MONTANA
STATE UNIVERSITY

MongoDB

Aggregation In Mongo

MongoDB

- Last week we looked at MongoDB, a popular Document Database
- This week we will discuss the Mongo aggregation framework

Aggregation in SQL

- Recall that aggregation in a RDBMS uses the GROUP BY clause in a SELECT statement
- Groups rows of data into a single row
  - Typically paired with an aggregation function such as SUM() or COUNT()

```sql
SELECT
        tracks.AlbumId,
        title,
        SUM(Milliseconds) AS length
FROM
        tracks
INNER JOIN albums ON albums.AlbumId = tracks.AlbumId
GROUP BY
        tracks.AlbumId
HAVING
        length > 60000000;
```

Aggregation in Mongo

- Aggregation in MongoDB is similar in some ways, and distinct in others
- Here is an example that aggregates the orders collection

```
db.orders.aggregate([
   { $match: { status: "A" } },
   { $group: { _id: "$cust_id", total: { $sum: "$amount" } } }
])
```

Aggregation in Mongo

- In Mongo, aggregations are done in stages
- In this example
  - Stage 1 - A Match Stage
    - All documents matching the given filter are passed through to the next stage
  - Stage 2 - A Group Stage
    - All documents that passed the match stage are grouped by some set of attributes

```
db.orders.aggregate([
   { $match: { status: "A" } },
   { $group: { _id: "$cust_id", total: { $sum: "$amount" } } }
])
```

Aggregation in Mongo

- Here the match stage is picking out all orders whose status is “A”
- And the group stage is grouping by the cust_id field, and computing a sum of the amount field

```
db.orders.aggregate([
   { $match: { status: "A" } },
   { $group: { _id: "$cust_id", total: { $sum: "$amount" } } }
])
```

Aggregation in Mongo

- While this may appear superficially similar to the GROUP BY clause in SQL, it is in fact more general
- An aggregation can consist of any number of stages
- Each stage can transform or filter documents to be passed on to the next stage

```
db.orders.aggregate([
   { $match: { status: "A" } },
   { $group: { _id: "$cust_id", total: { $sum: "$amount" } } }
])
```

Aggregation in Mongo

- Stage types
  - There are many different stage types
    - $match - matches documents
    - $group - groups documents
    - $sort - sorts documents
    - $limit - limits the number of documents
    - Etc.

```
db.orders.aggregate([
   { $match: { status: "A" } },
   { $group: { _id: "$cust_id", total: { $sum: "$amount" } } }
])
```

Aggregation in Mongo

- Stages can be in any order
- Here the $group stage and the $match stage are reversed
    - This $match stage is acting like a HAVING clause in SQL: it applies to the aggregate

```
db.sales.aggregate(
  [
    // First Stage
    {
      $group :
        {
          _id : "$item",
          totalSaleAmount: { $sum: { $multiply: [ "$price", "$quantity" ] } }
        }
     },
     // Second Stage
     {
       $match: { "totalSaleAmount": { $gte: 100 } }
     }
   ]
 )
```

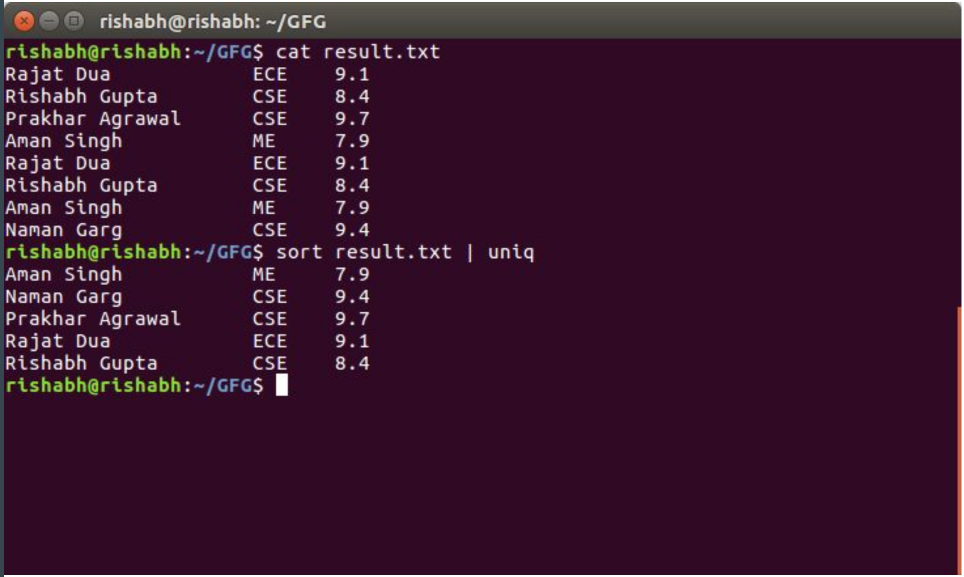
Aggregation in Mongo

- Most stage types can appear multiple times in a an aggregation pipeline
  - Here we group twice
    - Once to sum up all the population by city in a zipcode database
    - And then we group that info again by the state, using an average to get the average population of cities in a state

```
db.zipcodes.aggregate( [
   { $group: { _id: { state: "$state", city: "$city" },
               pop: { $sum: "$pop" } } },
   { $group: { _id: "$_id.state",
               avgCityPop: { $avg: "$pop" } } }
] )
```

Aggregation in Mongo

- The MongoDB aggregation pipeline is inspired by the Unix pipe concept
- You connect an arbitrary number of commands together to produce the desired results



```
rishabh@rishabh: ~/GFG
rishabh@rishabh:~/GFG$ cat result.txt
Rajat Dua           ECE    9.1
Rishabh Gupta       CSE    8.4
Prakhar Agrawal     CSE    9.7
Aman Singh          ME     7.9
Rajat Dua           ECE    9.1
Rishabh Gupta       CSE    8.4
Aman Singh          ME     7.9
Naman Garg          CSE    9.4
rishabh@rishabh:~/GFG$ sort result.txt | uniq
Aman Singh          ME     7.9
Naman Garg          CSE    9.4
Prakhar Agrawal     CSE    9.7
Rajat Dua           ECE    9.1
Rishabh Gupta       CSE    8.4
rishabh@rishabh:~/GFG$
```

Map/Reduce

- Mongo also supports the Map/Reduce programming model
- Map/Reduce is a programming model for processing large sets of data in a distributed manner
    - Cluster Friendly

Map/Reduce

- Concept has been patented by Google and was widely used there
  - As an intern I ported various batch processing jobs to the google Map/Reduce infrastructure
  - It was alright

MapReduce
frameworks and methods

Map/Reduce

- According to Wikipedia:

  “By 2014, Google was no longer using MapReduce as their primary big data processing model”

Map/Reduce

- Nonetheless, the idea has had a large influence on the Big Data world
- Many open source implementations now
    - Hadoop is the most popular one that I am aware of

MapReduce
frameworks and methods

Map/Reduce

- Map/Reduce algorithm
  - Data is read as an input
  - A function then *maps* each piece of input to a key/value pair
  - Partition: each key/value pair is sent to a *Reducer* based on the mapp value

Map/Reduce

- Map/Reduce algorithm
  - The reducer sorts the input and then processes it using its reduce function
  - This reduce function produces zero or more outputs
  - This output is finally written to stable storage

Google GTUG

MapReduce
frameworks and methods

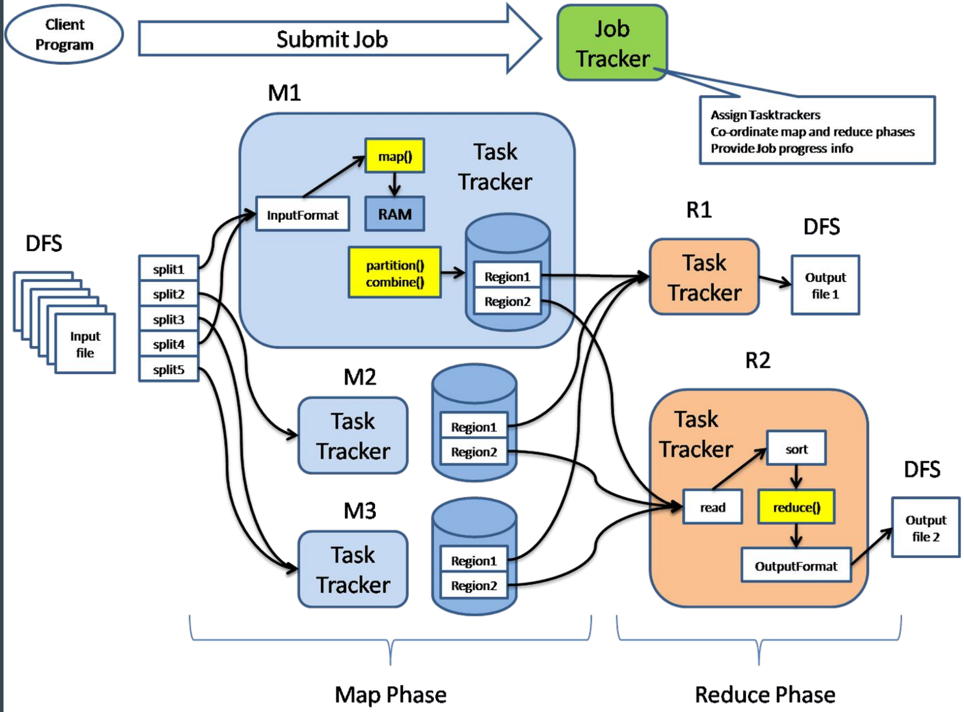
Map/Reduce

- There is controversy if this is a unique or even “good” idea
- It is taken from the functional programming concepts of the same names
- Many distributed computing researchers felt it was not a new idea, nor particularly flexible

MapReduce
frameworks and methods

Hadoop Map/Reduce

- Hadoop infrastructure
  - Client submits a job
  - Job is split into multiple mappers
  - Mappers partition and combine data
  - Reducers (optionally) read combined data and reduce to a final value



Hadoop

- Job Setup
  - Configure the map reduce job
  - Submit it to the JobTracker

```
//Main function
public static void main(String args[])throws Exception {
   JobConf conf = new JobConf(ProcessUnits.class);

   conf.setJobName("max_eletricityunits");
   conf.setOutputKeyClass(Text.class);
   conf.setOutputValueClass(IntWritable.class);
   conf.setMapperClass(E_EMapper.class);
   conf.setCombinerClass(E_EReduce.class);
   conf.setReducerClass(E_EReduce.class);
   conf.setInputFormat(TextInputFormat.class);
   conf.setOutputFormat(TextOutputFormat.class);

   FileInputFormat.setInputPaths(conf, new Path(args[0]));
   FileOutputFormat.setOutputPath(conf, new Path(args[1]));

   JobClient.runJob(conf);
}
```

Hadoop

- Mapper
  - Split value on tabs
  - Take the first value as year
  - Scan to end for last value
  - Parse as integer for average price for this row
  - Collect into a group based on the year

```
//Mapper class
public static class E_EMapper extends MapReduceBase implements
Mapper<LongWritable ,/*Input key Type */
Text,                /*Input value Type*/
Text,                /*Output key Type*/
IntWritable>        /*Output value Type*/
{
   //Map function
   public void map(LongWritable key, Text value,
   OutputCollector<Text, IntWritable> output,

   Reporter reporter) throws IOException {
      String line = value.toString();
      String lasttoken = null;
      StringTokenizer s = new StringTokenizer(line,"\t");
      String year = s.nextToken();

      while(s.hasMoreTokens()) {
         lasttoken = s.nextToken();
      }
      int avgprice = Integer.parseInt(lasttoken);
      output.collect(new Text(year), new IntWritable(avgprice));
   }
}
```

Hadoop

- Reduce
  - Iterate over input grouping
  - Filter out anything with an average price less than 30

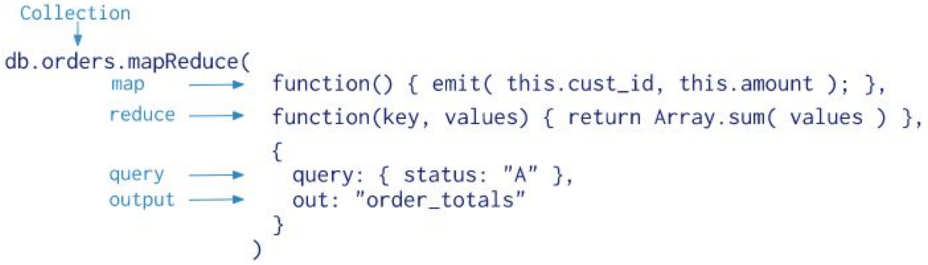
```
//Reducer class
public static class E_EReduce extends MapReduceBase implements Reducer< Text, IntWrita

   //Reduce function
   public void reduce( Text key, Iterator <IntWritable> values,
   OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
      int maxavg = 30;
      int val = Integer.MIN_VALUE;

      while (values.hasNext()) {
         if((val = values.next().get())>maxavg) {
            output.collect(key, new IntWritable(val));
         }
      }
   }
}
```

Hadoop

- Not a super interesting MapReduce but it gives you the flavor
- Hadoop is very mature and widely used in industry
- And they have a good logo

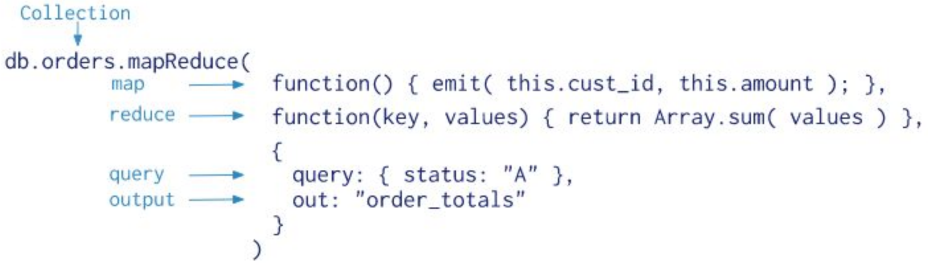
Mongo MapReduce

- In MongoDB, collections have a mapReduce() function available directly on them
- This function takes three arguments
  - A map function
  - A reduce function
  - A query/output pair



Mongo MapReduce

- The map function takes inputs from the collection and *maps* them to a particular key
- The reduce function takes a particular key and all the values mapped to it, and transforms them to a new, final value

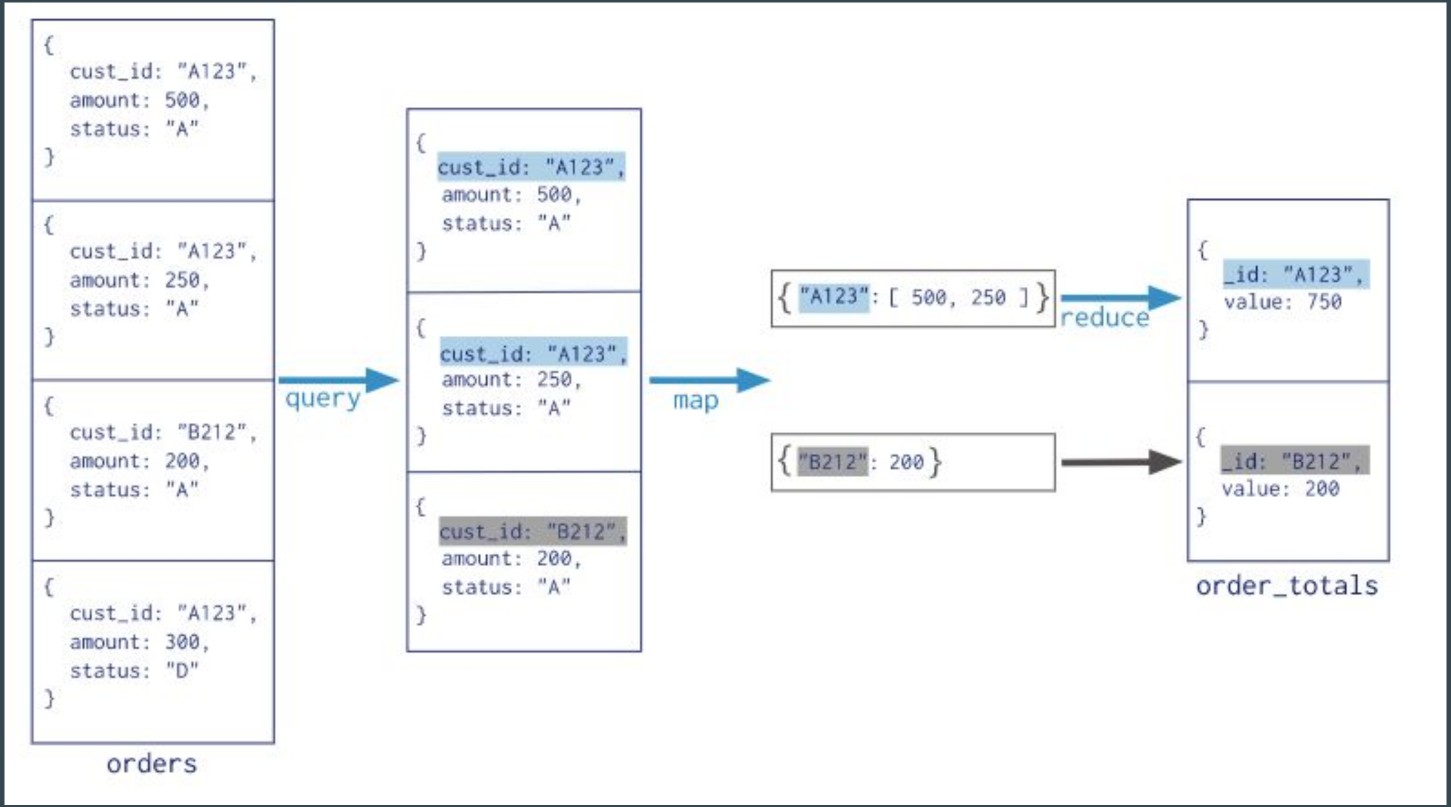


Mongo MapReduce

- *query* allows a pre-MapReduce filter to be applied to documents
- *out* allows you to specify an output collection (or inline)

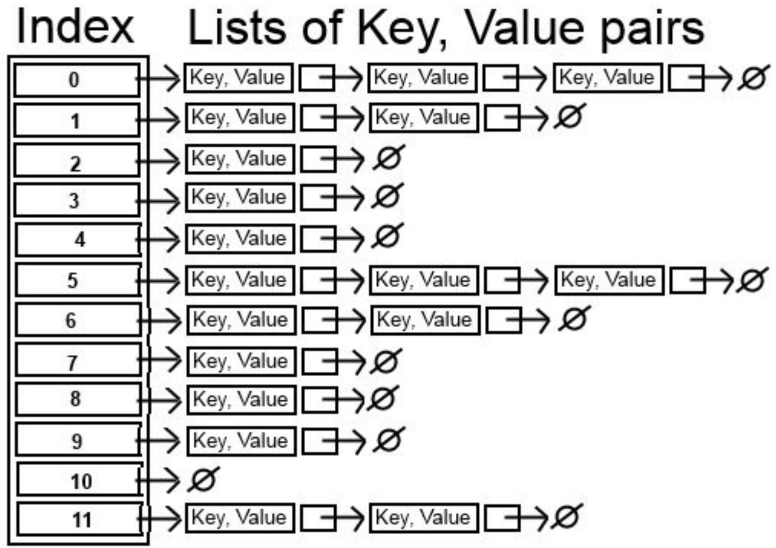
```
  Collection
      ↓
db.orders.mapReduce(
         map     ——→  function() { emit( this.cust_id, this.amount ); },
         reduce  ——→  function(key, values) { return Array.sum( values ) },
                      {
         query   ——→    query: { status: "A" },
         output  ——→    out: "order_totals"
                      }
                    )
```

Mongo MapReduce



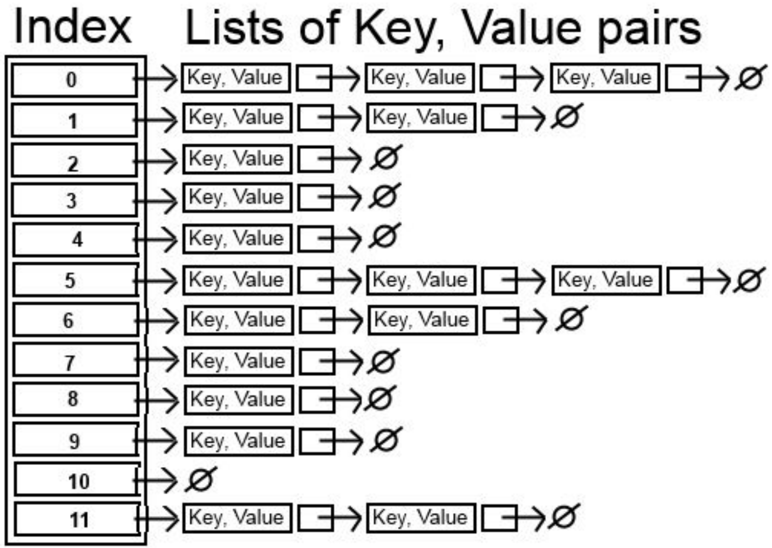
MapReduce & Hash Tables

- MapReduce works, in part, as a sort of network aware hash table
- Recall how a hash table works
  - A key is hashed to a particular index
  - A linked list (or similar) is maintained for each hash bucket



MapReduce & Hash Tables

- MapReduce suffers from many of the same pathological cases as Hashtables
  - E.g. What if everything maps to the same index?



MapReduce & SQL

- As we have discussed, aggregation in Mongo is similar but not identical to the GROUP BY functionality offered in SQL
- This table shows a rough correspondence between SQL feature and stage type

SQL Terms, Functions, and Concepts	MongoDB Aggregation Operators
WHERE	`$match`
GROUP BY	`$group`
HAVING	`$match`
SELECT	`$project`
ORDER BY	`$sort`
LIMIT	`$limit`
SUM()	`$sum`
COUNT()	`$sum` `$sortByCount`

MapReduce & SQL

- Keep in mind, though: Mongo allows multiple instances of most stage types
  - More like the Unix pipeline than traditional SQL and RDBMS implementations

SQL Terms, Functions, and Concepts	MongoDB Aggregation Operators
WHERE	`$match`
GROUP BY	`$group`
HAVING	`$match`
SELECT	`$project`
ORDER BY	`$sort`
LIMIT	`$limit`
SUM()	`$sum`
COUNT()	`$sum` `$sortByCount`

Mongo Aggregation

- MongoDB provides an aggregation framework similar in some ways to what is available in SQL
- Mongo Aggregation has *stages* rather than a fixed syntax
- A stage has a particular *stage type* that provides a particular type of functionality
    - Sorting, grouping, etc.
- Stages can be chained together, like a Unix pipeline
- Most stage types can appear more than once in a mongo aggregation
- Mongo also provides a MapReduce API that can be used for aggregation calculations

MONTANA
STATE UNIVERSITY