Univ.-Prof. Dr.-Ing. Matthias Boehm
Technische Universität Berlin
Faculty IV - Electrical Engineering and Computer Science
Berlin Institute for the Foundations of Learning and Data (BIFOLD)
Big Data Engineering (DAMS Lab) Group

1 DIA WiSe2024: Exercise – Streaming Full Text Search

Published: Oct 16, 2024 (last update: Dec 04)
Deadline: Jan 30, 2025, 11.59pm

This exercise is an alternative to the DIA programming projects, and aims to provide practical experience in the development of data engineering and ML pipelines. The task of this semester is to filter a stream of documents using a dynamic set of exact and approximate continuous keyword match queries. This task resembles the SIGMOD Programming Contest 2013. You may use any programming language(s) of your choosing, and utilize existing open-source ML frameworks and libraries. The expected result is a zip archive named `DIA_Exercise_<student_ID>.zip` (replace `<student_ID>` by your student ID) of max 5 MB, containing:

- The source code used to solve the individual sub-tasks
- A PDF report of up to 8 pages (10pt), including the names of all team members, a brief summary of how to run your code, and a description of the solutions to the individual sub-tasks.

Data and Reference Implementation: The task API header file, a reference implementation of the task interface, the test driver along with an example workload, and a Makefile are available at https://github.com/transactionalblog/sigmod-contest-2013. A detailed task description can be found here. Additionally, you can find both smaller and bigger datasets here.

Grading: This exercise can be pursued in teams of 1 to 3 persons (one submission, scale quality expectations). The overall grading is a *pass/fail* for the entire team. Exercises with $\geq 50/100$ points are a pass, and with $\geq 90/100$ points we receive 5 extra points in the exam.

1.1 Basic Implementation (30/100 points)

Implement the four core functions of the API: `StartQuery()`, `EndQuery()`, `MatchDocument()`, as well as `GetNextAvailRes()`. Detailed descriptions of the parameters and specifications for these functions can be found here. Write a test driver to validate these functions using both small and the big data files. The test driver should parse the files, invoke the API, and report results (errors, successes, and failures). For C/C++ implementations, you can reuse the provided test driver.

Expected Results: Code for basic implementation and the test driver, as well as an output file having the results of the tests. The report should discuss the implementation details.

1.2 Maximize Throughput (45/100 points)

Maximize the throughput of the basic implementation using optimization techniques such as multi-threading, caching, memory pre-allocation, and using specialized data structures like tries. The goal is to achieve a minimum speedup of 20x compared to either the reference or basic implementation, whichever is slower. Partial credit will be awarded for speedups below 20x.

Expected results: A separate source code file with the optimized implementation. The report must include the speedup as well as descriptions of the techniques and data structures used.

1.3 Data-Parallel Implementation (25/100 points)

Reimplement the functions on top of a distributed, data-parallel computation framework such as Apache Spark, Flink, or Dask. This data-parallel implementation should also pass the tests. Compare the execution time of the data-parallel implementation with the basic and optimized implementations.

Expected Results: Code for the data-parallel implementation, a description of the approach, as well as the execution time.