ECE472 — Methods and tools for big data

Homework 3

Manuel — JI (Summer 2024)

Reminders

- Push to branch h3 and release with tag hw3
- Answer non-coding questions in a README.md
- Write in a complete style (subject, verb, and object)
- Explain your reasoning and be critical on your results

Ex. 1 — *MapReduce*

In this exercise we write a MapReduce program to solve the second exercise from lab 2.

1. Write a `Map` class which extends the MapReduce `Mapper` class, extracts, and outputs pairs composed of a student ID and a grade.
   Hint: read the file by line and tokenize each of them using `StringUtils`.
2. Write a `Reduce` class which extends the MapReduce `Reducer` class, outputs pairs composed of a student ID and its highest grade.
   Hint: use `Iterable<Text>` to iterate over all the values of a given key.
3. Write a driver function write set all the necessary properties to configure the MapReduce job.
   Hint: specify what classes are to be used by the Mapper and Reducer, as well as where the input and output files are located.
4. Run the MapReduce program and compare the running time to the streaming approach used in the lab. Draw a table showing the comparison for various file sizes.

Ex. 2 — *Avro*

1. Explain the three ways or API styles into which Avro can be used in MapReduce, and when to apply each of them.
   Hint: the three approaches are (i) specific, (ii) generic, and (iii) reflect.
2. Use your MapReduce program from the previous exercise to process the Avro file produced in Homework 2 exercise 4.

Ex. 3 — *Bloom filters*

In general data should be filtered before running actions on it. For instance in Lab 2 exercise 2, one might want to retrieve the maximum grade for students whose ID ends with a three. An efficient way to achieve this is to run a preprocessing job to create a Bloom filter and filter out records in the mapper.

1. Describe what a Bloom filter is and how it works.
2. Using the `BloomFilter` class write a mapper which creates a Bloom filter.
   Hint: check Hadoop documentation for more details on the `BloomFilter` class.
3. Using `Iterable<BloomFilter>` combine all the Bloom filters together in the reducer and output the result into a serialized Avro file.

Note: the example used in this exercise is very basic and does not reflect a real life setup. However our main goal is simply to understand how to work with Bloom Filters, It is therefore easier to apply them on

a dataset we are familair with and refer to code already written. For more information on Bloom Filters and why so they are very common in big data, refer to Box 1

Box 1: An brief introduction to Bloom Filters

A bloom filter is a probabilistic data structure: when "no" is returned it is 100% sure the element is not in the set, however when "yes" is returned then maybe the element is in the set, but it might also be a false positive.

In Exercise 3, we could "directly check" `x%10==3` in a deterministic was, however, this would require much more memory and time on a large dataset. From a general perspective, in big data statistical or probabilistic results are fully satisfying, since deterministic ones would take far too long and be very memory consuming.

From a computation perspective, bloom filters can be used as a "preprocessing" jobs, loaded by the mappers. Then when processing the map records the mappers decide, with respect to the bloom filters, whether or not to send each record to a reducer.

A bloom filter is "more complex" than a simple modular test. However the gain is massive in term of size and time as a much smaller set needs to be scanned. For instance, for 1% error on a set of `int` the size decreases from 64 bits down to less than 10 bits! For .1% error no more than 15bits are necessary. For a massive dataset the amount of memory used is much lower. As result, for MapReduce it will translate into much "faster processing".

As a general rule, when dealing with big data it is very seldom possible to be 100% accurate, Therefore precision and speed need to be balanced. In most applications, e.g. friend suggestion on social network, page ranking for web crawling, a small error is acceptable, e.g. if a website is ranked third instead of second or fifth, user is very unlikely to even notice it; similarly a "wrong" friend suggestion once every hundred suggestions is acceptable..