User and comment join[1]

Reduce side join pattern (Challenge 6_a)

For this exercise you have to prepare a report including:

- Your data collections comment describing what they represent.
- Your source codes: a file precising in a comment the compiling and execution commands.
- Propose a test battery and report execution measures.
- Compress everything and sent a .zip or a .tar to genoveva.vargas@gmail.com

1.1 Problem statement without filter

Problem: Given a set of user information and a list of user's comments, enrich each comment with the information about the user who created the comment.

In this challenge, we ask to perform an inner, outer, and antijoin.

1.2 Implementation

Look at page 112 of the book "Map Reduce design patterns" and see the proposed `Driver, Map` and `Reduce` codes. Prepare a data collection of your choice and implement the solution. As in previous challenges, you can use StackOverflow or any other source for generating your collection. Program the first part of the example.

In the example proposed by the book, two mapper classes are created: one for the user data and one for the comments. Each mapper class outputs the user ID as the foreign key, and the entire record as the value along with a single character to flag which record came from what set. The reducer then copies all values for each group in memory, keeping track of which record came from what data set. The records are then joined together and output.

- Describe the general principle of the solution proposed by the book.
- Explain the purpose and necessity of the `Driver` code.
- Discuss the statement of the book, use examples to support your arguments: "*When you output the value from the map side, the entire record doesnt have to be sent. This is an opportunity to optimize the join by keeping only the fields of data you want to join together. It requires more processing on the map side, but is worth it in the long run. Also, since the foreign key is in the map output key, you don't need to keep that in the value, either.*"
- Implement and compare intuitively and with tests the different implementations of the joins.
- Explain the raisons that make a combiner optimization not of much interest.
    - Prepare collections of different sizes to run your tests trying to get to the limits of your solution.
    - Make comparisons. Do not hesitate to prepare graphics.
- Explain the statement: "*Be considerate of follow on data parsing to ensure proper field delimiters. Outputting an empty text object is actually unwise. A record that contains*

[1] This challenge is an example proposed in the book MapReduce design patterns, pp. 111.

the proper structure but with null fields should be generated instead of outputting an empty object. This will ensure proper parsing for follow-on analytics."

1.3 Reduce side join with bloom filter

A possible optimization of the previous challenge is to add a bloom filter to filter out some of mapper output. This will help reduce the amount of data being sent to the reducers and in effect reduce the runtime of our analytic. Look at page 117 and implement the proposed solution.

- Explain and illustrate how the bloom filter optimizes a standard reduce side join.
- In which way are the mappers and drivers different or similar between both solutions? How does the `CommentJoin Mapper` work?
- Explain and discuss using examples the following book statement:

  "In this algorithm, we don't need to verify the user's reputation in the reducer prior to writing to the file system. While false positive records were output from the `CommentJoinMapperWithBloom`, they won't be joined up with users on the reduce side since there will be nothing to join them with. The 100% check was done by only outputting user IDs with a reputation greater than 1,500. The main gain we received out of this Bloom filter was vastly reducing the number of comments output to the mapper phase. Be conscious of Bloom filter false positives and how they will affect your reduce side join operation."