Univ.-Prof. Dr.-Ing. Matthias Boehm
Graz University of Technology
Computer Science and Biomedical Engineering
Institute of Interactive Systems and Data Science
BMK endowed chair for Data Management

4 Data Management WS20/21: Ex 04 – Large-Scale Data Analysis

Published: January 05, 2021
Deadline: January 26, 2021, 11.59pm

This exercise on large-scale data analysis aims to provide practical experience with distributed data management and large-scale data analysis on top of Apache Spark. The expected result is a zip archive named `DBExercise04_<student_ID>.zip`, submitted in TeachCenter. The entire exercise is *extra credit* for the course data management.

4.1 Apache Spark Setup (3/25 points)

As a preparation step, setup Apache Spark and necessary Hadoop client APIs inside an IDE (integrated development environment) of your language choice. This exercise can be done with the Spark language bindings Java, Scala, or Python. For example in Java, you include the maven dependencies `spark-core` and `spark-sql`. On Windows, please download `winutils.exe` from `https://github.com/steveloughran/winutils/tree/master/hadoop-2.7.1/bin`, put it into a directory `<some-path>/hadoop/bin`, and create an environment variable `HADOOP_HOME=<some-path>/hadoop`. The input data for this exercise is available at `https://mboehm7.github.io/teaching/ws2021_dbs/input_data.zip` (from Ex 3, based on the schema from Ex 2).

Partial Results: N/A (every submission receives these points).

4.2 Query Processing via Spark RDDs (10/25 points)

Spark's basic abstraction for distributed collections are so-called Resilient Distributed Datasets (RDDs). In this task, you should implement the queries **Q02** and **Q06** from Task 2.3 via RDD operations, collect the results in the driver and print the result list to stdout. Please implement these queries as two self-contained functions/methods `executeQ02RDD()` and `executeQ06RDD()` that internally create a `SparkContext` sc, read the files via `sc.textFile()`, and use only RDD[1] operations to compute the query results.

Partial Results: Source file `QueriesRDD.*`.

[1] `https://spark.apache.org/docs/latest/rdd-programming-guide.html`

4.3 Query Processing via Spark SQL (6/25 points)

Spark also provides the high-level APIs `Dataframe` and `Dataset` for SQL processing. In this task, you should implement queries **Q02** and **Q06** from Task 2.3 via Dataset operations, and write the outputs to JSON files `out02.json` and `out06.json`. Please implement these queries as two self-contained functions/methods `executeQ02Dataset()` and `executeQ06Dataset()` that internally create a `SparkSession` sc, read the inputs files via `sc.read().format("csv")`, and use only SQL or Dataset operations to compute and write the query results. You might either (1) register the individual input Datasets as temporary views and compute the results directly via SQL, or (2) alternatively use the functional API of Datasets. Both specifications share a common query optimization and processing pipeline.

Partial Results: Source file `QueriesDataset.*`.

4.4 Movie Recommendations (6 points)

Given the ratings table, first extract the user key `UKey`, movie key `MKey`, and rating, and then train a movie recommendation model with matrix-factorization-based collaborative filtering (e.g., alternating least squares) on top of Apache Spark. The basic idea is to train low-rank factors $\mathbf{U}$ (number of users times rank) and $\mathbf{V}$ (rank times number of movies), that matrix multiplied together approximate the ratings matrix $\mathbf{U}\,\mathbf{V} \approx \mathbf{X}$ (for existing non-zero entries). These factors then allow predicting the unknown movie ratings by a given user (or movie), and pick the top-k movies for recommendation. You might implement the training procedure from scratch or use existing Spark ML library algorithms. Finally, store the computed model.

Partial Results: Source file `Recommendation.*`.