CS4417 Assignment 3 - MongoDB, Aggregation, and MapReduce

16 March 2024

The goal of this assignment is for you to gain familiarity with MongoDB, one of the most widely-used tools for the management and querying of big, unstructured data.

MongoDB uses the JSON (and related BSON) format for data, as we saw in class.
The MongoDB shell provides an interactive JavaScript interface to MongoDB.

The mongodb manual is here: https://www.mongodb.com/docs/

Installing MongoDB

There are various ways of installing MongoDB on different systems. You can use any approach you want. We are providing detailed instructions using `docker` which should work in a platform-independent fashion. You will need at least a few GB of free hard drive space to install.

If Installing using Docker on your Own Machine

Windows alert: If after installing docker desktop and restarting the PC you get a message that states, "WSL 2 installation is incomplete," then going to https://docs.microsoft.com/en-us/windows/wsl/install-win10#step-4---download-the-linux-kernel-update-package and installing `wsl_update_x64` and then restarting the PC should fix this problem.

First, install https://www.docker.com/ Docker Desktop or Engine. We'll use the Docker command line https://docs.docker.com/engine/reference/commandline/docker/ to install mongodb.

Make sure docker has fully started before issuing the following commands. Then, from a command prompt on your machine:

Download a docker image containing mongodb.

```
docker pull mongo:6.0
```

Create a new container from this image.

```
docker run --name my-mongo -d mongo:6.0
```

Running `docker ps` should show something like

```
    CONTAINER ID   IMAGE       COMMAND                  CREATED          STATUS
PORTS       NAMES
    4a442375f0ff   mongo:6.0   "docker-entrypoint.s…"   32 seconds ago   Up 32
seconds   27017/tcp   my-mongo
```

You can now start a shell from within your docker container like so:

```
docker exec -it my-mongo bash
```

If your docker container is stopped for whatever reason after you do the original `docker run`, you can start it again with

```
docker start my-mongo
```

You can also use the Dashboard of the Docker Desktop application. If you run into trouble with docker and the `docker` commands stop working, you can re-start it using "Restart Docker" in the Docker Desktop application. You can also use it to see all of the containers you've created and delete any that you don't need.

Copying Tweets for Import

We will use the same `tweets-utf-8.json` file as for assignment 2.

Docker on own machine

If you are using docker, copy the `tweets-utf-8.json` file to your container. From the command prompt of your operating system (for this command, *not* the prompt in your docker container) in the folder where you saved the `tweets-utf-8.json` file, run the following command.

```
docker cp tweets-utf-8.json my-mongo:/
```

Importing the Tweet Data

From this point onward, we assume you are running a shell, either in the docker container as described above.

You can now import our example twitter data, contained in the `tweets.json` file, by typing

```
mongoimport --db tweetdb --collection tweets --file tweets-utf-8.json
```

MongoDB organizes data into *databases* and *collections*. In this case, you will use the *tweetdb* database and the *tweets* collection of documents.

MongoDB uses its own shell that allows users to run queries. To start the shell, run the following command:

```
mongosh
```

Once you have started the shell, to access the *tweetdb* database, type

```
use tweetdb
```

Having done so, you can now access the *tweets* collection. For example, to print an example tweet, type

```
db.tweets.findOne()
```

Have a look at the resulting JSON object.

MongoDB *queries* are written in JavaScript. They are in some ways analogous to SQL. The following tutorial has many examples:

https://docs.mongodb.com/v6.0/tutorial/query-documents/

The following reference page gives the different operators that can be used in queries:

https://docs.mongodb.com/v6.0/reference/operator/query/

For this assignment, we recommend you compose your queries in a local file on your computer, and copy-and-paste it into the mongo shell to try them out.

Submit a file called `queries.js` that contains a query for each of the following questions.

1) [5pts] Retrieve all tweets that are replying to the user with screen name “globeandmail”

2) [5pts] Retrieve all tweets made by the user with screen name “MLHealthUnit”

Submit a file called `aggregations.js` that contains a query for each of the following questions, using the MongoDB aggregation framework.

Aggregations in MongoDB are summaries of a collection. They are similar in concept to the operations performed in a MapReduce. MongoDB aggregations are more restrictive than MapReduce, but their implementation is very efficient. See the following documents for details.

https://docs.mongodb.com/v6.0/core/aggregation-introduction/

https://docs.mongodb.com/manual/reference/operator/aggregation/

3) [10pts] Produce a list of users, together with the total number of times they tweeted, sorted in decreasing order.

4) [10pts] Produce a list of place names, together with the total number of tweets from that place name, sorted in decreasing order.

5) [15pts] Produce a list of users, together with the total number of replies to that user, sorted in decreasing order.

6) [15pts] Produce a list of users, together with the total number of hashtags used by that user, sorted in decreasing order.

Submit a file called `mapreduce.js` that provides a mapper, reducer, and mongodb query to answer the question below.

MongoDB also provides a mechanism for MapReduce computations, though it is deprecated and *you will get a warning when you try to use it, but that is OK for this assignment.* Here is an example of mapreduce in mongodb:

```
    function myMapper() {
        //The mapper function is called with each document, which has the special
name 'this'
        //Emit a key-value pair:
        //(the mapper can emit many key-value pairs if needed)
        emit(this.user.screen_name, 1);
    }

    function myReducer(key, values) {
        //The reducer is called once for each key, and is passed an array
        //containing all values corresponding to that key.
        //Produce the desired result
        return Array.sum( values );
    }

    db.tweets.mapReduce(myMapper, myReducer, { query: {}, out: "mroutput" })
    db.mroutput.aggregate({$sort: {value: -1}})
```

Note that the output of the MapReduce has been placed in a new collection called `mroutput`, which is then queried to get the top answers. (This collection can be given any name.)

Again, you can paste this whole code block into mongosh to try it.

7) [40pts] Produce a new collection that contains each hashtag used in the collection of tweets, along with the number of times that hashtag was used.

Hint:

To do something with each object in an array in javascript, we can write:

```
for(obj of arr) {
    //Do something with obj, or obj.field, or whatever...
```

```
        }
```