Chapter 3
Storing Twitter Data

In the previous chapter, we covered data collection methodologies. Using these methods, one can quickly amass a large volume of Tweets, Tweeters, and network information. Managing even a moderately-sized dataset is cumbersome when storing data in a text-based archive, and this solution will not give the performance needed for a real-time application. In this chapter we present some common storage methodologies for Twitter data using NoSQL.

3.1 NoSQL Through the Lens of MongoDB

Keeping track of every purchase, click, and "like" has caused the data needs of many companies to double every 14 months. There has been an explosion in the size of data generated on social media. This data explosion calls for a new data storage paradigm. At the forefront of this movement is NoSQL [3], which promises to store big data in a more accessible way than the traditional, relational model.

There are several NoSQL implementations. In this book, we choose MongoDB[1] as an example NoSQL implementation. We choose it for its adherence to the following principles:

- **Document-Oriented Storage.** MongoDB stores its data in JSON-style objects. This makes it very easy to store raw documents from Twitter's APIs.
- **Index Support.** MongoDB allows for indexes on any field, which makes it easy to create indexes optimized for your application.
- **Straightforward Queries.** MongoDB's queries, while syntactically much different from SQL, are semantically very similar. In addition, MongoDB supports MapReduce, which allows for easy lookups in the data.

[1]http://www.mongodb.org/

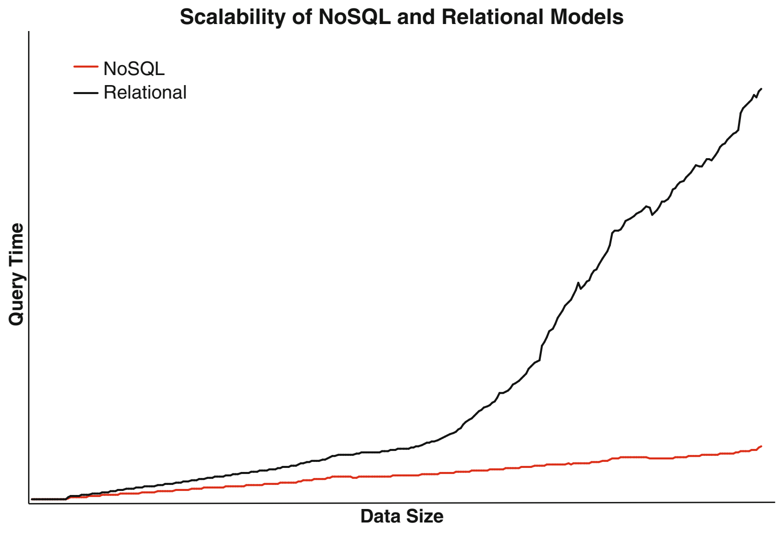

Fig. 3.1 Comparison of traditional relational model with NoSQL model. As data grows to a large capacity, the NoSQL database outpaces the relational model

- **Speed.** Figure 3.1 shows a comparison of query speed between the relational model and MongoDB.

In addition to these abilities, it also works well in a single-instance environment, making it easy to set up on a home computer and run the examples in this chapter.

3.2 Setting Up MongoDB on a Single Node

The most simple configuration of MongoDB is a single instance running on one machine. This setup allows for access to all of the features of MongoDB. We use MongoDB 2.4.4,[2] the latest version at the time of this writing.

3.2.1 Installing MongoDB on Windows®

1. Obtain the latest version of MongoDB from http://www.mongodb.org/downloads. Extract the downloaded `zip` file.
2. Rename the extracted folder to `mongodb`.
3. Create a folder called `data` next to the `mongodb` folder.

[2] http://docs.mongodb.org/manual/

4. Create a folder called `db` within the `data` folder. Your file structure should reflect that shown below.

```
  /
  ├── mongodb
  │   └── ...
  └── data
      └── db
```

3.2.2 Running MongoDB on Windows

1. Open the command prompt and move to the directory above the `mongodb` folder.
2. Run the command `mongodb\bin\mongod.exe-dbpath data\db`
3. If Windows prompts you, make sure to allow MongoDB to communicate on private networks, but not public ones. Without special precautions, MongoDB should not be run in an open environment.
4. Open another command window and move to the directory where you put the `mongodb` folder.
5. Run the command `mongodb\bin\mongo.exe`. This is the command-line interface to MongoDB. You can now issue commands to MongoDB.

3.2.3 Installing MongoDB on Mac OS X®

1. Obtain the latest version of MongoDB from http://www.mongodb.org/downloads.
2. Rename the downloaded file to `mongodb.tgz`.
3. Open the "Terminal" application. Move to the folder where you downloaded MongoDB.
4. Run the command `tar -zxvf mongodb.tgz`. This will create a folder with the name `mongodb-osx-[platform]-[version]` in the same directory. For version 2.4.4, this folder will be called `mongodb-osx-x86_64-2.4.4`.
5. Run the command `mv -n mongodb-osx-[platform]-[version]/ mongodb`. This will give us a more convenient folder name.
6. Run the command `mkdir data && mkdir data/db`. This will create the subfolders where we will store our data.

3.2.4 Running MongoDB on Mac OS X

1. Open the "Terminal" application and move to the directory above the `mongodb` folder.
2. Run the command `./mongodb/bin/mongod-dbpath data/db`
3. Open another tab in Terminal (Cmd-T).
4. Run the command `./mongodb/bin/mongo`. This is the command-line interface to MongoDB. You can now issue commands to MongoDB.

3.3 MongoDB's Data Organization

MongoDB organizes its data in the following hierarchy: database, collection, document. A database is a set of collections, and a collection is a set of documents. The organization of data in MongoDB is shown in Fig. 3.2. Here we will demonstrate how to interact with each level in this hierarchy to store data.

3.4 How to Execute the MongoDB Examples

The examples presented in this chapter are written in JavaScript – the language underpinning MongoDB. To run these examples, do the following:

1. Run `mongod`, as shown above. The process doing this varies is outlined in Sect. 3.2.

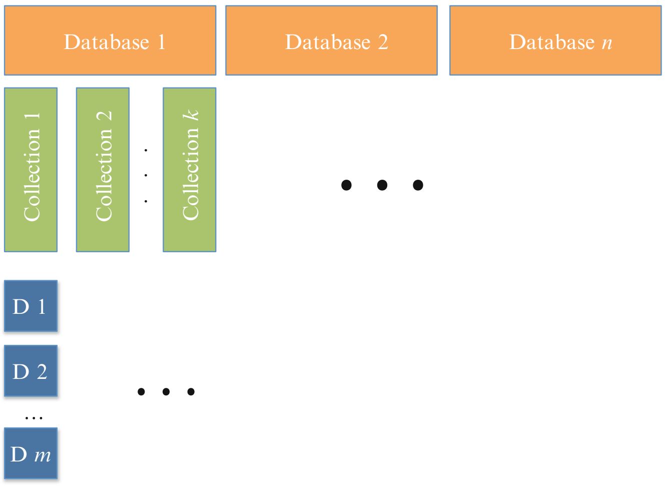

Fig. 3.2 Organization of MongoDB data

2. Change directories to your `bin` folder: `cd mongodb/bin`.
3. Execute the following command: `mongo localhost/tweetdata path/to/example.js`. This will run the example on your local MongoDB installation. If you are on windows, you will have to replace `mongo` with `mongo.exe`.

3.5 Adding Tweets to the Collection

Now that we have a collection in the database, we will add some Tweets to it. Because MongoDB uses JSON to store its documents, we can import the data *exactly* as it was collected from Twitter, with no need to map columns. To load this, download the Occupy Wall Street data included in the supplementary materials, `ows.json`. Next, with mongod running, issue the following command[3]:

```
mongoimport -d tweetdata -c tweets -file ows.json
```

`mongoimport` is a utility that is packaged with MongoDB that allows you to import JSON documents. By running the above command, we have added all of the JSON documents in the file to the collection we created earlier. We now have some Tweets stored in our database, and we are ready to issue some commands to analyze this data.

3.6 Optimizing Collections for Queries

To make our documents more accessible, we will extract some key features for indexing later. For example, while the "created_at" field gives us information about a date in a readable format, converting it to a JavaScript date each time we do a date comparison will add overhead to our computations. It makes sense to add a field "timestamp" whose value contains the Unix timestamp[4] representing the information contained in "created_at". This redundancy trades disk space for efficient computation, which is more of a concern when building real-time applications which rely on big data. Listing 3.1 is a post-processing script that adds fields that make handling the Twitter data more convenient and efficient.

[3]On Windows, you exchange `mongoimport` with `mongoimport.exe`.

[4]A number, the count of milliseconds since January 1st, 1970.

Listing 3.1 Post-processing step to add extra information to data

```
> //enumerate each Tweet
> db.tweets.find().forEach(function(doc){
...     //save the time string in Unix time.
...     doc.timestamp = +new Date(doc.created_at);
...     //reduce the geobox to one point
...     doc.geoflag = !!doc.coordinates;
...     if(doc.coordinates && doc.coordinates.coordinates){
...         doc.location = {"lat": doc.coordinates.coordinates
    [1], "lng": doc.coordinates.coordinates[0]};
...     }
...     //save a lowercased version of the screen name
...     doc.screen_name_lower = doc.user.screen_name.toLowerCase
    ();
...     //save our modifications
...     db.tweets.save(doc);
... });
Source: Chapter3/postProcessingExample.js
```

Listing 3.2 Create an index on the “timestamp” field

```
> db.tweets.ensureIndex({"timestamp": 1})
```

3.7 Indexes

We now have inserted some documents into a collection, but as they stand querying them will be slow as we have not created any indexes. That is, we have not told MongoDB which fields in the document to optimize for faster lookup.

One of the most important concepts to understand for fast access of a MongoDB collection is indexing. The indexes you choose will depend largely on the queries that you run often, those that *must* be executed in real time. While the indexes you choose will depend on your data, here we will show some indexes that are often useful in querying Twitter data in real-time.

The first index we create will be on our “timestamp” field. This command is shown in Listing 3.2.

When creating an index, there are several rules MongoDB enforces to ensure that an index is used:

- **Only one index is used per query.** While you can create as many indexes as you want for a given collection, you can only use one for each query. If you have multiple fields in your query, you can create a “compound index” on both fields. For example, if you want to create an index on “timestamp”, and then “retweet_count”, can pass `{"timestamp": 1, "retweet_count": 1}`.

- **Indexes can only use fields in the order they were created.** Say, for example, we create the index `{"timestamp": 1, "retweet_count": 1, "keywords" : 1}`.
  This query is valid for queries structured in the following order:
  - timestamp, retweet_count, keywords
  - timestamp
  - timestamp, retweet_count

  This query is **not** valid for queries structured in the following order:
  - retweet_count, timestamp, keywords
  - keywords
  - timestamp, keywords
- **Indexes can contain, at most, one array.** Twitter provides Tweet metadata in the form of arrays, but we can only use one in any given index.

3.8 Extracting Documents: Retrieving All Documents in a Collection

The simplest query we can provide to MongoDB is to return all of the data in a collection. We use MongoDB's `find` function to do this, an example of which is shown in Listing 3.3.

3.9 Filtering Documents: Number of Tweets Generated in a Certain Hour

Suppose we want to know the number of Tweets in our dataset from a particular hour. To do this we will have to filter our data by the `timestamp` field with "operators": special values that act as functions in retrieving data.

Listing 3.4 shows how we can drill down to extract data only from this hour. We use the `$gt` ("greater than"), and `$lte` ("less than or equal to") operators to pull dates from this time range. Notice that there is no explicit "AND" or "OR" operator specified. MongoDB treats all co-occurring key/value pairs as "AND"s unless explicitly specified by the `$or` operator.[5] Finally, the result of this query is passed to the `count` function, which returns the number of documents returned by the `find` function.

[5]For more operators, see http://docs.mongodb.org/manual/reference/operator/.

Listing 3.3 Get all of the Tweets in a collection

```
> db.tweets.find()
{ "_id" : ObjectId("51e6d70cd13954bd0dd9e09d"), ... }
{ "_id" : ObjectId("51e6d70cd13954bd0dd9e09e"), ... }
...
has more
Source: Chapter3/find_all_tweets.js
```

Listing 3.4 Get all of the Tweets from a single hour

```
> var NOVEMBER = 10; //Months are zero-indexed.
> var query = {
...     "timestamp" : {
...         "$gte": +new Date(2011, NOVEMBER, 15, 10),
...         "$lt": +new Date(2011, NOVEMBER, 16, 11)
...     }
... };
> db.tweets.find(query).count();
22169
Source: Chapter3/tweets_from_one_hour.js
```

Listing 3.5 Sort Tweets by time published

```
> db.tweets.find().sort({"timestamp": -1})
{ "_id" : ObjectId("51e6d713d13954bd0ddaa097"), ... }
{ "_id" : ObjectId("51e6d713d13954bd0ddaa096"), ... }
has more
Source: Chapter3/most_recent_tweets.js
```

3.10 Sorting Documents: Finding the Most Recent Tweets

To find the most recent Tweets, we will have to sort the data. MongoDB provides a `sort` function that will order the Tweet by a specified field. Listing 3.5 shows an example of how to use `sort` to order data by timestamp. Because we used "−1" in the value of the key value pair, MongoDB will return the data in descending order. For ascending order, use "1".

Without the index created in Sect. 3.7, we would have caused the error shown in Listing 3.6. Even with a relatively small collection, MongoDB cannot sort the data in a manageable amount of time, however with an index it is very fast.

Listing 3.6 Error generated without an index on "timestamp"

```
> db.tweets.find().sort({"timestamp": -1})
error: {
  "$err" : "too much data for sort() with no index.  add an
      index or specify a smaller limit",
  "code" : 10128
}
```

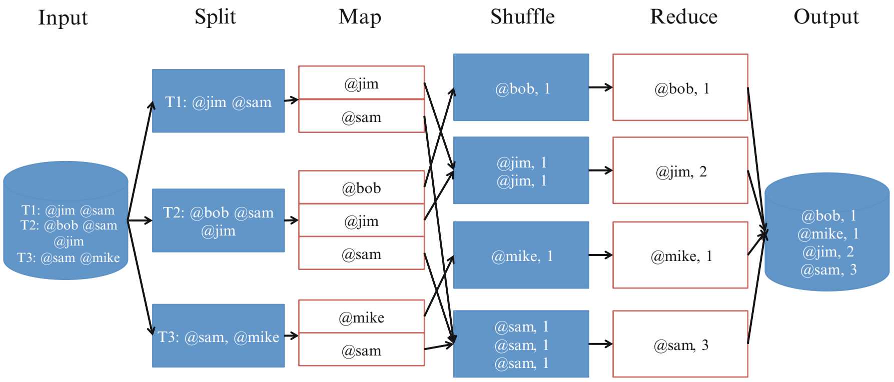

Fig. 3.3 MapReduce framework. The steps in white are implemented by the reader. MongoDB takes the documents from the database and runs each one through the map function. It then sorts the emitted keys and runs each key and its values through the reduce function. The output from the reduce function is stored in another collection

3.11 Grouping Documents: Identifying the Most Mentioned Users

With some simple use of the find and count functions, you can learn volumes about the data you have collected. However, when it comes to aggregating data, we will need to employ another set of functions, collectively called MapReduce.

MapReduce consists of two steps: "Map", and "Reduce". In the map step, data is extracted, filtered, and processed to be sent to the reduce function. The mapper processes in the map step emit a series of key/value pairs. These key value pairs are sorted, and the values associated with each unique key are sent to a reduce process. Each reduce process then computes a value for the key it is handed. A diagram for this process is shown in Fig. 3.3. The MongoDB code for this example is shown in Listing 3.7.

Listing 3.7 MapReduce function that lists the most mentioned users

```
> /*
> * This function extracts each user mentioned,
> * and the count of each mention.
> * The function takes 0 parameters, as the document
> * will be passed through context (the 'this' object).
> */
> var mapFunction = function(){
...     //loop through all of the mentions in the document.
...     var userMentions = this.entities.user_mentions;
...     for(var i = 0; i < userMentions.length; i++){
...         //check that the username is not blank.
...         if(userMentions[i].screen_name.length > 0){
...             //emit the username (key) and
...             //the count (value, in this case always 1).
...             emit(userMentions[i].screen_name, 1);
...         }
...     }
... }

> /*
> * This function sums the number of mentions of each user
> */
> var reduceFunction = function(keyUsername, occurs){
...     return Array.sum(occurs);
... }

> // Perform the MapReduce operation, and store the results
> // in a new collection, "most_mentioned_users".
> db.tweets.mapReduce(mapFunction, reduceFunction, {"out": "
    most_mentioned_users"});

> // List the top 5 most-mentioned users
> db.most_mentioned_users.find().sort({"value": -1}).limit(5)
{ "_id" : "MikeBloomberg", "value" : 727 }
{ "_id" : "OccupyWallSt", "value" : 588 }
{ "_id" : "OccupyWallStNYC", "value" : 428 }
{ "_id" : "JoshHarkinson", "value" : 295 }
{ "_id" : "ydanis", "value" : 260 }
```

Source: Chapter3/mapreduce.js

In Listing 3.7, the MapReduce is constructed as follows. The map function, called `mapFunction`, looks at each individual Tweet and pulls out the mentioned users. It then constructs the key/value pair to be sent to the reducer. The key is the user that was mentioned, and the value is 1. MongoDB then creates a unique reducer for each unique key and calls the reduce function, `reduceFunction`, on each key. The reducer then takes this list of values and calculates the sum. The result is a list of mentioned users and the count of the number of mentions for that user.

3.12 Further Reading

More information on MongoDB can be found in [2] and the MongoDB specification [1]. For more conversions between MongoDB's document-based syntax and SQL, see http://docs.mongodb.org/manual/reference/sql-comparison/. More information on other NoSQL implementations can be found in [3].

References

1. 10gen. The mongodb 2.4 manual. http://docs.mongodb.org/manual/, 2013.
2. K. Chodorow. *MongoDB: the definitive guide*. O'Reilly, 2013.
3. E. Redmond and J. R. Wilson. *Seven Databases in Seven Weeks*. Pragmatic Programmers, 2012.