mongoDB

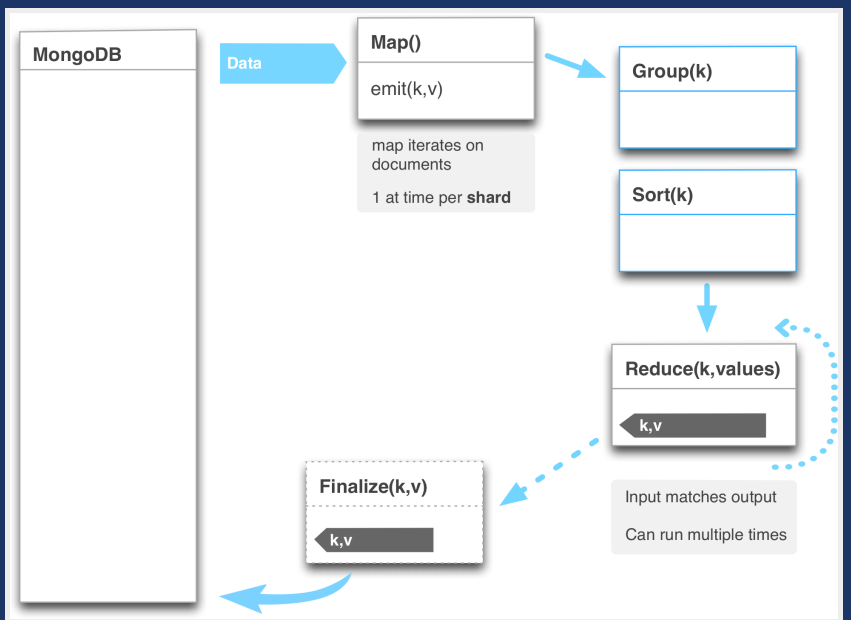
New In 2.2!

Aggregation Framework

Jeremy Mikola
@jmikola

MapReduce

MapReduce


MongoDB
Data
Map()
emit(k,v)
map iterates on documents
1 at time per shard
Group(k)
Sort(k)
Reduce(k,values)
k,v
Input matches output
Can run multiple times
Finalize(k,v)
k,v

MapReduce

- Extremely versatile, powerful
- Intended for complex data analysis
- Overkill for simple aggregation tasks
  - Averages
  - Summation
  - Grouping

MapReduce in MongoDB

- Implemented with JavaScript
  - Single-threaded
  - Difficult to debug
- Concurrency
  - Appearance of parallelism
  - Write locks

Using MapReduce for Aggregation

```
{ author: "bob", tags: ["business", "sports", "tech"] }
```

```
{ author: "jen", tags: ["politics", "tech"] }
```

```
{ author: "sue", tags: ["business"] }
```

```
{ author: "tom", tags: ["sports"] }
```

Determine the set of authors associated with each tag

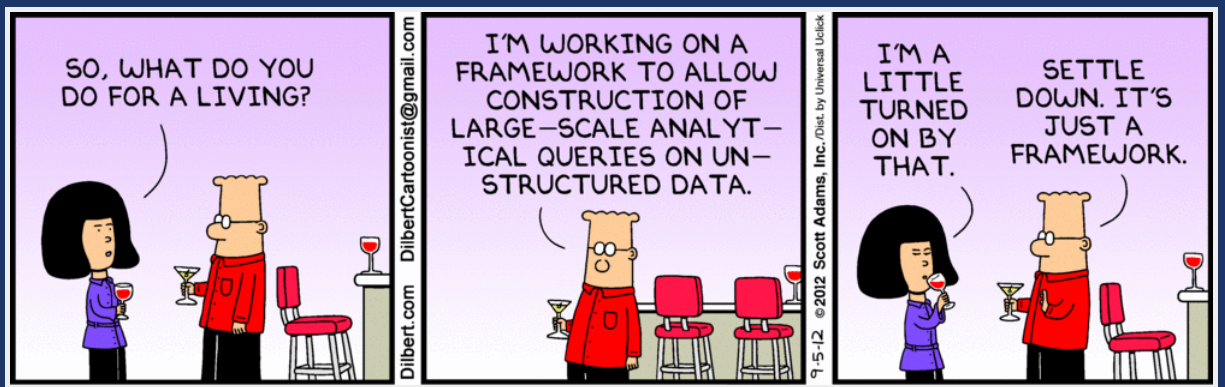
Using MapReduce for Aggregation

```
map = function() {
  for (var i = 0; i < this.tags.length; i++) {
    emit(this.tags[i], { authors: [this.author] });
  }
};

reduce = function(key, values) {
  var result = { authors: [] };
  values.forEach(function(value) {
    value.authors.forEach(function(author) {
      if (-1 == result.authors.indexOf(author)) {
        result.authors.push(author);
      }
    });
  });
  return result;
};

db.articles.mapReduce(map, reduce, { out: { inline: 1 }});
```

Using MapReduce for Aggregation

```
{
  results: [
    { _id: "business", value: { authors: ["bob", "sue"] } },
    { _id: "politics", value: { authors: ["jen"] } },
    { _id: "sports",   value: { authors: ["bob", "tom"] } },
    { _id: "tech",     value: { authors: ["bob", "jen"] } }
  ],
  timeMillis: 0,
  counts: {
    input: 4,
    emit: 7,
    reduce: 3,
    output: 4
  },
  ok: 1,
}
```

Aggregation Framework



Aggregation Framework

- Declarative (BSON, not JavaScript)
- Implemented in C++
- Flexible and functional
  - Operation pipeline
  - Computational expressions
- Plays nice with sharding

Pipeline

Pipeline

- Process a stream of documents
  - Original input is a collection
  - Final output is a result document
- Series of operators
  - Filter or transform data
  - Input/output chain

```
ps ax | grep mongod | head -n 1
```

Pipeline Operators

- `$match`
- `$project`
- `$group`
- `$unwind`
- `$sort`
- `$limit`
- `$skip`

$match

- Filter documents
- Place early in the pipeline if possible
- Uses existing **query syntax**
- No **geospatial operations** or `$where`

$match

Matching Field Values

```
{
  title: "The Great Gatsby",
  pages: 218,
  language: "English"
}
```

```
{
  title: "War and Peace",
  pages: 1440,
  language: "Russian"
}
```

```
{
  title: "Atlas Shrugged",
  pages: 1088,
  language: "English"
}
```

```
{ $match: {
  language: "Russian"
}}
```

```
{
  title: "War and Peace",
  pages: 1440,
  language: "Russian"
}
```

$match

Matching with Query Operators

```
{
  title: "The Great Gatsby",
  pages: 218,
  language: "English"
}
```

```
{
  title: "War and Peace",
  pages: 1440,
  language: "Russian"
}
```

```
{
  title: "Atlas Shrugged",
  pages: 1088,
  language: "English"
}
```

```
{ $match: {
  pages: { $gt: 1000 }
}}
```

```
{
  title: "War and Peace",
  pages: 1440,
  language: "Russian"
}
```

```
{
  title: "Atlas Shrugged",
  pages: 1088,
  language: "English"
}
```

$project

- Reshape documents
- Include, exclude or rename fields
- Inject computed fields
- Create sub-document fields

$project

Including and Excluding Fields

```
{
  _id: 375,
  title: "The Great Gatsby",
  ISBN: "9781857150193",
  available: true,
  pages: 218,
  subjects: [
    "Long Island",
    "New York",
    "1920s"
  ],
  language: "English"
}
```

```
{ $project: {
  _id: 0,
  title: 1,
  language: 1
}}
```

```
{
  title: "The Great Gatsby",
  language: "English"
}
```

$project

Renaming and Computing Fields

```
{
  _id: 375,
  title: "The Great Gatsby",
  ISBN: "9781857150193",
  available: true,
  pages: 218,
  subjects: [
    "Long Island",
    "New York",
    "1920s"
  ],
  language: "English"
}
```

```
{ $project: {
  upperTitle: {
    $toUpper: "$title"
  },
  lang: "$language"
}}
```

```
{
  _id: 375,
  upperTitle: "THE GREAT GATSBY",
  lang: "English"
}
```

$project

Creating Sub-document Fields

```
{
  _id: 375,
  title: "The Great Gatsby",
  ISBN: "9781857150193",
  available: true,
  pages: 218,
  subjects: [
    "Long Island",
    "New York",
    "1920s"
  ],
  language: "English"
}
```

```
{ $project: {
  title: 1,
  stats: {
    pages: "$pages",
    language: "$language",
  }
}}
```

```
{
  _id: 375,
  title: "The Great Gatsby",
  stats: {
    pages: 218,
    language: "English"
  }
}
```

$group

- Group documents by an ID
  - Field path reference
  - Object with multiple references
  - Constant value
- Other output fields are computed
  - `$max`, `$min`, `$avg`, `$sum`
  - `$addToSet`, `$push`
  - `$first`, `$last`
- Processes all data in memory

$group

Calculating an Average

```
{
  title: "The Great Gatsby",
  pages: 218,
  language: "English"
}
```

```
{
  title: "War and Peace",
  pages: 1440,
  language: "Russian"
}
```

```
{
  title: "Atlas Shrugged",
  pages: 1088,
  language: "English"
}
```

```
{ $group: {
  _id: "$language",
  avgPages: { $avg: "$pages" }
}}
```

```
{
  _id: "Russian",
  avgPages: 1440
}
```

```
{
  _id: "English",
  avgPages: 653
}
```

$group

Summating Fields and Counting

```
{
  title: "The Great Gatsby",
  pages: 218,
  language: "English"
}
```

```
{
  title: "War and Peace",
  pages: 1440,
  language: "Russian"
}
```

```
{
  title: "Atlas Shrugged",
  pages: 1088,
  language: "English"
}
```

```
{ $group: {
  _id: "$language",
  numTitles: { $sum: 1 },
  sumPages: { $sum: "$pages" }
}}
```

```
{
  _id: "Russian",
  numTitles: 1,
  sumPages: 1440
}
```

```
{
  _id: "English",
  numTitles: 2,
  sumPages: 1306
}
```

$group

Collecting Distinct Values

```
{
  title: "The Great Gatsby",
  pages: 218,
  language: "English"
}
```

```
{
  title: "War and Peace",
  pages: 1440,
  language: "Russian"
}
```

```
{
  title: "Atlas Shrugged",
  pages: 1088,
  language: "English"
}
```

```
{ $group: {
  _id: "$language",
  titles: { $addToSet: "$title" }
}}
```

```
{
  _id: "Russian",
  titles: [ "War and Peace" ]
}
```

```
{
  _id: "English",
  titles: [
    "Atlas Shrugged",
    "The Great Gatsby"
  ]
}
```

$unwind

- Operate on an array field
- Yield documents for each array value
  - Nothing for absent or empty fields
  - Error for non-array fields
- Complements `$group`

$unwind

Yielding Multiple Documents from One

```
{
  title: "The Great Gatsby",
  ISBN: "9781857150193",
  subjects: [
    "Long Island",
    "New York",
    "1920s"
  ]
}
```

▶

```
{ $unwind: "$subjects" }
```

▼

```
{
  title: "The Great Gatsby",
  ISBN: "9781857150193",
  subjects: "Long Island"
}
```

```
{
  title: "The Great Gatsby",
  ISBN: "9781857150193",
  subjects: "New York"
}
```

```
{
  title: "The Great Gatsby",
  ISBN: "9781857150193",
  subjects: "1920s"
}
```

$sort

- Sort documents by one or more fields
- Uses familiar **cursor format**
- Waits for earlier pipeline operator to return
- In-memory unless early and indexed

$sort

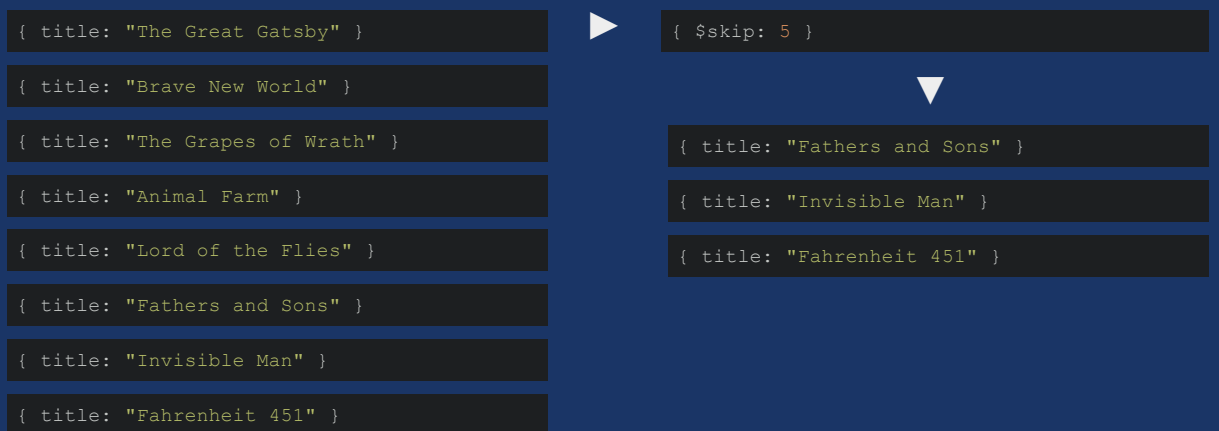
Sort All Documents in the Pipeline

$limit

Limit Documents through the Pipeline

$skip

Skip Over Documents in the Pipeline



Extending the Framework

- Adding new pipeline operators, expressions
- `$out` and `$tee` for output control
  - **https://jira.mongodb.org/browse/SERVER-3253**

Expressions

Expressions

- Return computed values
- Used with `$project` and `$group`
- May be nested

Boolean Operators

- Input array of one or more values
  - `$and`, `$or`
  - Short-circuit logic
- Inverse values with `$not`
- BSON standard for converting non-booleans
  - `null`, `undefined`, zero values → `false`
  - Non-zero values, strings, dates, objects → `true`

Comparison Operators

- Compare numbers, strings, and dates
- Input array with two operands
  - `$cmp`, `$eq`, `$ne`
  - `$gt`, `$gte`, `$lt`, `$lte`

Arithmetic Operators

- Input array of one or more numbers
  - `$add`, `$multiply`
- Input array of two operands
  - `$subtract`, `$divide`, `$mod`

String Operators

- `$strcasecmp` case-insensitive comparison
  - `$cmp` is case-sensitive
- `$toLower` and `$toUpper` case change
- `$substr` for sub-string extraction
- Not encoding aware
- Assumes Latin alphabet

Date Operators

- Extract values from date objects
  - `$dayOfYear`, `$dayOfMonth`, `$dayOfWeek`
  - `$year`, `$month`, `$week`
  - `$hour`, `$minute`, `$second`

Conditional Operators

- `$cond` ternary operator
- `$ifNull`

Usage

Usage

- `collection.aggregate()` method
  - Mongo shell
  - Most drivers
- `aggregate` database command
- Result limited by BSON document size

Collection Method

```
db.books.aggregate([
  { $project: { language: 1 }},
  { $group: { _id: "$language", numTitles: { $sum: 1 }}}
])
```

```
{
  result: [
    { _id: "Russian", numTitles: 1 },
    { _id: "English", numTitles: 2 }
  ],
  ok: 1
}
```

Database Command

```
db.runCommand({
  aggregate: "books",
  pipeline: [
    { $project: { language: 1 }},
    { $group: { _id: "$language", numTitles: { $sum: 1 }}}
  ]
})
```

```
{
  result: [
    { _id: "Russian", numTitles: 1 },
    { _id: "English", numTitles: 2 }
  ],
  ok: 1
}
```

Optimization

Early Filtering

```
[
  { $match:   { /* match criteria         */ }},
  { $sort:    { /* sort matched documents */ }},
  { $limit:   { /* limit document stream  */ }},
  { $group:   { /* group by a field       */ }},
  { $sort:    { /* sort by computed value */ }},
  { $project: { /* reshape result         */ }}
]
```

Early Filtering

```
db.collection
  .find({ /* match criteria         */ })
  .sort({ /* sort matched documents */ })
  .limit( /* limit document stream  */  )
```

```
[
  { $group:   { /* group by a field       */ }},
  { $sort:    { /* sort by computed value */ }},
  { $project: { /* reshape result         */ }}
]
```

Memory Usage

- `$group` and `$sort` entirely in-memory
- Operation memory usage limits
  - Warning at >5%
  - Error at >10%

Sharding

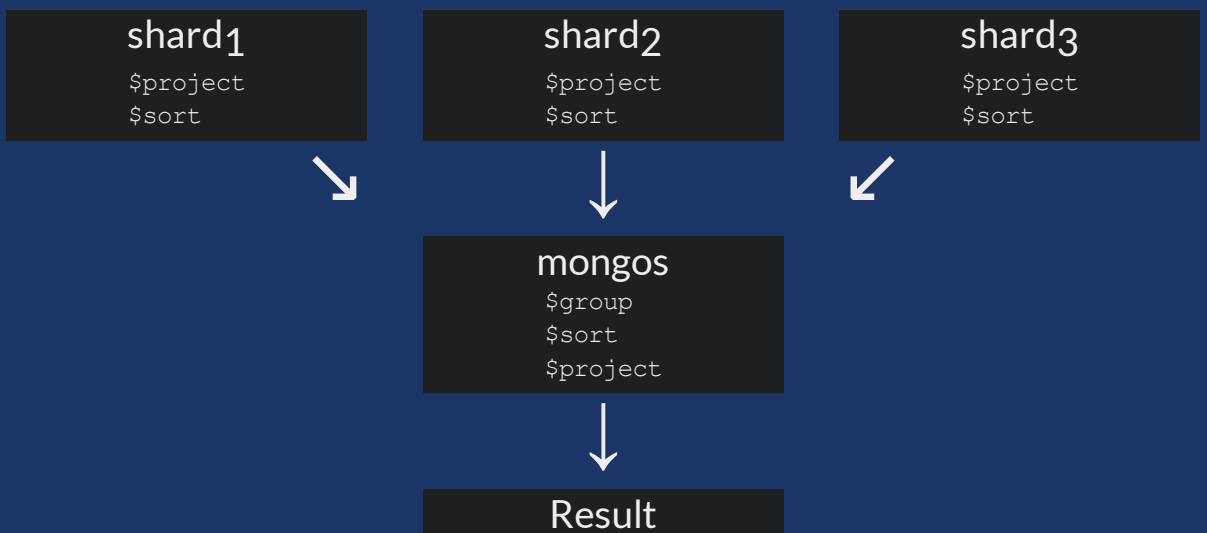
Sharding

- Split the pipeline at first `$group` or `$sort`
  - Shards execute pipeline before that point
  - **mongos** merges results and continues
- Early `$match` may excuse shards
- CPU and memory implications for **mongos**

Sharding

```
[
  { $project: { /* select fields     */ }},
  { $sort:    { /* sort by timestamp */ }},
  { $group:   { /* group by minute   */ }},
  { $sort:    { /* sort by timestamp */ }},
  { $project: { /* reshape result    */ }}
]
```

Sharding


shard1
$project
$sort
shard2
$project
$sort
shard3
$project
$sort
mongos
$group
$sort
$project
Result

Demo Time

Plotting the EUR/GDP exchange rate

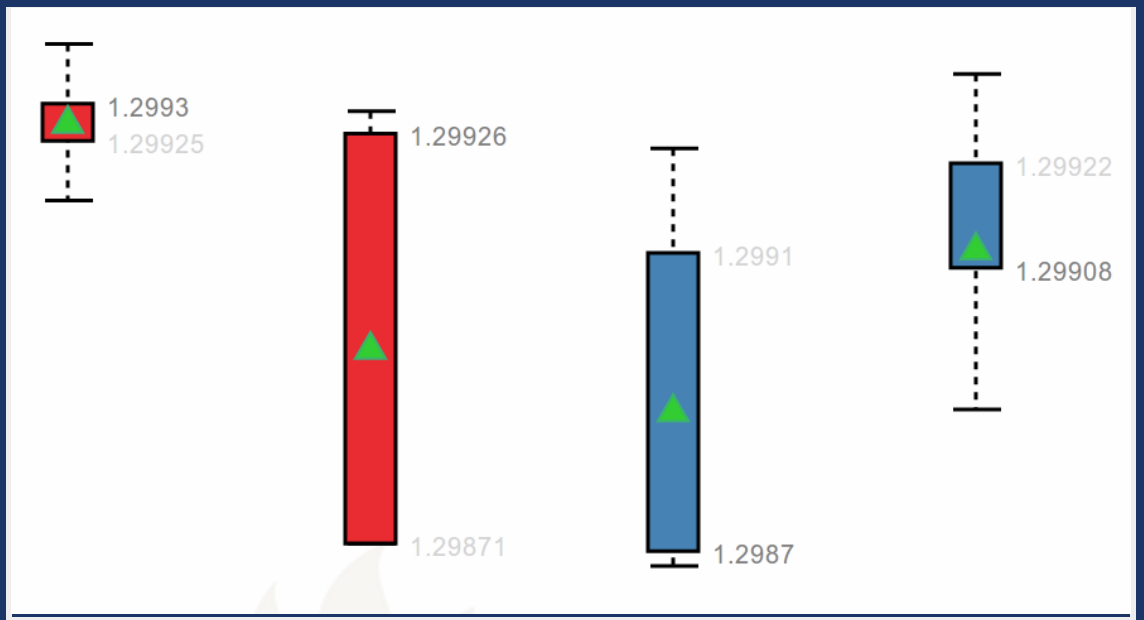
Input Data

```
{
  bid: 1.30019,
  ts: ISODate("2012-02-16T12:48:00Z")
}
```

```
{
  bid: 1.3002,
  ts: ISODate("2012-02-16T12:48:01Z")
}
```

```
{
  bid: 1.30021,
  ts: ISODate("2012-02-16T12:48:02Z")
}
```

Desired Output



Output Format

```
{
  _id: ISODate("2012-02-16T12:57:00Z"),
  bid: {
    open: 1.29994,
    close: 1.29997,
    high: 1.3001,
    low: 1.29992,
    avg: 1.2999995161290314
  }
}
```

Thanks!

- mongodb.org/downloads
- docs.mongodb.org/manual/aggregation
- github.com/rozza/demos

Questions?

Photo Credits

- http://dilbert.com/strips/comic/2012-09-05