ "_id" : ObjectId("4d8a1ccbe289ae2897caf53e"),
    "route_short_name" : 6, "route_long_name" : "Lexington
    Avenue Local" }
{ "_id" : ObjectId("4d8a1ccbe289ae2897caf540"),
    "route_short_name" : 7, "route_long_name" : "Flushing
    Local" }
{ "_id" : ObjectId("4d8a1ccbe289ae2897caf545"),
    "route_short_name" : "C", "route_long_name" : "8 Avenue
    Local" }
{ "_id" : ObjectId("4d8a1ccbe289ae2897caf547"),
    "route_short_name" : "E", "route_long_name" : "8 Avenue
    Local" }
{ "_id" : ObjectId("4d8a1ccbe289ae2897caf548"),
    "route_short_name" : "F", "route_long_name" : "Queens Blvd
    Express/ 6 Av Local" }
{ "_id" : ObjectId("4d8a1ccbe289ae2897caf54b"),
    "route_short_name" : "J", "route_long_name" : "Nassau St
    Local" }
```

# The basics of Querying V

```
{ "_id" : ObjectId("4d8a1ccbe289ae2897caf54c"),
    "route_short_name" : "L", "route_long_name" : "14
    St-Canarsie Local" }
{ "_id" : ObjectId("4d8a1ccbe289ae2897caf54d"),
    "route_short_name" : "M", "route_long_name" : "QNS BLVD-6th
    AVE/ Myrtle Local" }
{ "_id" : ObjectId("4d8a1ccbe289ae2897caf550"),
    "route_short_name" : "R", "route_long_name" : "Broadway
    Local" }

> db.routes.distinct("route_short_name")
 [
        1,
        2,
        3,
        4,
        5,
        6,
        7,
        "A",
        "B",
        "C",
```

# The basics of Querying VI

```
        "D",
        "E",
        "F",
        "G",
        "J",
    /*... */
]
```

# Query Operators I

- MongoDB is no mere Key-Value store. There are myriad powerful operators to enhance your MongoDB queries...
  - Conditional Operators: **$gt** (>), **$lt** (<), **$gte (>=)**, **$lte (<=)**
  - Negative Equality: **$ne** (!=)
  - Array Operators: **$in** (SQL "IN" clause...takes an array), **$nin** (Opposite of "IN"), **$all** (Requires all values in the array match), **$size** (Match the size of an array)
  - Field Defined: **$exists** (boolean argument)(Great in a schemaless world)
  - Regular Expressions (Language dependent - most drivers support it)
  - Pass Arbitrary Javascript with **$where**, boolean OR with **$or**
  - Negate any operator with **$not**
- Using a query operator requires nested objects...

# Query Operators II

```
> db.inventory.find({"price.discount": {$lt: 35, $gt: 25}},
    {"title": 1, "author": 1}).limit(2)
{ "_id" : ObjectId("4d59b6aecad4987053000173"), "author" :
    "Benjamin C. Pierce", "title" : "Basic Category Theory For
    Computer Scientists" }
{ "_id" : ObjectId("4d59b43acad498705300002b"), "author" :
    "{Brooks, Jr.}, Frederick P.", "title" : "The Mythical Man
    Month: Essays on Software Engineering" }

> db.inventory.find({"publisher": {$in: ["O'Reilly &
    Associates, Inc", "The Pragmatic Programmers, LLC"]}},
    {"title": 1, "author": 1, "publisher": 1})
{ "_id" : ObjectId("4d59b6a0cad498705300016b"), "author" :
    "Terence Parr", "publisher" : "The Pragmatic Programmers,
    LLC", "title" : "The Definitive ANTLR Reference: Building
    Domain-Specific Languages" }
{ "_id" : ObjectId("4d59b468cad4987053000046"), "author" :
    "Mike Clark", "publisher" : "The Pragmatic Programmers,
    LLC", "title" : "Advanced Rails Recipes: 84 New Ways to
    Build Stunning Rails Apps" }
```

mongoDB

# Query Operators III

```
{ "_id" : ObjectId("4d59b686cad498705300015e"), "author" :
    "Michael T. Nygard", "publisher" : "The Pragmatic
    Programmers, LLC", "title" : "Release It!: Design and
    Deploy Production-Ready Software" }
{ "_id" : ObjectId("4d59b72ccad49870530001af"), "author" :
    "Rachel Sedley and Liz Davies", "publisher" : "The
    Pragmatic Programmers, LLC", "title" : "Agile Coaching" }
{ "_id" : ObjectId("4d59b6efcad49870530001 90"), "author" :
    "Leonard Richardson and Sam Ruby", "publisher" : "O'Reilly
    & Associates, Inc", "title" : "RESTful Web Services" }
{ "_id" : ObjectId("4d59b757cad49870530001c4"), "author" :
    "Sonatype Company", "publisher" : "O'Reilly & Associates,
    Inc", "title" : "Maven: The Definitive Guide" }
```

- No syntactic sugar in Java to make it easier. . .

mongoDB

# Insert/Update/Save I

- Objects in MongoDB Collections have an "_id" field, which must be unique.
- Three ways to add/update data in MongoDB...
  - `insert()` always attempts to add a new row. If "_id" is present and contains a value already in the collection, insert fails.
  - `save()` inserts if there is no "_id" field, otherwise it tries to update the document with the specified "_id".
  - `update()` takes a query and the new values to save. By default it updates only the first document matching the query.
  - For `update()` you can specify two booleans whose default is false: *upsert*, which indicates you wish to create a new document if the query doesn't match, and *multi*, which allows updating **all** documents who match the query.

# Insert/Update/Save II

```
> db.testData.insert({"userCount": 5})
> x = db.testData.findOne({"userCount": 5})
{ "_id" : ObjectId("4c607f48150c335a4e187f41"), "userCount" : 5
    }
> x.userCount
5
> x.userCount = 20
20
> db.testData.save(x)
> db.testData.findOne({_id: x._id})
{ "_id" : ObjectId("4c607f48150c335a4e187f41"), "userCount" :
    20 }
> db.testData.update({_id: x._id}, {$inc: {"userCount": 12}})
> db.testData.findOne({_id: x._id})
{ "_id" : ObjectId("4c607f48150c335a4e187f41"), "userCount" :
    32 }
// upsert
> db.testData.update({"userCount": 5}, {"userCount": 209}, true)
> db.testData.findOne({"userCount": 209} )
{ "_id" : ObjectId("4c60800e08c3693f5962dda5"), "userCount" :
    209 }
```

# Insert/Update/Save III

# Atomic Operators I

## Atomic Operations for in-place modification without transactions

- $set to modify specific fields in a document; $inc to atomically increment (or decrement) a value

```
> db.inventory.update(
 {
 "title": "The Hitchhiker's Guide to the Galaxy: The Greatest
    Book Ever Written"
 },
 {
  $set: {"title": "The Hitchhiker's Guide to the Galaxy",
    "genre": "Humor"},
  $inc: {"qty": 50}
 })
> db.inventory.findOne({"title": "The Hitchhiker's Guide to the
    Galaxy"})
{
  "_id" : ObjectId("4c868434cad498145f000001"),
  "author" : "Douglas Adams",
  "genre" : "Humor",
  "isbn" : "0345391803",
```

# Atomic Operators II

## Atomic Operations for in-place modification without transactions

```
  "publicationYear" : 1995,
  "publisher" : "The Ballantine Publishing Group",
  "price": {
    "msrp": 15.00,
    "discount": 10.00
  },
  "qty" : 150,
  "title" : "The Hitchhiker's Guide to the Galaxy",
  "tags": [
    {"tag": "science fiction", "weight": .75},
    {"tag": "comedy", "weight": .75},
    {"tag": "humor", "weight": .75},
    {"tag": "snarky", "weight": .70},
    {"tag": "towels", "weight": .5}
  ]
}
```

- Array Operations for Atomic Modification ($push, $pull, $addtoSet (treat arrays like a set))

# Atomic Operators III

## Atomic Operations for in-place modification without transactions

```
> db.inventory.update({"title": "The Hitchhiker's Guide to the
    Galaxy"},
... {"$push": {"tags": {"tag": "exploding whales", "weight":
    .25}}})
> db.inventory.findOne({"title": "The Hitchhiker's Guide to the
    Galaxy"})
{
    "_id" : ObjectId("4c868434cad498145f000001"),
    "author" : "Douglas Adams",
    "isbn" : "0345391803",
    "price" : {
        "msrp" : 15,
        "discount" : 10
    },
    "publicationYear" : 1995,
    "publisher" : "The Ballantine Publishing Group",
    "quantity" : 150,
    "tags" : [
        {
            "tag" : "science fiction",
```

# Atomic Operators IV

## Atomic Operations for in-place modification without transactions

```
            "weight" : 0.75
        },
        {
            "tag" : "comedy",
            "weight" : 0.75
        },
        {
            "tag" : "humor",
            "weight" : 0.75
        },
        {
            "tag" : "snarky",
            "weight" : 0.7
        },
        {
            "tag" : "towels",
            "weight" : 0.5
        },
        {
            "tag" : "exploding whales",
            "weight" : 0.25
```

# Atomic Operators V

## Atomic Operations for in-place modification without transactions

```
            }
        ],
        "title" : "The Hitchhiker's Guide to the Galaxy"
}


> db.inventory.update({"title": "The Hitchhiker's Guide to the
    Galaxy"},
... {"$pull": {"tags": {"tag": "exploding whales"}}})
> db.inventory.findOne({"title": "The Hitchhiker's Guide to the
    Galaxy"})
{
        "_id" : ObjectId("4c868434cad498145f000001"),
        "author" : "Douglas Adams",
        "isbn" : "0345391803",
        "price" : {
            "msrp" : 15,
            "discount" : 10
        },
        "publicationYear" : 1995,
        "publisher" : "The Ballantine Publishing Group",
```

# Atomic Operators VI

## Atomic Operations for in-place modification without transactions

```
    "quantity" : 150,
    "tags" : [
        {
            "tag" : "science fiction",
            "weight" : 0.75
        },
        {
            "tag" : "comedy",
            "weight" : 0.75
        },
        {
            "tag" : "humor",
            "weight" : 0.75
        },
        {
            "tag" : "snarky",
            "weight" : 0.7
        },
        {
            "tag" : "towels",
            "weight" : 0.5
```

# Atomic Operators VII

## Atomic Operations for in-place modification without transactions

```
            }
        ],
        "title" : "The Hitchhiker's Guide to the Galaxy"
}


> db.inventory.update({"title": "The Hitchhiker's Guide to the
    Galaxy"},
... {"$addToSet": {"tags": {"tag": "humor", "weight": .75}}})
> db.inventory.update({"title": "The Hitchhiker's Guide to the
    Galaxy"},
... {"$addToSet": {"tags": {"tag": "humor", "weight": .75}}})
> db.inventory.update({"title": "The Hitchhiker's Guide to the
    Galaxy"},
... {"$addToSet": {"tags": {"tag": "humor", "weight": .75}}})
> db.inventory.findOne({"title": "The Hitchhiker's Guide to the
    Galaxy"})
{
        "_id" : ObjectId("4c868434cad498145f000001"),
        "author" : "Douglas Adams",
        "isbn" : "0345391803",
```

# Atomic Operators VIII

## Atomic Operations for in-place modification without transactions

```
    "price" : {
        "msrp" : 15,
        "discount" : 10
    },
    "publicationYear" : 1995,
    "publisher" : "The Ballantine Publishing Group",
    "quantity" : 150,
    "tags" : [
        {
            "tag" : "science fiction",
            "weight" : 0.75
        },
        {
            "tag" : "comedy",
            "weight" : 0.75
        },
        {
            "tag" : "humor",
            "weight" : 0.75
        },
        {
```

# Atomic Operators IX

## Atomic Operations for in-place modification without transactions

```
            "tag" : "snarky",
            "weight" : 0.7
        },
        {
            "tag" : "towels",
            "weight" : 0.5
        },
        {
            "tag" : "humor",
            "weight" : 0.75
        }
    ],
    "title" : "The Hitchhiker's Guide to the Galaxy"
}
```

# Finally, Data Scalability.

- Traditional master-slave replication
- Replica Sets (new in 1.6)
  - Replaces master-slave setup with 1-7 server clusters
  - Automatic failover and recovery
- AutoSharding (new in 1.6)
  - Horizontal scaling - partition your collections & data across as many nodes as necessary.
  - Multiple nodes can service the same shard, allowing for balancing & failover.
  - Map/Reduce runs across multiple shards, allowing concurrency.

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

# 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

# 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.

- 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:


```
{ "_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

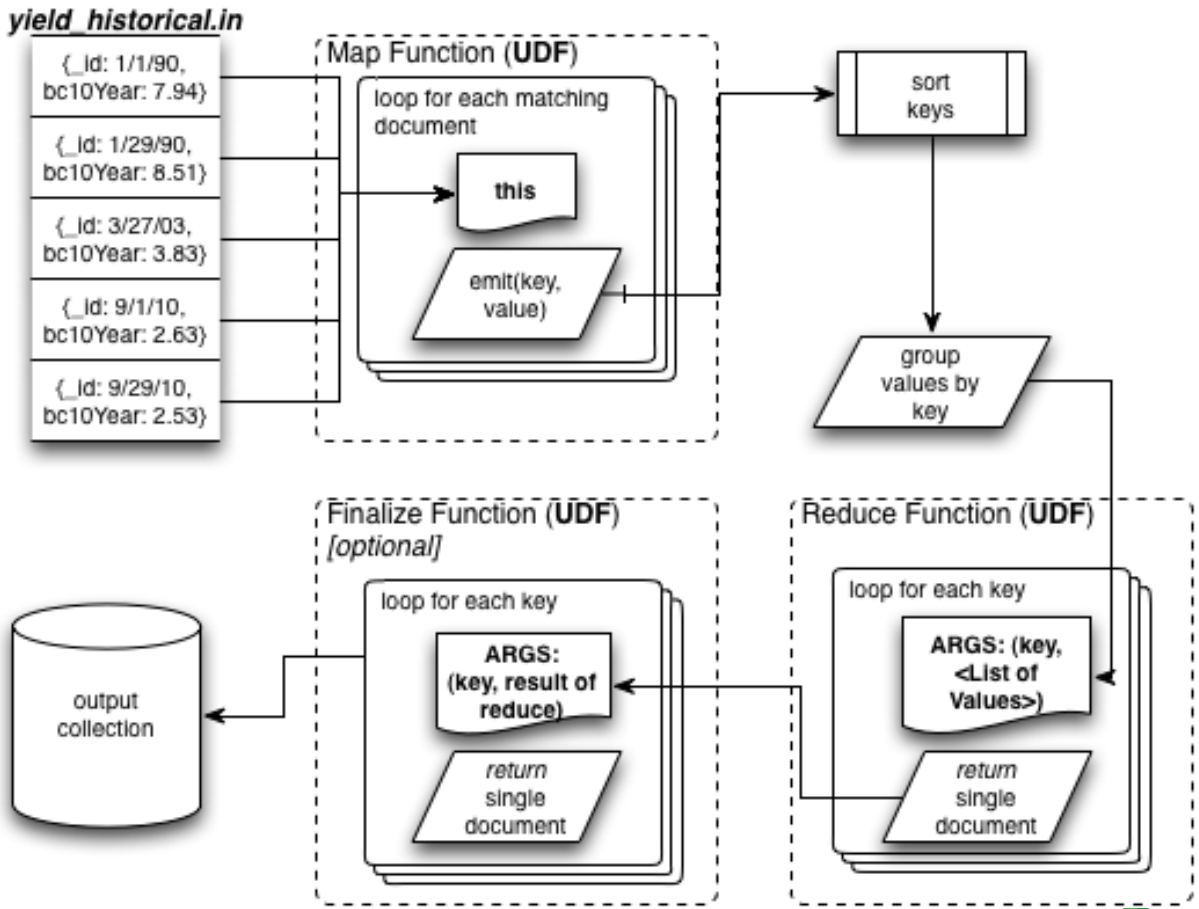
## Running a MapReduce

```
{ "_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 MapReduce IV

## Running a MapReduce

- The MongoDB JavaScript mapReduce:

```
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;
    return value.avg;
}
```

# MongoDB MapReduce V

## Running a MapReduce

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

# MongoDB MapReduce VI

## Running a MapReduce

- Job Output:

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

# MongoDB MapReduce VII

## Running a MapReduce

- Read the collection for your results:

```
> 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 }
{ "_id" : 106, "value" : 4.794999999999955 }
{ "_id" : 107, "value" : 4.634661354581674 }
{ "_id" : 108, "value" : 3.6642629482071714 }
```

# MongoDB MapReduce VIII

## Running a MapReduce

```
{ "_id" : 109, "value" : 3.2641200000000037 }
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

```
// 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)
...         minuteOk = date.getMinutes() == minuteStart;
...     return hourOk && minuteOk;
... }
```

# MongoDB's Stored JavaScript II

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

# MongoDB's Stored JavaScript III

```
        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,
        "price" : {
                "currency" : "USD",
                "msrp": 29.99,
```

# MongoDB's Stored JavaScript IV

```
                "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 NYC Subways (current as of Feb. 2011)
- Quick & Dirty Python script to create the index:

# GeoSpatial Indexing II

```
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()
db = connection['transit']
print "Indexing the Stops Data."
```

# GeoSpatial Indexing III

```
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."
```

- What are the 5 nearest BART or Caltrain stops to our current location ('40.749992, -73.991160')?

# GeoSpatial Indexing IV

```
> db.stops.find({"stop_geo": {$near: [40.749992, -73.991160]}},
    {"stop_name": 1, "stop_desc": 1}).limit(5)
{ "_id" : ObjectId("4d8a1ccbe289ae2897caf365"), "stop_name" :
    "34 St - Penn Station" }
{ "_id" : ObjectId("4d8a1ccbe289ae2897caf413"), "stop_name" :
    "34 St - Penn Station" }
{ "_id" : ObjectId("4d8a1ccbe289ae2897caf505"), "stop_name" :
    "34 St - Herald Sq" }
{ "_id" : ObjectId("4d8a1ccbe289ae2897caf451"), "stop_name" :
    "34 St - Herald Sq" }
{ "_id" : ObjectId("4d8a1ccbe289ae2897caf366"), "stop_name" :
    "28 St" }
```

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

mongoDB

# GridFS: Scalable MongoDB File Storage I

- Specification for storing large files in MongoDB, supported in all official drivers as reference implementation.
- Works around BSON document size limits 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

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

# GridFS: Scalable MongoDB File Storage III

```
    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`

mongoDB