Exercise 2 (Summer 2019)

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 Java source files (.java) for 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 compile and 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. Tutorials on how set up a work environment and to connect to the cluster are provided on TUWEL. In addition to this, as part of the lecture, we provide a recorded presentation that will walk you through this step by step. You will receive your account information via email.

Deadlines

at latest 14th of May	**12.00 pm**	Upload your submission on TUWEL
at latest 16th of May	**11.55 pm**	Register for an exercise interview in TUWEL

Exercise Interviews

During Exercise Interviews, the correctness of your solution as well as your understanding of the underlying concepts will be assessed. **Every group member has to be able to explain all parts of your submission.**
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 bring your student id.** It is not possible to receive a score for your solution without an id.

Question Sessions

About a week before the submission deadline, we offer question sessions 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 locations will be announced 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.*

MapReduce Practice

Exercise 1 (MapReduce) *[4 points]*
For this exercise, you are tasked with writing your own Hadoop MapReduce program in Java and to run it on our cluster on two provided datasets.
The dataset for this exercise consists of data from the Seattle Public Library. The first is the inventory of the entire library collection[1] of over 700000 unique titles. The second are the checkout records from 2005 onwards[2], detailing when each item was borrowed.
You can find the datasets on the cluster under the following locations:

- local file system:
  /home/adbs/2019S/shared/seattle-checkouts-by-title/checkouts-by-title.csv
  /home/adbs/2019S/shared/seattle-library-collection-inventory/library-collection-inventory.csv
- HDFS:
  /user/adbs/2019S/shared/seattle-checkouts-by-title/checkouts-by-title.csv
  /user/adbs/2019S/shared/seattle-library-collection-inventory/library-collection-inventory.csv

Note: Any MapReduce task (on the cluster) can only read from and write to the HDFS.

For use in local testing, we also provide smaller versions of both datasets named "library-collection-inventory-abbr.csv" and "checkouts-by-title.csv-abbr.csv" on the local file-system. Make sure that your MapReduce job also works correctly on the actual dataset on the cluster.

Tasks:

a)
- **Write a MapReduce job which takes as input the checkout records and computes the following:**
  For each author, list the title that was checked out (borrowed) the most, and make sure to have the following format in your final output:

  Author, Title

  If a title was never checked out (*the 'checkouts' attribute can be 0 !*), it should not appear in your final output at all.
  Make sure that your program correctly deals with the header, and possible sparse values.

  Hint: You can implement this as a one-round job, with a more complex reducer, or a multiple round job, with simpler mapper and reducer functions.
- Once your job has 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, in addition to the input and output size.

We provide you with a custom class TextPair.java that might be used for this exercise (or simply serve as a template for your own custom types). Since it inherits a Hadoop internal class, it can be used to define the output or input types of mapper or reducer functions. Furthermore, you might find the class CSVSplitter.java useful to parse the csv files.

[1] https://www.kaggle.com/city-of-seattle/seattle-library-collection-inventory
[2] https://www.kaggle.com/city-of-seattle/seattle-checkouts-by-title
[3] https://c100.local:8088, only accessible from internal TU network **and** with port forwarding from lbd.zserv.tuwien.ac.at at port 5150, see the tutorial for help in setting this up

We recommend that you work locally under the CloudEra Quickstart VM, as it comes pre-installed with all the libraries and tools you need to run local Hadoop jobs. We also provide a Maven *pom.xml* file that provides additional libraries that you might want to use and that allow you to run our MyWordCount.java example as a baseline for your solution.

b)
- **Write a MapReduce job which takes as input both the library inventory *and* the checkout records and computes the following:**
  For each author, list the title that was checked out (borrowed) the most, and make sure to have the following format in your final output:

  Author, Title, PublicationYear, Subjects, ItemLocation

  Make sure to only list tuples where the same author and title pair appears in both the library inventory and the checkout records.

  Note: You are strongly advised to build on your results from a). *Also, assume for this exercise that both datasets are always consistent on author and title names, i.e. that they use the same string for the same author or title.*
- Once your job has run successfully on the cluster, use the Hadoop Web Interface to find your MapReduce job(s) and find out for each job what the replication rate was, in addition to the input and output size.

For b) you will have to use the MultipleInputs[4] class. It allows you to create MapReduce jobs with multiple mapper functions, each for a different input file. The results of the mappers is then sent to the same reducer function.

Your solution will consist of:

- A short written report
- The Java source files needed to compile and run a Hadoop MapReduce job on our cluster. *Submissions that don't correctly compile on the cluster will not be counted as a valid submission. Furthermore, we do not want you submitting .jar files, nor will we count them toward a valid submission for this exercise.*

[4] https://hadoop.apache.org/docs/r2.6.0/api/org/apache/hadoop/mapreduce/lib/input/MultipleInputs.html

Theory of MapReduce

Exercise 2 (Costs of MapReduce[4]) *[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 3 (Relational Operations[4]) *[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 for computing the following operations on bags R and S:

a) **Bag Union**, defined to be the bag of tuples in which tuple t appears the sum of the numbers of times it appears in R and S, e.g. if tuple t occurs in R 3 times, and in S 4 times, then it should occur in the bag union R and S 7 times.

b) **Bag Difference**, 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.

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 3 and 4 will consist of:

- A short written report, detailing your answers.

[4]These are taken from "Mining of Massive Datasets" from Leskovec et al. http://www.mmds.org/

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[5].
To access Hive on our cluster, simply connect via SSH to lbd.zserv.tuwien.ac.at and type "hive". The dataset for this exercise consists of data extracted from StackExchange[6].

You can find the datasets on the cluster under the following locations:

- local file system:
  /home/adbs/2019S/shared/hive/badges.csv
  /home/adbs/2019S/shared/hive/comments.csv
  /home/adbs/2019S/shared/hive/posts.csv
  /home/adbs/2019S/shared/hive/postlinks.csv
  /home/adbs/2019S/shared/hive/users.csv
  /home/adbs/2019S/shared/hive/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= X;
SET hive.**exec**.max.dynamic.partitions.pernode = Y
where X and Y are numbers that you should set to fit your table.

Use **SELECT COUNT** (DISCTINT column_name) to determine how many partitions a given column would need. Then you can simply use **INSERT TABLE** to populate the newly created tables.

Consider the following SQL query:

```
SELECT p.id FROM posts p, comments c, users u, votes v
WHERE c.postid=p.id AND c.userid=p.owneruserid AND u.id=p.owneruserid
AND u.reputation > 100 AND v.postid = p.id AND v.userid = p.owneruserid
AND NOT EXISTS (SELECT 1 FROM postlinks l WHERE l.relatedpostid = p.id)
```

[5] https://cwiki.apache.org/confluence/display/Hive/LanguageManual
[6] 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[7].

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.upvotes+3 >= (SELECT COUNT(upvotes)
                    FROM users
                    WHERE u.creationdate = c.creationdate)
AND EXISTS (SELECT 1 FROM postlinks l WHERE l.relatedpostid > p.id)
AND u.id = b.userid
AND (b.name LIKE 'Autobiographer');
```

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 will consist of:

- A short written report, detailing your answers.
- 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 group name or the student id of one you). Our cluster uses an embedded Metastore, so be careful not to delete or manipulate the database of any other group. 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.

[7] 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[8] 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 try to reason about them.

To familiarize 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[9] and Spark SQL[10] 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[11] 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 will consist of:

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

[8] https://lbd.zserv.tuwien.ac.at:8000/hub/login
[9] https://spark.apache.org/docs/latest/rdd-programming-guide.html
[10] https://spark.apache.org/docs/latest/sql-programming-guide.html
[11] The link to this is generated when you first initialize your Spark interpreter. If you don't clear any output cells, then this message should always be on your Scala notebook. Accessing the Spark Web UI, as with the Hadoop Internal Web UI, requires port forwarding.