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 WS21/22: Ex 04 – Large-Scale Data Analysis

Published: Dec 11, 2021 (updates: Jan 17 update data T3)
Deadline: Jan 18, 2022, 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`[1], 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/ws2122_dbs/T3_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 (11/25 points)

Apache Spark's basic abstraction for distributed collections are so-called Resilient Distributed Datasets (RDDs). In this task, you should implement the query **Q09** from Task 3.1 (whose reference implementation we will share by Dec 28) via RDD operations, collect the results in the driver and print the result list to stdout. Please implement this query as a self-contained function/method `executeQ09RDD()` that internally creates a `SparkContext` sc, reads the files via `sc.textFile()`, and uses only RDD[2] operations to compute the query results.

Partial Results: Source file `QueryRDD.*`.

[1] The latest versions of precompiled `winutils.exe` can be found at `https://github.com/cdarlint/winutils`.
[2] `https://spark.apache.org/docs/latest/rdd-programming-guide.html`

4.3 Query Processing via Spark SQL (5/25 points)

Spark also provides the high-level APIs `Dataframe` and `Dataset` for SQL processing. In this task, you should implement query **Q09** from Task 3.1 via Dataset operations, and write the outputs to JSON files `out09.json`. Please implement this query as a self-contained function/method `executeQ09Dataset()` that internally creates a `SparkSession` sc, reads the inputs files via `sc.read().format("csv")`, and uses 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 `QueryDataset.*`.

4.4 Graph Processing (6 points)

Given a co-author graph (with vertexes being authors, and edges being co-author relationships) in form of two alternative graph representations (provided as `AuthPapersCOO.csv`, and `AuthPapersCSR.csv` in `https://mboehm7.github.io/teaching/ws2122_dbs/graph_data.zip`), write a program to compute the *connected components* of the co-author graph. Your program should leverage Spark, but the API selection is up to you (e.g., RDD operations, SQL, or higher-level libraries like Spark GraphX). The expected output is a text file mapping vertexes (author IDs) to components, as well as a summary of the number of components printed to stdout.

Partial Results: Source file `Components.*`.