Contents

Introduction 2
Example Data Set 3
Map Reduce 4
Mapping 5
Reducing 7
Data Aggregations 9
Aggregates vs Map/Reduce 11
Conclusion 12

Introduction

MongoDB provides support for large data processing tasks such as Map Reduce and Aggregation. Map Reduce is the process of processing the entire database with 2 steps **Map** and **Reduce**. The Map step maps every document in the database to a category or key. Then in the reduce step, every key reduces all of its mapped values by aggregating all the values with some type of algorithm.

MongoDB aggregation tasks allow similar operations as Map Reduce but works as a pipeline rather than a 2 step process. Aggregations allow you take a collection and transform it N number of times until you get the collection you desire. The advantage to Aggregations vs Map Reduce is that it allows you to only process the parts of your database you need and exclude parts you don't. Map Reduce requires that your *entire* database be processed.

Below I will take the example treated by Steven Edouard and Rami Sayyar from Microsoft in the module4 from the “You've Got Documents! A MongoDB Jump Start”. You can see the video on YouTube using the link https://www.youtube.com/watch?v=W-WihPoEbR4

Example Data Set

The example data set we will be dealing with is City of Chicago crime police report data from November 2014 to December 2015. The data comes in a large .csv file you can also find the original unzipped download link from the City of Chicago here.

After downloading the CSV file, we can import this data to our database using the **`mongoimport`** utility we used previously to import the test bank_data json:

```
mongoimport Crimes_-_2001_to_present.csv --type csv --headerline --
collection crimes
```

The `--type` parameter specifies its a csv file, `--headerline` indicates that the first line of the csv file has the field names and obviously `--collection` specifies the collection to insert the new documents into.

NB: Doing this command may take a while and it should be noted that you should do this on a fairly fast machine or else working with data this large may hang up your machine.

Afterwards you should have about 301261 documents uploaded, each representing a police report incident. Here's a sample of what your documents will look like:

```
{ "_id" : ObjectId("5462725476ecd357dbbc721e"), "ID" : 9844675, "Case
Number" : "HX494115", "Date" : "11/03/2014 11:51:00 PM", "Block" : "056XX S
MORGAN ST", "IUCR" : 486, "Primary Type" : "BATTERY", "Description" :
"DOMESTIC BATTERY SIMPLE", "Location Description" : "ALLEY", "Arrest" :
"false", "Domestic" : "true", "Beat" : 712, "District" : 7, "Ward" : 16,
"Community Area" : 68, "FBI Code" : "08B", "X Coordinate" : 1170654, "Y
Coordinate" : 1867165, "Year" : 2014, "Updated On" : "11/10/2014 12:43:02
PM", "Latitude" : 41.790980835, "Longitude" : -87.649786614, "Location" :
"(41.790980835, -87.649786614)" }
```

Map Reduce

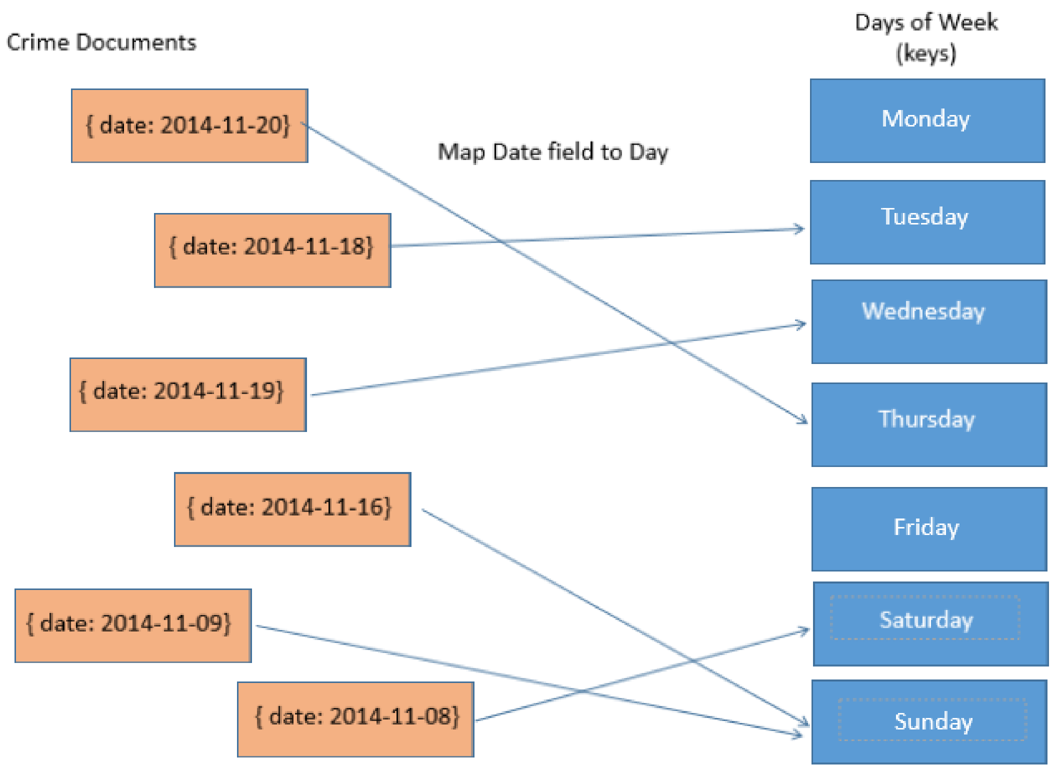
The best way to explain map reduce is to attempt to answer a question about the massive amount of data we have. Let's try this one out for size:

What day of the week has the most crime incidents recorded in Chicago?

Map Reduce breaks down this problem into 2 steps, **Map** and **Reduce**.

Mapping

The question essentially asks us to break down the number of crimes that have occurred from November 2014 in Chicago by day. Notice that for each crime document there is a date field which indicates the exact date of the incident. We can use this data to **map** the crime document to a specific day of the week. Here's a graphical view of what we will be doing in this step:



Each of the 301261 police reports will be processed by MongoDB and we will specify that we want the document to be mapped to either the 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', or 'Sunday' keys depending on the date field of the document. We do this by calling the **mapReduce** function on the collection. **mapReduce** takes two functions as parameters, a **map** function and a Reduce function.

The MongoDB driver (and interactive shell) allow us to specify a **map** function which defines how MongoDB will map the documents to their respective keys. To emit a map, the **emit** function can be called within the function to let MongoDB know that you have a mapping for this document.

Here's what this looks the function using the interactive shell:

```
var map=function () {
    var milis = Date.parse(this.Date);
    var date = new Date(milis);
    var daysOfWeek = ["Sunday", "Monday", "Tuesday", "Wednesday",
"Thursday", "Friday", "Saturday"];
    //emit the mapping between this document and the Day which it
cooresponds to.
    emit(daysOfWeek[date.getDay()], 1);
}
```

Notice that we parse the date field using the JavaScript Date class and then use that to easily acquire the day of the week. We then call getDay() to retrieve the day which is a number between 0 and 6. We then use a JavaScript array to map those values to a day of the week since every JavaScript object is already a key/value dictionary.

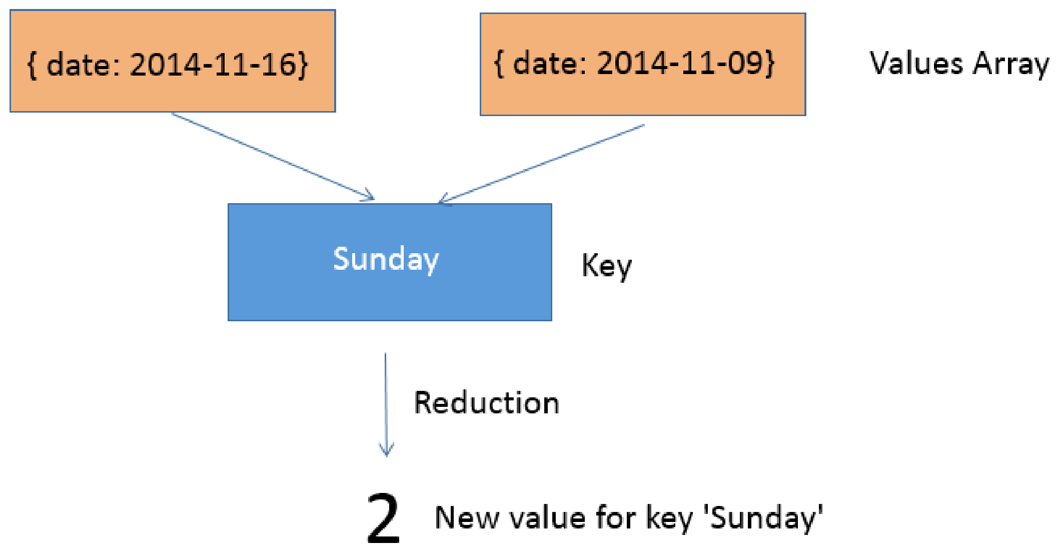
We can't execute the code snippet above since it's not complete yet without our **reduce** function.

Reducing

The second step in this process is to reduce the mappings into a resulting data set which sort of *summarizes* the mappings we've created. The original question at hand would like us to *summarize* which day of the week crimes happened most frequently in Chicago.

The Map step has already created a large amount of mappings of 301261 crime documents to 7 different types of mappings. If we summarized the data by summing the total number of crime documents for each key (the day of the week) we would quickly be able to answer the question at hand.

Here's a Visual representation of what the Reduce step does:



The **reduce** function is the second parameter to the **mapReduce** function where we do this summarization. Reduce is called with 2 parameters, **key** and **values**.

```
// function to reduce where we do the summarizations
var reduce= function (key, values){  return   Array.sum(values);  }
```

It turns out that the reduce function for this question is quite easy. We just have to return the number of documents that have been mapped to the key (the day of the week).

The final parameter for **mapReduce** is the output collection for the results. This is the collection where the results for each key will be placed. We can pass a simple javascript object that specifies this as an **out** collection:

```
{
    out: "crime_day_frequencies"
},
```

Putting the entire MapReduce call together in the shell looks like:

```
//Question: What day did most crimes occur in chicago from September 2001
to present?
  var map=function () {
    var milis = Date.parse(this.Date);
    var date = new Date(milis);
    var daysOfWeek = ["Sunday", "Monday", "Tuesday", "Wednesday",
"Thursday", "Friday", "Saturday"];
    //emit the mapping between this document and the Day which it
cooresponds to.
    emit(daysOfWeek[date.getDay()], 1);
}
//======================================================================

var reduce= function (key, values){  return   Array.sum(values);  }

//======================================================================

var result = db.runCommand(
{"mapreduce" : "crimes",
"map" : map,
"reduce" : reduce,
"out" : " crime_day_frequencies "})

//======================================================================
```

Finally, the output of this map/reduce job gives us the answer we were looking for in the form of a MongoDB collection:

```
Number of crimes based on each day of the week
{ "_id" : "Friday", "value" : 44449 }
{ "_id" : "Monday", "value" : 43027 }
{ "_id" : "Saturday", "value" : 44305 }
{ "_id" : "Sunday", "value" : 42866 }
{ "_id" : "Thursday", "value" : 41974 }
{ "_id" : "Tuesday", "value" : 42071 }
{ "_id" : "Wednesday", "value" : 42569 }
```

From the results above, barring any actual statistical science, it appears that the most common day that crimes have occurred in Chicago since November 2014 to December 2015 has been Fridays. This could be a very interesting insight!

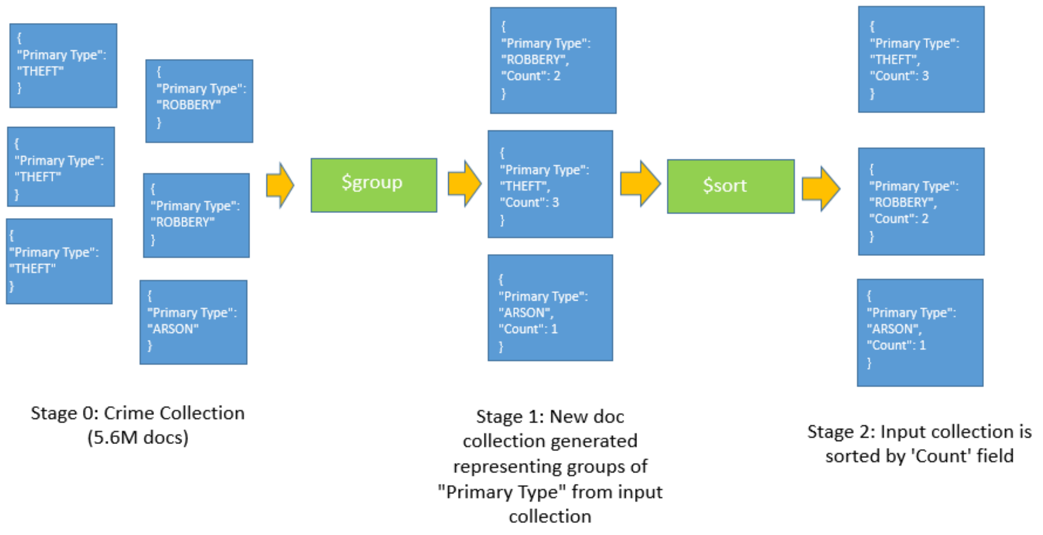
Data Aggregations

Aggregations work on a concept of processing pipelines which consist of a number of stages. At each stage the data in the returned collection of documents is transformed.

As before, the best way to learn this concept is by asking a question about the data and using Aggregations to answer it. Let's go with the question:

What is the most common type of crime committed in Chicago between November 2014 and December 2015?

It’s possible to answer this with Map/Reduce but let's use Aggregates instead. To do aggregates we first start with the original crimes collection of 301261 documents, group those documents by the `Primary Type` field, keep track of the count for the number of documents in the Primary Type group and then finally sort those groups by the generated count values.
Visually we are processing the crimes collection through a series of pipeline steps:


{ "Primary Type": "THEFT" }
{ "Primary Type": "ROBBERY" }
{ "Primary Type": "THEFT" }
{ "Primary Type": "ROBBERY" }
{ "Primary Type": "THEFT" }
{ "Primary Type": "ARSON" }
$group
{ "Primary Type": "ROBBERY", "Count": 2 }
{ "Primary Type": "THEFT", "Count": 3 }
{ "Primary Type": "ARSON", "Count": 1 }
$sort
{ "Primary Type": "THEFT", "Count": 3 }
{ "Primary Type": "ROBBERY", "Count": 2 }
{ "Primary Type": "ARSON", "Count": 1 }
Stage 0: Crime Collection (5.6M docs)
Stage 1: New doc collection generated representing groups of "Primary Type" from input collection
Stage 2: Input collection is sorted by 'Count' field

Stage 0 is the initial collection of Crime elements, which has approximately 301261 documents. We then use a **$group** operator to group all those documents into groups by `Primary Type` which essentially is the type of crime that was committed.

This leaves us with a new collection of documents one for each group, We will add a **count** field to each group to record how many documents are in that group. Finally we do our next pipeline step which is to sort the input collection which is the collection that came out of the $group operator using the **$sort** operator. We will sort descending by the count field on the grouped collection.

Here's what this looks like:

```
db.crimes.aggregate({ $group : { _id: "$Primary Type" , count:{$sum: 1} }},
{$sort: { count: -1 } })
```

For the provided dataset, here are the results, which also yields some interesting insights:

```
{ "_id" : "THEFT", "count" : 66446 }
{ "_id" : "BATTERY", "count" : 56372 }
{ "_id" : "CRIMINAL DAMAGE", "count" : 33281 }
{ "_id" : "NARCOTICS", "count" : 25799 }
{ "_id" : "OTHER OFFENSE", "count" : 19937 }
{ "_id" : "ASSAULT", "count" : 19494 }
{ "_id" : "DECEPTIVE PRACTICE", "count" : 16359 }
{ "_id" : "BURGLARY", "count" : 15636 }
{ "_id" : "MOTOR VEHICLE THEFT", "count" : 11805 }
{ "_id" : "ROBBERY", "count" : 11408 }
{ "_id" : "CRIMINAL TRESPASS", "count" : 7566 }
{ "_id" : "WEAPONS VIOLATION", "count" : 3813 }
{ "_id" : "PUBLIC PEACE VIOLATION", "count" : 2827 }
{ "_id" : "OFFENSE INVOLVING CHILDREN", "count" : 2486 }
{ "_id" : "PROSTITUTION", "count" : 1612 }
{ "_id" : "INTERFERENCE WITH PUBLIC OFFICER", "count" : 1501 }
{ "_id" : "CRIM SEXUAL ASSAULT", "count" : 1483 }
{ "_id" : "SEX OFFENSE", "count" : 978 }
{ "_id" : "HOMICIDE", "count" : 562 }
{ "_id" : "ARSON", "count" : 510 }
Type "it" for more
> it
{ "_id" : "LIQUOR LAW VIOLATION", "count" : 341 }
{ "_id" : "GAMBLING", "count" : 337 }
{ "_id" : "KIDNAPPING", "count" : 222 }
{ "_id" : "STALKING", "count" : 173 }
{ "_id" : "INTIMIDATION", "count" : 133 }
{ "_id" : "OBSCENITY", "count" : 53 }
{ "_id" : "CONCEALED CARRY LICENSE VIOLATION", "count" : 35 }
{ "_id" : "NON-CRIMINAL", "count" : 29 }
{ "_id" : "NON - CRIMINAL", "count" : 25 }
{ "_id" : "PUBLIC INDECENCY", "count" : 16 }
{ "_id" : "HUMAN TRAFFICKING", "count" : 14 }
{ "_id" : "OTHER NARCOTIC VIOLATION", "count" : 7 }
{ "_id" : null, "count" : 1 }
```

From our results, it would appear that Theft and Battery by far have been the most common types of crimes in Chicago for the past year.

Aggregates vs Map/Reduce

Map Reduce and Aggregations are very powerful tool sets to gain insights into MongoDB database. It makes sense to compare and contrast the difference between the two.

Map Reduce is fundamentally designed to process all of the data. Every single document is looked at least once in a Map/Reduce task. Map reduce allows for more complex logic in mapping documents to keys. Similarly you can implement more complex behavior for reduction as well.

Aggregations came to MongoDB after version 2.0 partially out of a need to avoid having to process the entire database if it wasn't needed. Because aggregations use a pipeline model, you can chain as many aggregation operations as you'd like and each subsequent operation can perform better given that the operation before reduced the number of documents.

In short, if the aggregation operators suit your needs and you don't need to process your entire database for each aggregation, than aggregates show a better value. If you constantly process the entire database or have more complex Map or Reduce logic Map/Reduce may be a more attractive option.

Conclusion

Whether you use Map Reduce or Aggregates you will notice that these tasks take quite some time. Due to this these operations are not designed to be real-time requests and thus should really be ran in the background of your application. Think - hourly, daily or weekly runs where you aggregate some result and do something interesting with that data.

With Map/Reduce and Aggregates you can really find fun and interesting insights without having to write very much code at all.