MapReduce in MongoDB

In this post, we will take a look at performing MapReduce operations on JSON documents present in MongoDB. We will generate dummy data using ***dummy-json,*** a node package and we will use ***Mongojs*** another node package to run MapReduce jobs on that data from our Node application.

For a quick sneak peak, take a look at this runnable (click on the ***run*** button).

Contents

You can find the complete code here.

What is MongoDB?

MongoDB is a NoSQL database. Unlike MySQL or MSSQL or Oracle DBs, here database have collections instead of tables. We have documents in collections insteads of rows in a table. And best of all, all the

documents are stored as JSON. You can know more about MongoDB here.

You can install mongoDB locally from here.

If you have never worked with MongoDB before, you can remember the following commands to navigate around and perform basic operation

Command	Result
`mongod`	will start the MongoDB service
`mongo`	will step you inside the MongoDB shell (*when run in a new terminal while Mongod is running*)
`show dbs`	will show the list of databases
`use <<database name>>`	will step you inside the database
`show collections`	will show the list of collections once you are inside the database
`db.collectionName.find()`	will show all the documents in that collection
`db.collectionName.findOne()`	will show the first document

- Backbonejs
- Responsive Web Design

Projects

- Rotten Tomatoes jQuery Rest Client
- blueimp-file-upload-expressjs - A Node Package
- lazyboy - Command line laziness redefined
- diskDB - A Light Weight Disk based JSON Database - A Node Package
- My Slush Generators
- My Yeoman Generators
- My Web Components
- My Bower Packages

Series

- Ionic, Twilio and Node

`db.collectionName.find().pretty()`	will pretty print the JSON data in console
`db.collectionName.insert({key : value})`	will insert a new record
`db.collectionName.update({condition : value}, {$set : {key:value}}, {upsert : true})`	will update a record with the given condition & sets the required value. If upsert is true a new document will be created if no documents with matching condition are found
`db.collectionName.remove({})`	will remove all the documents in that collection
`db.collectionName.remove({key : value})`	will remove the documents matching the condition

You can learn more about MongoDB here.

What is MapReduce?

It is very essential that you get an understanding as how a MapReduce job works. Without this clarity, you may not really achieve the output you are expecting while running these jobs.

From Mongodb.org

Scheduler – A Reminder App
- Node Webkit Apps
- Raspberry Pi

Guest Posts

- Building a Chat App with node-webkit, Firebase, and AngularJS
- Creating a Sentiment Analysis Application Using Node.js
- Implementing pagination with MongoDB, Express.js & Slush
- Creating a Firebase Powered End to End Ionic Application
- Getting started with MongoDB and Mongoose
- Yeoman, Mongoose and MongoDB

> Map-reduce is a data processing paradigm for condensing large volumes of data into useful aggregated results. For map-reduce operations, MongoDB provides the mapReduce database command.

In very simple terms, the mapReduce command takes 2 primary inputs, the mapper function and the reducer function.

A Mapper will start off by reading a collection of data and building a Map with only the required fields we wish to process and group them into one array based on the key. And then this key value pair is fed into a Reducer, which will process the values.

Ex: Let's say that we have the following data

```
sample data from DB
[
    {
        name: foo,
        price: 9
    },
    {
        name: foo,
        price: 12
    },
    {
        name: bar,
        price: 8
    },
    {
        name: baz,
        price: 3
    }
    {
        price: 5
    }
]
```

And we want to count the price for all the items with same name. We will run this data through a Mapper and then a Reducer to achieve the result.

When we ask a Mapper to process the above data without any conditions, it will generate the following result

Key	Value
foo	[9,12]
bar	[8]
baz	[3,5]

That is, it has grouped all the data together which have a similar key, in our case a name. Then these results will be sent to the Reducer.

Now, in the reducer, we get the first row from the above table. We will iterate through all the values and add them up. This will be the sum for first row. Next, the reducer will receive the second and it will do the same thing, till all the rows are completed.

The final output would be

Name	Total
foo	21
bar	8

WordPress & Angular Material – An Offline Viewer
- Developing a MEAN app with Angular 2.0
- Pushbots and Cordova – Easy Push Notifications for your app
- Getting Started with Client Side Storage

Categories

- Architecture (8)
- CSS (9)
  - CSS3 (6)
  - LESS (3)
  - SASS (1)
- Database (17)
  - Disk DB (6)
  - Firebase (4)
  - MongoDB (8)
    - MongoLab (3)
    - Monk (1)
- Deployments (4)
  - Heroku (2)

baz	8

So now you can understand why a Mapper is called a Mapper (***because, it will create a map of data***) & why a Reducer is called a Reducer (***because it will reduce the data that the mapper has generated to a more simplified form***).

If you run a couple of examples, you will get an idea as how this works. You can read more about MongoDB MapReduce here.

Set Up a Project

As you have seen earlier, we can run queries directly in the mongo shell and see the output. But. for these examples, to keep things more ***tutorial-ish***, We will build a node project and then run the commands.

Mongojs

We will be using ***mongojs*** (a ***node package for interacting with MongoDB***),to write our MapReduce commands and execute them. You can run the same code in the mongo shell directly and see the same results. You can read more about mongojs here.

Dummy-json

We will use ***dummy-json*** (***a Node utility that allows you to generate random JSON data using Handlebars templates***) to set up a few thousand sample JSON

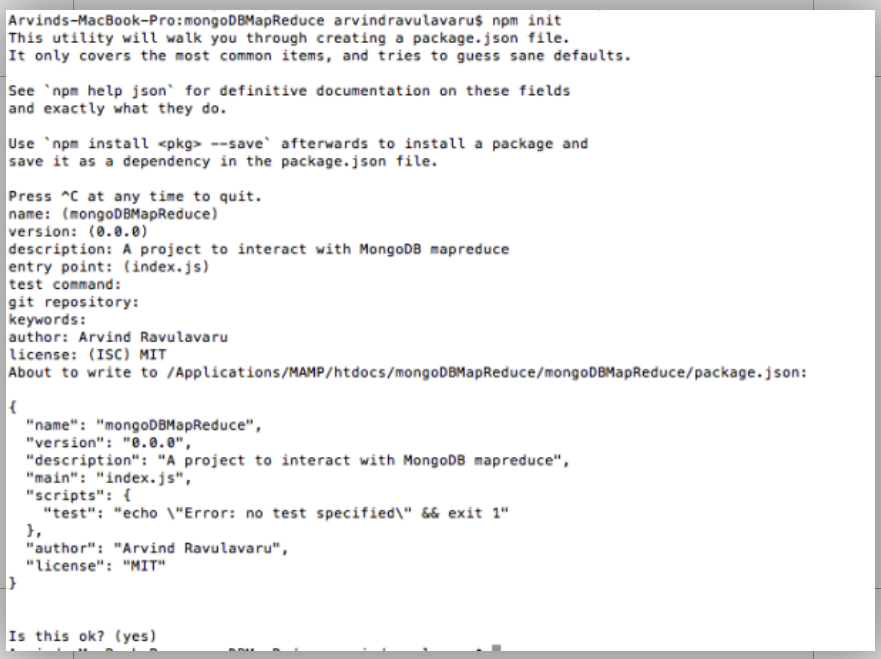
documents. You can find more information on dummy-json here. Then we will run MapReduce commands on top of it to generate some meaningful results.

So lets get started.

First you need Node js to be installed. You can find details here, Next, Create a new folder named ***mongoDBMapReduce.*** Then open a terminal/prompt here. Now we will create a ***package.json*** to store our project details. Run,

```
npm init
```

and fill it as (***or whatever you like***)



```
Arvinds-MacBook-Pro:mongoDBMapReduce arvindravulavaru$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sane defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg> --save` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (mongoDBMapReduce)
version: (0.0.0)
description: A project to interact with MongoDB mapreduce
entry point: (index.js)
test command:
git repository:
keywords:
author: Arvind Ravulavaru
license: (ISC) MIT
About to write to /Applications/MAMP/htdocs/mongoDBMapReduce/mongoDBMapReduce/package.json:

{
  "name": "mongoDBMapReduce",
  "version": "0.0.0",
  "description": "A project to interact with MongoDB mapreduce",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Arvind Ravulavaru",
  "license": "MIT"
}


Is this ok? (yes)
```

Next, we will add the project dependencies. Run

```
npm i mongojs --save-dev
```

```
npm i dummy-json --save-dev
```

This will take care of installing dependencies and adding them to our *package.json*.

Generate Dummy Data

Next, we are going to generate dummy data using the dummy-json module. Create a new file named *dataGen.js* at the root of the project. We will keep the data generation logic in a separate file. In future, if you need to add more data, you can run this file.

Copy the below contents to *dataGen.js*

dataGen.js JavaScript

```
var mongojs = require('mongojs');
var db = mongojs('mapReduceDB', ['sourceData']);
var fs = require('fs');
var dummyjson = require('dummy-json');

var helpers = {
  gender: function() {
    return ""+ Math.random() > 0.5 ? 'male' : 'female';
  },
  dob : function() {
    var start = new Date(1900, 0, 1),
        end = new Date();
        return new Date(start.getTime() + Math.random() *
    },
  hobbies : function () {
    var hobbysList = [];
    hobbysList[0] = [];
    hobbysList[0][0] = ["Acrobatics", "Meditation", "Music
    hobbysList[0][1] = ["Acrobatics", "Photography", "Papi
    hobbysList[0][2] = [ "Papier-Mache"];
    return hobbysList[0][Math.floor(Math.random() * hobby
  }
};

console.log("Begin Parsing >>");

var template = fs.readFileSync('template.hbs', {encoding:
var result = dummyjson.parse(template, {helpers: helpers})
```


Archives

Inform me about new posts

```
29
30 console.log("Begin Database Insert >>");
31
32 db.sourceData.remove(function (argument) {
33     console.log("DB Cleanup Completd");
34 });
35
36 db.sourceData.insert(JSON.parse(result), function (err, do
37     console.log("DB Insert Completed");
38 });
```

Lines 1 to 4, we include all the required packages.

Line 2, we create a new database named *mapReduceDB.* Inside that, we create a new collection named *sourceData,* if either of them do not exist already.

Lines 6 to 23 is a Handlebar helper. You can know more about that on the dummy-json page

Lines 27, 28, we read a *schema.hbs* file (which we will create in a moment) & the parse it to generate the sample JSON.

Line 32, we clean up the old data before dumping new data, Comment this out, incase you want to append to the existing collection

Line 36, insert the generated data to the DB

Next, create a new file named *schema.hbs* at the root of the project. This will consist of the schema that will constitute one JSON document. Paste the below contents into it

```
schema.hbs
```

```
1  [
2      {{#repeat 9999}}
3      {
4        "id": {{index}},
5        "name": "{{firstName}} {{lastName}}",
6        "email": "{{email}}",
7        "work": "{{company}}",
8        "dob" : "{{dob}}",
9        "age": {{number 1 99}},
10       "gender" : "{{gender}}",
11       "salary" : {{number 999 99999}},
12       "hobbies" : "{{hobbies}}"
13     }
14     {{/repeat}}
15 ]
```

Do notice on **line 2**, we are going to generate 9999 documents. Thats is it, we are all set to generate some data.

Open a *new* terminal/prompt and run

```
mongod
```

This will start the MongoDB service. Now back to our other terminal/prompt, run

```
node dataGen.js
```

The result should be

```
Arvinds-MacBook-Pro:mongoDBMapReduce arvindravulavaru$ node dataGen.js
Begin Parsing >>
Begin Database Insert >>
DB Cleanup Completd
DB Insert Completed
```

Kill the node program by pressing ctrl + c.

To verify, you can open up a new terminal/prompt, run `mongo` command and check the data. You should see

```
> db.sourceData.findOne()
{
        "id" : 0,
        "name" : "Leanne Flinn",
        "email" : "leanne.flinn@unilogic.com",
        "work" : "Unilogic",
        "dob" : "Sun Mar 14 1909 12:45:53 GMT+0530 (IST)",
        "age" : 27,
        "gender" : "male",
        "salary" : 16660,
        "hobbies" : "Acrobatics,Photography,Papier-Mache",
        "_id" : ObjectId("5357923a5973be730d5a0833")
}
> db.sourceData.count()
9999
>
```

Make sense of the data

Okay, we have dumped in 9999 documents of user data. Let try and make sense of it.

Example 1 : Get the count of Males and Females

First, create a new file named *example1.js* at the root of the project. We will write a MapReduce job, to fetch the count of Males and Females.

Mapper Logic

The only thing we are expecting from Mapper is that, it will extract the gender as key and value as 1.
One because, every user is either a male or a female.
So the output of the mapper will be

Key	Value
Male	[1,1,1,1,1,1,1,1.....]

Female	[1,1,1,1,1,1,1,1,1,....]

Reducer Logic

In reducer, we get the above 2 rows. All we need to do is sum up all the values for one row, that will result in the sum of that gender. And the final output of Reducer would be

Key	Value
Male	5031
Female	4968

Code

Now, lets write some code to achieve this. In *example1.js*, first we will require all our dependencies.

example1.js JavaScript

```
1 var mongojs = require('mongojs');
2 var db = mongojs('mapReduceDB', ['sourceData', 'example1_re
```

Do notice on **Line 2**, the first argument is the name of the database and second argument is an Array of collection that we are going to query. *example1_results* is the collection which we are going to generate with our result.

Next, Lets add the mapper and reducer functions

exampl1.js

```
1 var mapper = function () {
2     emit(this.gender, 1);
```

```
3 };
4
5 var reducer = function(gender, count){
6     return Array.sum(count);
7 };
```

On **line 2**, `this` will be populated with the current document. And `this.gender` will be either a male or a female, which would be our key. And `emit()` will push the data to a temporary hash table, that will store the mapper results.

On **line 5**, we simply add up add all the values for a gender.

Finally, add the logic to execute the mapReduce

```
mapReduce command
1  db.sourceData.mapReduce(
2      mapper,
3      reducer,
4      {
5          out : "example1_results"
6      }
7   );
8
9   db.example1_results.find(function (err, docs) {
10     if(err) console.log(err);
11     console.log(docs);
12  });
```

On **line 5**, we set the output collection name and on **line 9**, we will fetch the results from *example1_results* collection and display it.

Back to terminal/prompt and run

```
node example1.js
```

and the result will be

```
Arvinds-MacBook-Pro:mongoDBMapReduce arvindravulavaru$ node example1.js
[ { _id: 'female', value: 4968 }, { _id: 'male', value: 5031 } ]
```

My count may not match with yours, but the sum of males and females should be 9999 (*Duh!*).

Mongo Shell code

If you want to run the above in mongo shell, you can do by pasting the following into the terminal/prompt

mongo shell code

```
mapper = function () {
    emit(this.gender, 1);
};

reducer = function(gender, count){
    return Array.sum(count);
};

db.sourceData.mapReduce(
    mapper,
    reducer,
    {
        out : "example1_results"
    }
 );

 db.example1_results.find()
```

And you should see

```
Arvinds-MacBook-Pro:mongoDBMapReduce arvindravulavaru$ mongo
MongoDB shell version: 2.4.9
connecting to: test
Server has startup warnings:
Wed Apr 23 15:42:30.368 [initandlisten]
Wed Apr 23 15:42:30.368 [initandlisten] ** WARNING: soft rlimits too low. Number of files is 256, should be at least 1000
> use mapReduceDB
switched to db mapReduceDB
> mapper = function () {
... emit(this.gender, 1);
... };
function () {
emit(this.gender, 1);
}
> reducer = function(gender, count){
...  return Array.sum(count);
... };
function (gender, count){
 return Array.sum(count);
}
> db.sourceData.mapReduce(
... mapper,
... reducer,
... {
... out : "example1_results"
... }
...  );
{
        "result" : "example1_results",
        "timeMillis" : 121,
        "counts" : {
                "input" : 9999,
                "emit" : 9999,
                "reduce" : 200,
                "output" : 2
        },
        "ok" : 1,
}
> db.example1_results.find()
{ "_id" : "female", "value" : 4968 }
{ "_id" : "male", "value" : 5031 }
>
```

Simple right?

Example 2 : Get the Eldest and Youngest Person in each gender

For this example, create a new file named *example2.js* at the root of the project. Here, we will group all the users based on gender & pull out the eldest and youngest in each gender. A bit more complex than the earlier example.

Mapper Logic

In mapper, we will return the gender as key and we will return an object as value. The object will hold the user's age and user's name. Age will be used for calculation where as the name is only for display purposes.

Key	Value

Male	[{age : 9, name : John}, {}, {} ,{}...]
Female	[{age : 19, name : Rita}, {}, {} ,{}...]

Reducer Logic

Our reducer will be a bit more complex than the last example. Here we will perform a check on all the ages corresponding to a gender and sort the based on eldest or youngest. And the final result should look something like

Key	Value
Male	{'min':{'name':'Haydee Milligan','age':1},'max': {'name':'Darrell Sprowl','age':99}}
Female	{'min':{'name':'Cory Hollis','age':1},'max': {'name':'Shea Mercer','age':99}}

Code

Now, open example2.js and paste the below code.

example2.js

```
1  var mongojs = require('mongojs');
2  var db = mongojs('mapReduceDB', ['sourceData', 'example2_r
3
4
5  var mapper = function () {
6      var x = {age : this.age, name : this.name};
7      emit(this.gender, {min : x , max : x});
8  };
9
10
11 var reducer = function(key, values){
12     var res = values[0];
13     for (var i = 1; i < values.length; i++) {
14         if(values[i].min.age < res.min.age)
```

```
15              res.min = {name : values[i].min.name, age : va
16          if (values[i].max.age > res.max.age)
17             res.max = {name : values[i].max.name, age : val
18      };
19      return res;
20  };
21
22
23  db.sourceData.mapReduce(
24      mapper,
25      reducer,
26      {
27          out : "example2_results"
28      }
29   );
30
31   db.example2_results.find(function (err, docs) {
32      if(err) console.log(err);
33      console.log(JSON.stringify(docs));
34   });
```

On **line 6**, we build an object and send it as part of the value. **Lines 13 to 18**, we iterate through all the objects and check if the current value object's age is greater than the previous or less and update the res.max value. And similarly, the min value. FInally on **line 27**, we push the result set into a new collection named *example2_results.*

To run this example, back to terminal/prompt and run

```
node example2.js
```

And you should see something like

```
Arvinds-MacBook-Pro:mongoDBMapReduce arvindravulavaru$ node example2.js
[ { _id: 'female', value: { min: [Object], max: [Object] } },
  { _id: 'male', value: { min: [Object], max: [Object] } } ]
```

Since our dataset is huge and there is a high chance that all the numbers from 1 to 99 are used. You can change this in *schema.hbs* **line no 9**. Then re run the

dataGen.js. Now, you can run the above example and check the values.

Example 3 : Count the number of users in each hobby

In our final example, we will see how many users have similar hobbies. For that, lets first create a new file named ***example3.js*** at the root of the project. Data for one user would be


```
{
        "id" : 0,
        "name" : "Leanne Flinn",
        "email" : "leanne.flinn@unilogic.com",
        "work" : "Unilogic",
        "dob" : "Sun Mar 14 1909 12:45:53 GMT+0530 (IST)",
        "age" : 27,
        "gender" : "male",
        "salary" : 16660,
        "hobbies" : "Acrobatics,Photography,Papier-Mache",
        "_id" : ObjectId("5357923a5973be730d5a0833")
}
```


As you can see, every user has a list of hobbies separated by comma. We will find out how many users have ***Acrobatics*** as a hobby and so on.

Mapper Logic

Our mapper is a bit complex for this scenario. We will emit a new key value pair for each hobby of a user. This way, we will fire 1 count for each hobby per user. By the end of the mapper, we will end up with something like

Key	Value
Acrobatics	[1,1,1,1,1,1,....]

Meditation	[1,1,1,1,1,1,....]
Music	[1,1,1,1,1,1,....]
Photography	[1,1,1,1,1,1,....]
Papier-Mache	[1,1,1,1,1,1,....]

Reducer Logic

Here, we simply count each of the values for a hobby. And finally we will have

Key	Value
Acrobatics	6641
Meditation	3338
Music	3338
Photography	3303
Papier-Mache	6661

Code

example3.js

```
1  var mongojs = require('mongojs');
2  var db = mongojs('mapReduceDB', ['sourceData', 'example3_r
3
4
5  var mapper = function () {
6       var hobbys = this.hobbies.split(',');
7        for (i in hobbys) {
8          emit(hobbys[i], 1);
9      }
10 };
11
12 var reducer = function (key, values) {
13     var count = 0;
14     for (index in values) {
```

```
15          count += values[index];
16      }
17
18      return count;
19  };
20
21
22  db.sourceData.mapReduce(
23      mapper,
24      reducer,
25      {
26          out : "example3_results"
27      }
28   );
29
30   db.example3_results.find(function (err, docs) {
31      if(err) console.log(err);
32      console.log(docs);
33   });
```

Do notice on **lines 7 to 9**, we iterate through each hobby and emit one count of it. **Lines 13 to 18** can replaced with a simple `Array.sum(values)` , but this is another way of doing the same. And the finally we run the job and the result would be

```
Arvinds-MacBook-Pro:mongoDBMapReduce arvindravulavaru$ node example3.js
[ { _id: 'Acrobatics', value: 6641 },
  { _id: 'Meditation', value: 3338 },
  { _id: 'Music', value: 3338 },
  { _id: 'Papier-Mache', value: 6661 },
  { _id: 'Photography', value: 3303 } ]
```

If you did notice, the collection will get overridden when you run a new query on an existing collection.

So, this is how we can run MapReduce jobs in MongoDB. But do remember that sometimes a simple query can get the job done.

Thanks for reading! Do comment.
@arvindr21

Like this:

Be the first to like this.

Related

Nodejs Restify MongoDB - Build your own REST API
April 20, 2014
In "Database"

Building a Todo App with DiskDB
August 24, 2014
In "Angularjs"

Re-Architecting a Firebase app to work with Node.js and MongoDB
December 12, 2014
In "Architecture"

APRIL 23, 2014 ARVIND RAVULAVARU

22 Comments The Jackal of Javascript 1 Login

Recommend Share Sort by Best

Join the discussion…

acveer · 7 months ago

Aravind,
Excellent tutorial. Well-written. It help me get started with mapReduce. I have tried your examples with mongoDB aggregate framework & thought these might help someone. My intention here is share what I have learned and to seek expertise on the

here is share what I have learned and to seek expertise on the aggregation framework.
Below queries can be run on mongo shell to get the same results with aggregate framework.
Example1 :
db.sourceData.aggregate({$group: {_id: "$gender", total: {$sum: 1}}});

Example2 was a little complicated for me, still learning/trying. If you know how to achieve this, please help.

Example3:

The dummy-json template + custom helper logic above is returning hobbies like below:

see more

1 · Reply · Share ›

Arvind Ravulavaru Mod acveer · 7 months ago

Hello acveer. Thanks! And Thanks for sharing your solutions.

I guess you can tweak the `dataGen.js` to give you the array you want from inside the `hobby()`. Let me know if it works.

A few days ago I was planning to do a comparison between AF and MR, never got to it though. Did you try anything in that space?

Thanks.

· Reply · Share ›

acveer Arvind Ravulavaru · 7 months ago

Arvind,

I have not done enough MP or AF for the detailed comparision, but here is what I understood so far.

1. AF is much faster than MP.
2. Both are targeted to be used in batch operations in background mode (not real-time). (so convenience/control).
3. Both have respective limitations in terms of

document size, max documents in process (100MB) and sharding support.
4. MongoDB 2.6 has more handy features added for AF.

I could not figure out how to insert hobbies array by changing the template or helpers function. The array needs to be this way ""hobbies" : ["Painting","Cooking","Reading"]. So I just inserted empty hobbies array from datagen.js and written another script to update hobbies array separately.

Will post solution for example 2, once I make progress.

Reply · Share ›

Arvind Ravulavaru Mod → acveer · 7 months ago

Thanks for sharing your findings. I will also take a look at hobbies array and get back.

Reply · Share ›

Peter Boot · 2 months ago

What is the correct format for template.hbs ?

Reply · Share ›

Arvind Ravulavaru Mod → Peter Boot · 2 months ago

Correct format as in?

Reply · Share ›

Logic Town · 3 months ago

Do you know if it's possible to invoke a function provided by an npm module inside the mapper or reducer functions?

For instance

```
var mymod = require('mymod');

var mapper = function(){
emit(this.key, mymod.apply(this));
}

...
```

• Reply • Share ›

Arvind Ravulavaru Mod ➚ Logic Town • 3 months ago

I have not tried but it should be possible. Did you try it?

• Reply • Share ›

Logic Town • 3 months ago

I reckon the functions such as

```
db.example3_results.find(function (err, docs) {
if(err) console.log(err);
console.log(docs);
});
```

should be passed as a callback to the mapReduce command, otherwise they will be invoked before the job completes.

• Reply • Share ›

Arvind Ravulavaru Mod ➚ Logic Town • 3 months ago

I am not sure if Mongojs API has a callback for `mapreduce()`, if it does then that is the place to put it. But I think `mapreduce()` in a blocking way here. Not sure though.

• Reply • Share ›

Harshavardhan Reddy • 5 months ago

Hey Aravind,
I want to do with multiple collections.
Can you please help with that?
Thank you.

• Reply • Share ›

Arvind Ravulavaru Mod ➚ Harshavardhan Reddy • 5 months ago

Hello Harsha,

Take a look at : http://stackoverflow.com/a/383...

Thanks,
Arvind.

• Reply • Share ›

rajkumar • 7 months ago

Hi i have json like this

"marks":{
"sem1" :{
"mark1":10,
"total":100
},
"sem2":{
"mark2":20,
"total":200
},
"sem3":{
"mark2":30,
"total":300
}
}

I need result like

mark total sem

10 100 sem1
20 200 sem2
30 300 sem3

how can i achive above format using monogodb query.query is jaspersoft related means very useful

 · Reply · Share ›

Arvind Ravulavaru Mod ➚ rajkumar · 7 months ago

I have not worked on jaspersoft so I am not sure how the query needs to written. If you want to return an object and compare a value refer example 2 in the post. -- Thanks.

 · Reply · Share ›

syd · 8 months ago

and thanks for your quick response.

 · Reply · Share ›

syd · 8 months ago

can you explain a little briefly please, as i am doing a project to calculate the time taken by mongoDB and hadoop in mapReduce algorithm when they store or retrieve different types of bulk data.

• Reply • Share ›

Arvind Ravulavaru Mod ➚ syd • 7 months ago

I have not dug deeper into the algorithms, so I am not aware of it. You can look for more info here : https://github.com/mongodb

• Reply • Share ›

syd ➚ Arvind Ravulavaru • 7 months ago

Thank you

1 • Reply • Share ›

syd • 8 months ago

what are the different techniques to find the time taken by my query to fetch output in mapReduce

• Reply • Share ›

Arvind Ravulavaru Mod ➚ syd • 8 months ago

AFAIK, there is no official benchmarking tool for MongoDB. You can track the time taken from within the node application. Try this : http://blog.nodejs.org/2012/04...

Or you can try

```
console.time('myJob');
// your job
console.timeEnd('myJob');
```

This should give a fair idea.

• Reply • Share ›

sppericat • a year ago

This is nice, but why don't you use the the aggregation framework instead ?

• Reply • Share ›

Arvind Ravulavaru Mod ➚ sppericat • a year ago

Thanks sppericat,

No argument there. You can use an aggregation framework to do the same. I wanted to throw some light on aggregating data using MapReduce.

IMO, MapReduce is a verbose version of the aggregation framework & is also an alternative. The only key difference I see between MapReduce & Aggregation Framework is the built in pipe operators like $geoNear. $sum, $gte etc. In a MapReduce paradigm, you end up writing these on your own.

Thanks,
Arvind.

 • Reply • Share ›