Exercise 2 (Summer 2020)

6.0 VU Advanced Database Systems

Information

General

Work through the exercises below and write a report on your solutions. Submit the report and your source files as a single .zip file (containing one PDF (.pdf) file, the Python source files (.py) for your MapReduce programs, a script file (.sh) to your MapReduce programs, an .sql file for your Hive commands and a JupyterLab notebook (.ipynb)) (max. 20MB) in TUWEL and register for an exercise interview. **You can only receive points for this exercise if you attended the interview.** We expect that your solutions actually run on the cluster. Furthermore, you are expected to discuss your solutions and answer questions regarding it.
This exercise involves working with the Hadoop cluster and accessing the JupyterLab web interface. Instructions on how to access the web interfaces and to connect to the cluster are provided as part of the lecture. In addition to this, we also provide a recorded presentation that will walk you through this. You will receive your account information via email.

Deadlines

at the latest April 24th	**23:59**		Upload your submission on TUWEL
Don't forget!		⇒	Register for an exercise interview slot in TUWEL

Exercise Interviews

During Exercise Interviews, the correctness of your solution as well as your understanding of the underlying concepts will be assessed.
The scoring of your submission is primarily based on your performance at the interview. Therefore it is possible (in extreme cases) to get 0 points even though the submitted solution was technically correct.
Please be punctual for your interview. Otherwise we cannot guarantee that your full solution can be graded in your assigned time slot. **Remember to show your student id on the video.** It is not possible to receive a score for your solution without showing your id, as we cannot confirm your identify.

Question Sessions

We offer question sessions via the TUChat to help you with any problems you have with the exercises. The goal of these sessions is to help you understand the material, not to check your solutions or solve the exercises for you. In this spirit we also ask that you engage with the exercise sheet before coming to the session. **Participation is completely optional.**
Exact times and URLs are provided on TUWEL.

TUWEL Forum

You can use the course forum on TUWEL to clarify questions regarding the exercise sheets. *Please do not post your (even partial) solutions on the forum.*

Theory of MapReduce

Exercise 1 (Costs of MapReduce[1]) *[1 points]* Suppose we execute the WordCount MapReduce program described in the lecture on a large repository, such as a copy of the Web. We shall use 100 Map tasks and some number of Reduce tasks.

a) Suppose we do not use a combiner at the Map tasks. Do you expect there to be *skew* in the times taken by the various reducers to process their value lists? Why or why not?

b) If we combine the reducer functions into a small number of Reduce tasks, say 10 task, at random, do you expect the skew to be significant? What if we instead combined the reducers into 10.000 Reduce tasks ?!

c) Suppose we do use a combiner at the 100 Map tasks. Do you expect the skew to be significant? Why or why not?

d) Suppose we increase the number of Reduce tasks significantly, say to 1.000.000. Do you expect the communication cost to increase? What about the replication rate or reducer size? Justify your answers.

Exercise 2 (Relational Operations[1]) *[2 points]*
In the form of relational algebra implemented in SQL, relations are not sets, but bags; that is, tuples are allowed to appear more than once. There are extended definitions of union, intersection, and difference for bags, which we shall define below. Sketch out formal MapReduce algorithms (consisting of map and reduce function) for computing the following operations on bags R and S:

a) **Bag Difference $\mathbf{R - S}$**, defined to be the bag of tuples in which the number of times a tuple t appears is equal to the number of times it appears in R minus the number of times it appears in S. A tuple that appears more times in S than in R does not appear in the difference, e.g. if tuple t occurs in R 4 times, and in S 3 times, then it should occur in the bag difference of R with S only once.

b) **Anti Join $\mathbf{R \Delta S}$**: For this assume the following schema: $R(A, B, C)$ and $S(B, C, D)$, where A, B, C are some distinct subset of any domain D. In other words, we assume for each tuple $t \in R$ to have the form (a, b, c) where $a \in A$, $b \in B$ and $c \in C$ and analogously for S. The *anti join* of R with S on (B, C) shall be defined as the bag of tuples $(a, b_r, c_r) \in R$ s.t. there are no tuples $(b_s, c_s, d) \in S$ where $b_r = b_s$ and $c_r = c_s$.

In addition to this, also identify the communication cost of each of your algorithms, as a function of the input size.

Your solution for Exercises 1 and 2 will consist of:

- A short written report, detailing your answers.

[1] These are from or inspired by "Mining of Massive Datasets" from Leskovec et al. http://www.mmds.org/

MapReduce in Practice

Exercise 3 (MapReduce) *[4 points]* For this exercise, you are tasked with writing your own Hadoop MapReduce program in Python and to run it on the cluster on two provided datasets. The dataset for this exercise consists of data from a fictional eCommerce company. One is data from the month of October 2019, the other from November 2019[2]. We provide a JupyterLab Notebook to write and execute MapReduce jobs directly on JupyterLab.
You can find the datasets on the cluster under the following locations:

- **local file system & Hadoop file system (HDFS)**
  /home/adbs20/shared/ecommerce/2019-Nov.csv
  /home/adbs20/shared/ecommerce/2019-Oct.csv

Note: When running your job on the command-line with the intention of using the HDFS, you need to prepend "hdfs://" before the address string.

For use in local testing, we also provide smaller versions of both datasets on the local file-system, ending with "-short.csv". Make sure that your MapReduce job also works correctly on the complete dataset on the cluster.

Tasks:

a) **Write a MapReduce job with "2019-Oct.csv" as input and following output:** For each customer, show the brand of which they purchased the most products; make sure to have the following format in your final output:

user_id, brand

If a user never bought a product, they should not appear in your final output at all. Make sure that your program correctly deals with the header, and possible sparse values.

b) **Write a MapReduce job with both datasets as input and following output:** For each brand, show the number of customers, who purchased something in **both** November *and* October of 2019 and who purchased more products of this brand than any other brand; make sure to have the following format in your final output:

brand, user_count

Note: You are strongly advised to build on your work from a).

c) Once your jobs have run successfully on the cluster, use the Hadoop Web Interface[3] to find your MapReduce job(s) and find out for each job what the replication rate was, as well as the input and output size.

Your solution for Exercise 3 will consist of:

- A short written report
- The Python source files needed to run your Hadoop MapReduce jobs on the cluster. *Submissions which do not run on the cluster will not be counted as a valid submission.*
- A script file (.sh) which runs your Python jobs with HDFS input on the cluster.

[2] https://www.kaggle.com/mkechinov/ecommerce-behavior-data-from-multi-category-store
[3] https://c100.local:8088, only accessible from internal TU network **and** with port forwarding from lbd.zserv.tuwien.ac.at, see the guide for help in setting this up.

Hive

Exercise 4 (Hive Exercise) *[4 points]* For this exercise you will be working with Hive. The most important commands you will need are listed and explained on the slide deck Slides II.2 Hadoop. More information on Hive QL can be found in its language manual[4].
To access Hive on the cluster, first set up a VPN connection to the TU Wien, then open the URL `https://lbd.zserv.tuwien.ac.at:8889/hue/editor/?type=hive`.

You can find the datasets for this exercise[5] on the cluster under the following locations:

- local file system:
  /home/adbs20/shared/stackexchange/badges.csv
  /home/adbs20/shared/stackexchange/comments.csv
  /home/adbs20/shared/stackexchange/posts.csv
  /home/adbs20/shared/stackexchange/postlinks.csv
  /home/adbs20/shared/stackexchange/users.csv
  /home/adbs20/shared/stackexchange/votes.csv

Tasks:

a) **Create a new database and create tables for all datasets listed above.**
Import the data from the CSV files into your Hive tables.

b) **Explore how Partitions and Buckets affect joins**
For this part you will need to create new versions of your tables with various partitions and buckets (since Hive does not allow these to be changed after creating a table). However, it is simple to insert the data of an existing table to another with buckets and partitions.

Make sure to set these options to allow "dynamic" inserts into buckets and partitions:

```
set hive.enforce.bucketing = true;
set hive.exec.dynamic.partition = true;
set hive.exec.dynamic.partition.mode = nonstrict;
set hive.exec.max.dynamic.partitions.pernode=1000;
set hive.exec.max.dynamic.partitions = 1000;
```

Use **SELECT COUNT** (DISCTINT column_name) to determine how many partitions a given column would need. For this exercise you are given the hard limit of 1000 partitions, so your chosen partition column must not have more values. Hive will simply report a nondistinct error if you attempt to create tables with more than 1000 partitions.

Consider the following SQL query:

```
SELECT p.posttypeid, COUNT(p.id), COUNT(c.id) FROM posts p, comments c, users u
WHERE c.postid=p.id AND u.id=p.owneruserid AND u.reputation > 100
AND NOT EXISTS (SELECT * FROM postlinks l WHERE l.relatedpostid = p.id)
GROUP BY p.posttypeid
```

[4] `https://cwiki.apache.org/confluence/display/Hive/LanguageManual`
[5] Extracted from StackExchange https://archive.org/download/stackexchange

1) **Explore how Partitions or Buckets on the join columns affect the query** Use EXPLAIN (EXTENDED) to analyse the query plans for each case. Be careful what you choose as partition or buckets, as you want each to evenly subdivide your dataset (ideally). For example, setting a primary key as a partition will create a large number of directories each with a single tuple. This will take a very long time to create and it will obviously not help to optimise any query. Avoid this by *thinking ahead.*

Report on your findings, and what did or did not surprise you.

***Note**: It is not required to test all possible join attributes and combinations of buckets and partitions. The focus is on trying to develop an understanding of how Hive uses (or does not use) the physical storage of the table to optimise queries.*

c) **Correlated Subqueries**
Hive does not support the entire SQL standard. Specifically, the use of subqueries is limited, as was mentioned in the lecture. For a more detailed technical description of what Hive supports, you can check its language manual[6].

Consider the following SQL query:

```
SELECT p.id FROM posts p, comments c, users u, badges b
WHERE c.postid=p.id  AND u.id=p.owneruserid   AND u.id = b.userid
AND  u.upvotes IN  (SELECT MAX(upvotes)
        FROM users u WHERE u.creationdate > p.creationdate)
AND EXISTS (SELECT 1 FROM postlinks l WHERE l.relatedpostid > p.id)
AND (b.name LIKE 'Research Assistant');
```

This query cannot be run on Hive. Try to find a semantically equivalent query (i.e. a query which produces the same, desired results on any given input database instance) which Hive *does* accept.

Your solution for Exercise 4 will consist of:

- A short written report, detailing your findings.
- An .sql file with all the commands needed to generate your tables and run the queries.

***Note:** Be sure to give your database a unique name (ideally with your student id). The cluster uses an embedded Metastore, so be careful not to delete or manipulate the database of any other person. We advise that you save all your Hive commands externally while working on this exercise, so you can quickly restore everything in case something does get accidentally deleted.*

[6] `https://cwiki.apache.org/confluence/display/Hive/LanguageManual+SubQueries`

Spark

Exercise 5 (Spark in Scala) *[4 points]*
For this exercise, you will work on the JupyterLab[7] notebook which is provided on TUWEL, and solve the tasks listed therein. These tasks, in addition to writing Spark code, require you to analyse (among other things) various query plans and to reason about them.

To familiarise yourself with Spark and the Scala language, we also provide you with two JupyterLab notebooks, namely Notebook 1 and Notebook 2, which you can upload on JupyterLab and run yourself. To get a more fine-grained understanding, and look up the types and definitions of various functions, we recommend that you visit the Spark[8] and Spark SQL[9] documentation.

Tasks:

a) **Elementary RDD functions**
You are given an RDD object, and need to produce a set of specified RDDs from it. Use the RDD API for this, specifically functions such as map, flatmap, reduce, etc.

b) **SQL to Dataframe (and back again)**
You are given a series of Spark SQL (resp. Dataframe) queries, and are asked to provide an equivalent Dataframe (resp. Spark SQL) version of it.

Furthermore, look at the query plans and use the Spark Internal Web UI[10] to determine if they have the same performance (or which version is better).

c) **Wide and Narrow Dependencies**
Look at the Dataframe queries given to you or for which you wrote the Dataframe version as part of b).
Use the Spark Internal Web UI to analyse the dependencies and stages of the queries, and determine which commands on which Dataframes are executed as wide dependencies and which as narrow dependencies.

Your solution for Exercise 5 will consist of:

- (as per usual) a short written report
- A JupyterLab notebook file (.ipynb) containing your solutions.

[7] `https://lbd.zserv.tuwien.ac.at:8000/hub/login` – **requires VPN connection to TU Wien**
[8] `https://spark.apache.org/docs/latest/rdd-programming-guide.html`
[9] `https://spark.apache.org/docs/latest/sql-programming-guide.html`
[10] Step-by-step Video Instruction: `https://tuwel.tuwien.ac.at/mod/resource/view.php?id=728017`