[II.2414]Big Data and Data Bases

NoSQL Project Report

Olof Berg Marklund
Xalo Rancaño Rodríguez
Deepthi Mamillapalli

Group

July 2, 2018

Table of contents

1 Introduction

The aim of this project is to deepen our knowledge in how a company could retrieve and analyze data using a NoSQL database. The report will describe our struggle and work method for importing all the data and trying to represent the data in a correct manner. Because we spent so much time trying to figure out how to make the queries with our first database model with separate collection, this report will describe that failure aswell as our final solution
This section describes the process from how the data from EnerNOC have been recovered to which model is appropriate to use in NoSQL to work with the data. Some NoSQL queries will also be defined and analyzed.

1.1 Data Integration

ENERNOC has consumption data for 100 commercial/industrial sites from 2012. Each site has metadata, ID, square footage, lat/lng and timezone provided. A time series of electricity consumption is called a Load Curve, in this report it will be called LD.
The folder all-data was downloaded from the following page: `https://open-enernoc-data.s3.amazonaws.com/anon/index.html`

We noticed that the files where in csv format. We suspected that some parts could be easier if we converted them to JSON, because of the way you can look at the object in a browser console when using JSON.

In our first attempt we tried to load the files into MongoDB as they where in the csv format. We did this by putting the data into the bin folder, and use "mongoimport" with the different files.

```
./mongoimport --type csv -d metadb -c metadata --file
    ../all-data/meta/all\_sites.csv --headerline
```

We got the error:

```
Failed: fields cannot be identical: 'America/Chicago' and
    'America/Chicago'
```

We found a fix by opening it in Microsoft Excel and then saving it again, we did not have that program available at the moment and chose to convert it to JSON instead. The conversion was made with the site `https://www.csvjson.com/csv2json`
After that we did the import with the command previously mentioned.

```
./mongoimport --type json -d metadb -c metadata --file
    ../all-data/meta/csvjson.json --jsonArray
```

For the rest of the data we wrote a for loop to import all the data:

```
for i in *.csv; do ../../bin/mongoimport --db metadb
    --collection ${i%.*} --type csv --file $i --headerline ;
    done
```

The collection name was the same as the file names. We quickly noticed that having the collection started with a number is not allowed. We dropped the data, and then we imported it again with the collection name "data[filename]" instead.

```
for i in *.csv; do ../../bin/mongoimport -d metadb
   --collection data${i%.*} --type csv --file $i --headerline
   ; done
```

To use the new database "metadb" and to make sure it was implemented correctly we used the following commands:

```
use metadb
show collection
db.[collectionname].find();
```

According to the little knowledge and experience we had with Data Bases in general, and MongoDB in particular, we didn't thought about merging all the files together in a first place. The reason was we thought that the date for all the sites was obtained from the same period of time, so that at least the timestamps would be the same for all of them. We started by solving the first exercise following this idea. We thought that the best way to solve our problem would be to create a loop that would enter in each file, retrieve the specific value of Energy Consumption for each specific timestamp and sum them up. We would implement this loop into another loop that would be changing the timestamp, so that we would be able to make the sum of Energy Consumption per timestamp every 5 mins.

```
db.data6.find().forEach(function(row){
    var sum = 0;
db.getCollectionNames().forEach(function(collname){
     var sum = sum + db[collname].findOne( {"timestamp" :
        row.timestamp} ).value
    })
print("sum for row " + JSON.stringify(row.timestamp) + " is "
   + sum);
});
```

We ended up realising that not all the sites had timestamps from the same period of time, so we could not implement a solution like this. Although it ended up not working, we had already created some valid simple queries to add to the project, so we will add them here also, even though we have created new ones for the merged file with all the data from all the sites.
After a lot of errors we decided to finally try import all the data to one collection. First we tried to do this by using the mongoDB javasrcipt syntax, this is an example of how we tried to enter the SITE_ID from the metadata into each and single row for all the different csv file.

```
db.getCollectionNames().forEach(function(collname) {
    db[collname].find().forEach(function(row){
        db.metadata.find().forEach(function(doc){
            db[collname].update({timestamp: row.timestamp },
               {$set:{"SITE_ID":doc.SITE_ID}});
        });
    });
});
```
And found out that it would be easier to do it in another way. So we created a python script which creates a csv file by reading the data from the other csv file.

```
import numpy as np

loadercsv =
   np.genfromtxt('/Applications/MongoDB.app/Contents/Resources/
Vendor/mongodb/all-data/csv/6.csv', delimiter=',', dtype='str')
loadermeta =
   np.genfromtxt('/Applications/MongoDB.app/Contents/Resources/
Vendor/mongodb/all-data/meta/all_sites.csv', delimiter=',',
   dtype='str')
row = (str(loadercsv[0][0])+','+str(loadercsv[0][1])+','
+str(loadercsv[0][2])+','+str(loadercsv[0][3])+','+str(loadercsv[0][4])+
','+str(loadermeta[0][0])+','+str(loadermeta[0][1])+','+
str(loadermeta[0][2])+','+str(loadermeta[0][3])+','+str(loadermeta[0][4])+
','+str(loadermeta[0][5])+','+str(loadermeta[0][6])
+','+str(loadermeta[0][7])+'\n')
newFile = open('merged.csv', 'a')
newFile.write(row)

x=1
for i in loadermeta:
    print str(loadermeta[x][0])
    loadcsv =
       np.genfromtxt('/Applications/MongoDB.app/Contents/Resources/
    Vendor/mongodb/all-data/csv/'+str(loadermeta[x][0])+'.csv',
    delimiter=',', dtype='str', skiprows=1)
    y=0
    for j in loadcsv:
        datarow =
           (str(loadcsv[y][0])+','+str(loadcsv[y][1])+','+
        str(loadcsv[y][2])+','+str(loadcsv[y][3])+','+str(loadcsv[y][4])+
        ','+str(loadermeta[x][0])+','+str(loadermeta[x][1])+','+
        str(loadermeta[x][2])+','+str(loadermeta[x][3])+','+
        str(loadermeta[x][4])+','+str(loadermeta[x][5])+','+
        str(loadermeta[x][6])+','+str(loadermeta[x][7])+'\n')
        newFile.write(datarow)
        y+=1
    x = x + 1
newFile.close() #close it after everything is done.
```

Finally to import the new csv file into mongoDB we use.

```
/Applications/MongoDB.app/Contents/Resources/Vendor/mongodb/bin/
mongoimport --type csv -d mergeddb -c merged  --file
   merged.csv --headerline
```

And to use it mongodb

```
use mergeddb
```

1.2 Data Modeling

MongoDB is a document oriented NoSQL, without implied relations between the tables. This makes mongoDB a very simple data base, but with a powerful model. It is also a good data base for scaling up.
MongoDB is structured as a series of collectioons that are divided in several documents. Documents are the ultimate data that we are interested on when using this data base. This documents are also divided into fields, which represent different characteristics of the document as a whole. This is not a relational data base, alghough we could do some similarities in regard to collections being similar to the tables or views, and documents being similar to rows. The storaging in MongoDB is done in BSON, which stands for binary JSON. It is a binary-encoded serialization of JSON-like documents. BSON supports the embedding of documents and arrays within other documents and arrays. It also contains extensions that allow representation of data types that are not part of the JSON spec. For example, BSON has a Date type and a BinData type.

BSON was designed to have the following three characteristics:

- Lightweight: Keeping spatial overhead to a minimum is important for any data representation format, especially when used over the network.
- Traversable: BSON is designed to be traversed easily. This is a vital property in its role as the primary data representation for MongoDB.
- Efficient: Encoding data to BSON and decoding from BSON can be performed very quickly in most languages due to the use of C data types.

In our case we chose to use mongoDB because the way we got the data. All the metadata files where rows with a series of fields. This structure just reminded a document kind, and we used it to store all our data. Also, when we where trying to use all the files as separate files, it was comfortable to use mongoDB to switch between the files. Once we decided to merge all the files, it was pretty fast doing it with mongoDB as all the documents had the same document like format.

2 Simple Queries

This section begins with the simple queries we did for our database which had different collections.

2.1 List Latitude by order

List 5 companies by latitude in descending order.

```
db.metadata.find().sort({LAT: -1}).limit(5);
```

Data retrieved:

```
{ "_id" : ObjectId("5b0ff5f6ab440be48f0d3dd4"), "SITE_ID" :
   14, "INDUSTRY" : "Commercial Property", "SUB_INDUSTRY" :
   "Business Services", "SQ_FT" : 1675720, "LAT" :
   47.82001743, "LNG" : -122.101407, "TIME_ZONE" :
   "America/Los_Angeles", "TZ_OFFSET" : "-07:00" }
{ "_id" : ObjectId("5b0ff5f6ab440be48f0d3dd9"), "SITE_ID" :
   30, "INDUSTRY" : "Commercial Property", "SUB_INDUSTRY" :
   "Bank/Financial Services", "SQ_FT" : 496517, "LAT" :
   47.29794461, "LNG" : -122.5435293, "TIME_ZONE" :
   "America/Los_Angeles", "TZ_OFFSET" : "-07:00" }
{ "_id" : ObjectId("5b0ff5f6ab440be48f0d3dde"), "SITE_ID" :
   44, "INDUSTRY" : "Commercial Property", "SUB_INDUSTRY" :
   "Shopping Center/Shopping Mall", "SQ_FT" : 192167, "LAT" :
   44.74241776, "LNG" : -73.58554074, "TIME_ZONE" :
   "America/New_York", "TZ_OFFSET" : "-04:00" }
{ "_id" : ObjectId("5b0ff5f6ab440be48f0d3e1f"), "SITE_ID" :
   703, "INDUSTRY" : "Light Industrial", "SUB_INDUSTRY" :
   "Food Processing", "SQ_FT" : 117651, "LAT" : 44.47177365,
   "LNG" : -75.61789475, "TIME_ZONE" : "America/New_York",
   "TZ_OFFSET" : "-04:00" }
{ "_id" : ObjectId("5b0ff5f6ab440be48f0d3e26"), "SITE_ID" :
   755, "INDUSTRY" : "Light Industrial", "SUB_INDUSTRY" :
   "Manufacturing", "SQ_FT" : 225974, "LAT" : 43.29647851,
   "LNG" : -71.72979539, "TIME_ZONE" : "America/New_York",
   "TZ_OFFSET" : "-04:00" }
```

2.2 Show metadata

Show the metadata and limit it to 5.

```
 db.metadata.find().limit(5);
```

Data retrieved:

```
{ "_id" : ObjectId("5b0ff5f6ab440be48f0d3dd0"), "SITE_ID" : 8,
   "INDUSTRY" : "Commercial Property", "SUB_INDUSTRY" :
   "Shopping Center/Shopping Mall", "SQ_FT" : 823966, "LAT" :
   40.32024733, "LNG" : -76.40494239, "TIME_ZONE" :
   "America/New_York", "TZ_OFFSET" : "-04:00" }
{ "_id" : ObjectId("5b0ff5f6ab440be48f0d3dd1"), "SITE_ID" : 6,
   "INDUSTRY" : "Commercial Property", "SUB_INDUSTRY" :
   "Shopping Center/Shopping Mall", "SQ_FT" : 161532, "LAT" :
   34.78300117, "LNG" : -106.8952497, "TIME_ZONE" :
   "America/Denver", "TZ_OFFSET" : "-06:00" }
{ "_id" : ObjectId("5b0ff5f6ab440be48f0d3dd2"), "SITE_ID" :
   12, "INDUSTRY" : "Commercial Property", "SUB_INDUSTRY" :
   "Business Services", "SQ_FT" : 179665, "LAT" : 39.69454093,
   "LNG" : -74.89916595, "TIME_ZONE" : "America/New_York",
   "TZ_OFFSET" : "-04:00" }
{ "_id" : ObjectId("5b0ff5f6ab440be48f0d3dd3"), "SITE_ID" :
   13, "INDUSTRY" : "Commercial Property", "SUB_INDUSTRY" :
   "Commercial Real Estate", "SQ_FT" : 185847, "LAT" :
   38.78465068, "LNG" : -77.47833675, "TIME_ZONE" :
   "America/New_York", "TZ_OFFSET" : "-04:00" }
{ "_id" : ObjectId("5b0ff5f6ab440be48f0d3dd4"), "SITE_ID" :
   14, "INDUSTRY" : "Commercial Property", "SUB_INDUSTRY" :
   "Business Services", "SQ_FT" : 1675720, "LAT" :
   47.82001743, "LNG" : -122.101407, "TIME_ZONE" :
   "America/Los_Angeles", "TZ_OFFSET" : "-07:00" }
```

2.3 Time zones

How many different time-zones are the companies from. Display the number of them and after that, show all the time-zones.

```
db.metadata.distinct("TIME\_ZONE").length;
```

Data retrieved:

```
5
```

```
db.metadata.distinct("TIME\_ZONE");
```

Data retrieved:

```
[
        "America/New\_York",
        "America/Denver",
        "America/Los\_Angeles",
        "America/Chicago",
        "America/Phoenix"
]
```

2.4 Sort, Limit and Skip

This query is going to merge three well know tools from mongoDB, sort, limit and skip. Retrieve the five companies after the top5 that have the biggest amount of square footage in their industries..

```
db.metadata.find().sort( {SQ_FT: 1 }).limit(5).skip(5);
```

Data retrieved:

```
{ "_id" : ObjectId("5b0ff5f6ab440be48f0d3e29"), "SITE_ID" :
   766, "INDUSTRY" : "Light Industrial", "SUB_INDUSTRY" :
   "Food Processing", "SQ_FT" : 12261, "LAT" : 43.00244098,
   "LNG" : -114.1137001, "TIME_ZONE" : "America/Denver",
   "TZ_OFFSET" : "-06:00" }
{ "_id" : ObjectId("5b0ff5f6ab440be48f0d3e1c"), "SITE_ID" :
   674, "INDUSTRY" : "Light Industrial", "SUB_INDUSTRY" :
   "Food Processing", "SQ_FT" : 12884, "LAT" : 38.04768265,
   "LNG" : -120.9719731, "TIME_ZONE" : "America/Los_Angeles",
   "TZ_OFFSET" : "-07:00" }
{ "_id" : ObjectId("5b0ff5f6ab440be48f0d3e24"), "SITE_ID" :
   742, "INDUSTRY" : "Light Industrial", "SUB_INDUSTRY" :
   "Food Processing", "SQ_FT" : 17717, "LAT" : 42.06854592,
   "LNG" : -71.2513348, "TIME_ZONE" : "America/New_York",
   "TZ_OFFSET" : "-04:00" }
{ "_id" : ObjectId("5b0ff5f6ab440be48f0d3e19"), "SITE_ID" :
   648, "INDUSTRY" : "Light Industrial", "SUB_INDUSTRY" :
   "Food Processing", "SQ_FT" : 20691, "LAT" : 34.64348922,
   "LNG" : -89.28865783, "TIME_ZONE" : "America/Chicago",
   "TZ_OFFSET" : "-05:00" }
{ "_id" : ObjectId("5b0ff5f6ab440be48f0d3e13"), "SITE_ID" :
   475, "INDUSTRY" : "Food Sales & Storage", "SUB_INDUSTRY" :
   "Grocer/Market", "SQ_FT" : 21505, "LAT" : 33.78885024,
   "LNG" : -118.3345952, "TIME_ZONE" : "America/Los_Angeles",
   "TZ_OFFSET" : "-07:00" }
```

2.5 Greater and lesser

This query shows the industries that has a square footage between 6001 and 8999feet.

```
db.metadata.find({SQ_FT:{$gt:6000, $lt:9000}})
```

Data retrieved:

```
{ "_id" : ObjectId("5b0ff5f6ab440be48f0d3e21"), "SITE_ID" :
    731, "INDUSTRY" : "Light Industrial", "SUB_INDUSTRY" :
    "Food Processing", "SQ_FT" : 6236, "LAT" : 41.11320068,
    "LNG" : -81.7077033, "TIME_ZONE" : "America/New_York",
    "TZ_OFFSET" : "-04:00" }
{ "_id" : ObjectId("5b0ff5f6ab440be48f0d3e2a"), "SITE_ID" :
    765, "INDUSTRY" : "Light Industrial", "SUB_INDUSTRY" :
    "Other Light Industrial", "SQ_FT" : 7394, "LAT" :
    40.02768022, "LNG" : -80.45340754, "TIME_ZONE" :
    "America/New_York", "TZ_OFFSET" : "-04:00" }
```

3 New queries for the big file

Here are some extra simple queries we did after we had merged all the files into one.

3.1 Sorting the timestamps in descending order

This query is to show the all the information of each of the four first sites but for the timestamp.

```
db.merged.find().sort( {timestamp: 1 } ).limit(4)
```

data retrieved:

```
{ "_id" : ObjectId("5b2511c46abc8e836c2d2e45"), "timestamp" :
    1325376300, "dttm_utc" : "2012-01-01 00:05:00", "value" :
    17.9272, "estimated" : 0, "anomaly" : "", "SITE_ID" : 9,
    "INDUSTRY" : "Commercial Property", "SUB_INDUSTRY" :
    "Corporate Office", "SQ_FT" : 169420, "LAT" : 40.94675135,
    "LNG" : -74.74208726, "TIME_ZONE" : "America/New_York",
    "TZ_OFFSET" : "-04:00" }
{ "_id" : ObjectId("5b2511c96abc8e836c30669c"), "timestamp" :
    1325376300, "dttm_utc" : "2012-01-01 00:05:00", "value" :
    27.0053, "estimated" : 0, "anomaly" : "", "SITE_ID" : 12,
    "INDUSTRY" : "Commercial Property", "SUB_INDUSTRY" :
    "Business Services", "SQ_FT" : 179665, "LAT" : 39.69454093,
    "LNG" : -74.89916595, "TIME_ZONE" : "America/New_York",
    "TZ_OFFSET" : "-04:00" }
{ "_id" : ObjectId("5b2511cf6abc8e836c339eec"), "timestamp" :
    1325376300, "dttm_utc" : "2012-01-01 00:05:00", "value" :
    136.7605, "estimated" : 0, "anomaly" : "", "SITE_ID" : 14,
    "INDUSTRY" : "Commercial Property", "SUB_INDUSTRY" :
    "Business Services", "SQ_FT" : 1675720, "LAT" :
    47.82001743, "LNG" : -122.101407, "TIME_ZONE" :
    "America/Los_Angeles", "TZ_OFFSET" : "-07:00" }
```

```
{ "_id" : ObjectId("5b2511db6abc8e836c3b9e70"), "timestamp" :
   1325376300, "dttm_utc" : "2012-01-01 00:05:00", "value" :
   7.3969, "estimated" : 0, "anomaly" : "", "SITE_ID" : 30,
   "INDUSTRY" : "Commercial Property", "SUB_INDUSTRY" :
   "Bank/Financial Services", "SQ_FT" : 496517, "LAT" :
   47.29794461, "LNG" : -122.5435293, "TIME_ZONE" :
   "America/Los_Angeles", "TZ_OFFSET" : "-07:00" }
```

3.2 Does any site share the same square footage

Retrieve the information of how many different square footage there are in the database.

```
> db.merged.distinct("SQ_FT").length
100
```

The result shows that every site has as a unique square footage.

3.3 Commercial Property sorted by timestamp

Retrieve the information of the company descending in timestamp in the "Commercial Property" industry.

```
db.merged.find({"INDUSTRY" : "Commercial Property"}).sort(
   {timestamp: -1}).limit(3)
```

data retreived:

```
{ "_id" : ObjectId("5b2511b96abc8e836c26bdba"), "timestamp" :
   1356998400, "dttm_utc" : "2013-01-01 00:00:00", "value" :
   56.6561, "estimated" : 0, "anomaly" : "", "SITE_ID" : 6,
   "INDUSTRY" : "Commercial Property", "SUB_INDUSTRY" :
   "Shopping Center/Shopping Mall", "SQ_FT" : 161532, "LAT" :
   34.78300117, "LNG" : -106.8952497, "TIME_ZONE" :
   "America/Denver", "TZ_OFFSET" : "-06:00" }
{ "_id" : ObjectId("5b2511bc6abc8e836c2859e2"), "timestamp" :
   1356998400, "dttm_utc" : "2013-01-01 00:00:00", "value" :
   27.9357, "estimated" : 0, "anomaly" : "", "SITE_ID" : 8,
   "INDUSTRY" : "Commercial Property", "SUB_INDUSTRY" :
   "Shopping Center/Shopping Mall", "SQ_FT" : 823966, "LAT" :
   40.32024733, "LNG" : -76.40494239, "TIME_ZONE" :
   "America/New_York", "TZ_OFFSET" : "-04:00" }
{ "_id" : ObjectId("5b2511be6abc8e836c29f60c"), "timestamp" :
   1356998400, "dttm_utc" : "2013-01-01 00:00:00", "value" :
   25.7216, "estimated" : 0, "anomaly" : "", "SITE_ID" : 9,
   "INDUSTRY" : "Commercial Property", "SUB_INDUSTRY" :
   "Corporate Office", "SQ_FT" : 169420, "LAT" : 40.94675135,
   "LNG" : -74.74208726, "TIME_ZONE" : "America/New_York",
   "TZ_OFFSET" : "-04:00" }
```

3.4 Different type of industries

This query shows how many different types of industires there are and there names.

```
> db.merged.distinct("INDUSTRY").length
4
> db.merged.distinct("INDUSTRY")
[
        "Commercial Property",
        "Education",
        "Food Sales & Storage",
        "Light Industrial"
]
```

3.5 Showing one object

This query shows of how one object in our database is defined.

```
db.merged.findOne()
{
        "_id" : ObjectId("5b2511b76abc8e836c2521a9"),
        "timestamp" : 1325376900,
        "dttm_utc" : "2012-01-01 00:15:00",
        "value" : 50.9517,
        "estimated" : 0,
        "anomaly" : "",
        "SITE_ID" : 6,
        "INDUSTRY" : "Commercial Property",
        "SUB_INDUSTRY" : "Shopping Center/Shopping Mall",
        "SQ_FT" : 161532,
        "LAT" : 34.78300117,
        "LNG" : -106.8952497,
        "TIME_ZONE" : "America/Denver",
        "TZ_OFFSET" : "-06:00"
}
```

4 Mandatory Queries

This section starts out with showing where we first noticed that we might need to use another datamodel. This is followed by our final solution for the mandatory queries.

4.1 Sum LD (5 min interval) first try

Every timestamp has an interval of 300sec which is 5 min. In our very first try we thought that we were supposed to calculate the sum for every site one by one.
We had a print in our for loop, and notice that everything came out as [Object object], so we changed the the print to printjson and with a JSON.stringify we noticed the information was in _batch

```
printjson(JSON.stringify(db.data6.aggregate([{$group:{_id:null,
totalcount:{$sum:"$value"}}}])))
{
        "_batch" : [
                {
                        "_id" : null,
                        "totalcount" : 3273816.8995
                }
        ],
}
```

So in the printjson in the for loop we added ._batch
The query used for this is showed below.

```
> db.getCollectionNames().forEach(function(collname) {
    printjson(db[collname].aggregate([{$group:{_id:[collname],
       totalcount:{$sum:"$value"}}}])._batch)
});
```

Data retrieved:

```
[ { "_id" : [ "data10" ], "totalcount" : 33544015.7803 } ]
[ { "_id" : [ "data100" ], "totalcount" : 670109.7884 } ]
[ { "_id" : [ "data101" ], "totalcount" : 3630818.6238 } ]
[ { "_id" : [ "data103" ], "totalcount" : 1918911.3582 } ]
[ { "_id" : [ "data109" ], "totalcount" : 1183644.5623 } ]
[ { "_id" : [ "data111" ], "totalcount" : 821927.0672 } ]
[ { "_id" : [ "data116" ], "totalcount" : 1968173.6505 } ]
[ { "_id" : [ "data12" ], "totalcount" : 4164841.9023 } ]
```

This is the result we got the first time we tried to solve this exercice. It actually calculates all the energy consumed by each of the sites. This is not what we want.

We started with a new possible solution, the pseudocode would look something like this.

```
For every row in data6
    get timestamp of row with skip
    for every collection
        sum the value of that row with the timestamp
```

And the final procuct look like this.

```
db.data6.find().forEach(function(row){
    var sum = 0;
db.getCollectionNames().forEach(function(collname){
     var sum = sum + db[collname].findOne( {"timestamp" :
        row.timestamp} ).value
    })
print("sum for row " + JSON.stringify(row.timestamp) + " is "
   + sum);
});
```

We noticed that the timestamp was not the same for every site. We got the advice to merge all the data into one file instead. In the next section

4.2 Sum LD (5 min interval) second try

This time we did it with the collection merged. This query works for finding the sum for one specific timestamp, in this case the first timestamp.

```
db.merged.aggregate([{ $match:{timestamp: 1325376900 }},{
   $group: {_id:{timestamp:"1325376900"}
   ,totalLD:{$sum:"$value"}, NrOfSites:{$sum:1} } }])
```

data retreived:

```
{ "_id" : { "timestamp" : "1325376900" }, "totalLD" :
   2716.4764, "NrOfSites" : 101 }
```

We use aggregate with the match on timestamp. We choose to show the result with the id as timestamp. The sum for is counting the value of each timestamp which matches with the one in the inside match. We also chose to show how many sites that was found for that timestamp.

We decided to do a for loop calculating all the data. We did this with a print so you could follow along while the function where computing all the data for every different timestamp.

```
db.merged.distinct('timestamp').forEach(function(timestamp){
    printjson(db.merged.aggregate([{ $match:{timestamp:
       timestamp }},{ $group: {_id:{timestamp: "$timestamp"}
       ,totalLD:{$sum:"$value"}, NrOfSites:{$sum:1} }
       }])._batch)
});


[
        {
                "_id" : {
                        "timestamp" : 1325376900
                },
                "totalLD" : 2716.4764,
                "NrOfSites" : 101
        }
]
[
        {
                "_id" : {
                        "timestamp" : 1325376600
                },
                "totalLD" : 2735.1286,
                "NrOfSites" : 101
        }
]
[
        {
                "_id" : {
                        "timestamp" : 1325377500
                },
                "totalLD" : 2687.1919,
                "NrOfSites" : 101
        }
]
[
        {
                "_id" : {
                        "timestamp" : 1325377800
                },
                "totalLD" : 2640.7312,
                "NrOfSites" : 101
        }
]
```

4.3 Average LD (5 min interval)

The average LD by sector of activity with a timestamp interval of 5 minutes was calculated with the following query:

```
> db.merged.aggregate([{ $match:{timestamp: 1325376900 }},{
    $group: {_id:{sector: "$INDUSTRY",
    timestamp:"$timestamp"},totalLD:{$sum:"$value"},
    averageLD:{$avg:"$value"}, NrOfSites:{$sum:1} } }])


{ "_id" : { "sector" : "Light Industrial", "timestamp" :
    1325376900 }, "totalLD" : 970.4796, "averageLD" : 40.43665,
    "NrOfSites" : 24 }
{ "_id" : { "sector" : "Food Sales & Storage", "timestamp" :
    1325376900 }, "totalLD" : 482.9139, "averageLD" :
    19.316556000000002, "NrOfSites" : 25 }
{ "_id" : { "sector" : "Education", "timestamp" : 1325376900
    }, "totalLD" : 161.2969, "averageLD" : 6.4518759999999995,
    "NrOfSites" : 25 }
{ "_id" : { "sector" : "Commercial Property", "timestamp" :
    1325376900 }, "totalLD" : 1101.786, "averageLD" :
    40.80688888888889, "NrOfSites" : 27 }
```

The only thing we added from the last one was the averageLD which is the avg from value instead of the sum. We also added the ID to also show which sector. We choose to keep the total LD, and also the number of sites. We did the query inside a loop like last time.

```
> db.merged.distinct('timestamp').forEach(function(timestamp){
        printjson(db.merged.aggregate([{ $match:{timestamp:
            timestamp }},{ $group: {_id:{sector: "$INDUSTRY",
            timestamp:"$timestamp"},totalLD:{$sum:"$value"},
            averageLD:{$avg:"$value"}, NrOfSites:{$sum:1} }
            }])._batch)
});


[
        {
                "_id" : {
                        "sector" : "Commercial Property",
                        "timestamp" : 1325376900
                },
                "totalLD" : 1101.786,
                "averageLD" : 40.80688888888889,
                "NrOfSites" : 27
        },
        {
                "_id" : {
                        "sector" : "Education",
                        "timestamp" : 1325376900
                },
                "totalLD" : 161.2969,
                "averageLD" : 6.4518759999999995,
                "NrOfSites" : 25
        },
        {
                "_id" : {
                        "sector" : "Food Sales & Storage",
                        "timestamp" : 1325376900
                },
                "totalLD" : 482.9139,
                "averageLD" : 19.316556000000002,
                "NrOfSites" : 25
        },
        {
                "_id" : {
                        "sector" : "Light Industrial",
                        "timestamp" : 1325376900
                },
                "totalLD" : 970.4796,
                "averageLD" : 40.43665,
                "NrOfSites" : 24
        }
]
[
        {
                "_id" : {
                        "sector" : "Commercial Property",
                        "timestamp" : 1325376600
                },
                "totalLD" : 1110.7974,
                "averageLD" : 41.14064444444444,
                "NrOfSites" : 27
        },
        {
                "_id" : {
                        "sector" : "Education",
                        "timestamp" : 1325376600
                },
                "totalLD" : 161.7156,
                "averageLD" : 6.468624,
                "NrOfSites" : 25
        },
```

4.4 Total LD (1 week interval)

The total LD for the 100 sites with a timestamp interval of 1 week was calculated with the following query: The timestamps are showned with an interval of 5 minutes or 300 seconds, to make it weekly instead, we calculated how many seconds there are in one week. 60 * 60 * 24 * 7 = 604800seconds in one week. so if we started from 1325376900 we used a match from 1325376900 to 1325376900 + 604800 = 1325981700.

```
> db.merged.aggregate([{ $match:{timestamp:{$gte: 1325376900,
   $lt: 1325981700}}},{ $group: {_id:''
   ,totalLD:{$sum:"$value"}, NrOfSites:{$sum:1} }}])
{ "_id" : "", "totalLD" : 9086501.0347, "NrOfSites" : 203616 }
```

We tried to do it with looping through the different weeks as well. We the value shown are the total for the week, the calculation starts with five minutes timestamp intervals.

```
> db.merged.distinct('timestamp').forEach(function(timestamp){
... printjson(db.merged.aggregate([{ $match:{timestamp: {$gte:
   timestamp,$lt: timestamp+604800}}},{ $group: {_id:''
   ,totalLD:{$sum:"$value"}, NrOfSites:{$sum:1} }}])._batch);
... });
[ { "_id" : "", "totalLD" : 9086501.0347, "NrOfSites" : 203616
   } ]
[ { "_id" : "", "totalLD" : 9085640.3387, "NrOfSites" : 203616
   } ]
[ { "_id" : "", "totalLD" : 9088164.8648, "NrOfSites" : 203616
   } ]
[ { "_id" : "", "totalLD" : 9089182.508, "NrOfSites" : 203616
   } ]
[ { "_id" : "", "totalLD" : 9087301.5128, "NrOfSites" : 203616
   } ]
[ { "_id" : "", "totalLD" : 9090223.0895, "NrOfSites" : 203616
   } ]
[ { "_id" : "", "totalLD" : 9091284.4833, "NrOfSites" : 203616
   } ]
[ { "_id" : "", "totalLD" : 9092321.529, "NrOfSites" : 203616
   } ]
[ { "_id" : "", "totalLD" : 9093341.6846, "NrOfSites" : 203616
   } ]
```

4.5 Average LD (1 week interval)

The average LD by sector of activity with a timestamp interval of 1 week was calculated with the following query:

Here the id is set for sector, we also added the averageLD

```
> db.merged.aggregate([{ $match:{timestamp: {$gte:
   1325376900,$lt: 1325981700}}},{ $group: {_id:{sector:
   "$INDUSTRY"} ,totalLD: {$sum:"$value"}, averageLD:
   {$avg:"$value"}, NrOfSites: {$sum:1} } }]);

{ "_id" : { "sector" : "Light Industrial" }, "totalLD" :
   3471810.5766, "averageLD" : 71.75534425843253, "NrOfSites"
   : 48384 }
{ "_id" : { "sector" : "Food Sales & Storage" }, "totalLD" :
   853349.2434, "averageLD" : 16.93153260714286, "NrOfSites" :
   50400 }
{ "_id" : { "sector" : "Education" }, "totalLD" :
   505669.08150000003, "averageLD" : 10.033116696428571,
   "NrOfSites" : 50400 }
{ "_id" : { "sector" : "Commercial Property" }, "totalLD" :
   4255672.1332, "averageLD" : 78.18327699147561, "NrOfSites"
   : 54432 }
```

We did a loop for this one to.

```
> db.merged.distinct('timestamp').forEach(function(timestamp){
printjson(db.merged.aggregate([{ $match:{timestamp: {$gte:
   timestamp,$lt: timestamp+604800}}},{ $group: {_id:{sector:
   "$INDUSTRY"} ,totalLD:{$sum:"$value"}, averageLD:
   {$avg:"$value"}, NrOfSites:{$sum:1} } }])._batch);
});
[
        {
                "_id" : {
                        "sector" : "Commercial Property"
                },
                "totalLD" : 4255672.1332,
                "averageLD" : 78.18327699147561,
                "NrOfSites" : 54432
        },
        {
                "_id" : {
                        "sector" : "Education"
                },
                "totalLD" : 505669.08150000003,
                "averageLD" : 10.033116696428571,
                "NrOfSites" : 50400
        },
        {
                "_id" : {
                        "sector" : "Food Sales & Storage"
                },
                "totalLD" : 853349.2434,
                "averageLD" : 16.93153260714286,
                "NrOfSites" : 50400
        },
        {
                "_id" : {
                        "sector" : "Light Industrial"
                },
                "totalLD" : 3471810.5766,
                "averageLD" : 71.75534425843253,
                "NrOfSites" : 48384
        }
]
[
        {
                "_id" : {
                        "sector" : "Commercial Property"
                },
                "totalLD" : 4255233.022,
                "averageLD" : 78.17520983980012,
                "NrOfSites" : 54432
        },
        {
                "_id" : {
                        "sector" : "Education"
                },
                "totalLD" : 505672.3867,
                "averageLD" : 10.033182275793651,
                "NrOfSites" : 50400
        },
```