Assignment 2: Simple Search Engine using Hadoop MapReduce

Dmitrii Kuzmin

April 2025

Contents

1 Methodology

1.1 Data Collection and Preparation

A PySpark script (`prepare_data.py`) was used to extract 1000 documents from a Parquet dataset (`a.parquet`). Each document row (`id`, `title`, `text`) was sampled, sanitized, and written as UTF-8 text files under `/data` in HDFS. Subsequently, an RDD was created from these files and a unified tab-delimited file was stored in HDFS under `/index/data`, with each record: <doc_id> \t <doc_title> \t <doc_text>.

1.2 Indexer Design

Hadoop MapReduce pipelines were implemented in Python. The first pipeline consisted of `mapper1.py` and `reducer1.py`. In the mapper, text was tokenized and normalized (lowercase, punctuation was removed), and (term, doc_id) pairs were emitted. Term frequencies per document were aggregated by the reducer and document frequencies per term were computed.

The index was stored in Cassandra. The schema included three tables:

- `vocab(term TEXT, id TEXT, tf COUNTER, PRIMARY KEY(term, id))`
- `stats(category TEXT, key TEXT, value FLOAT, name TEXT, PRIMARY KEY(category, key))`
- `term_index(term TEXT PRIMARY KEY, idx INT)`

Term frequencies and document lengths were recorded for BM25 computation.

1.3 Ranker Design

The ranking component was implemented in `query.py` using the PySpark RDD API. The user query was read from stdin, tokenized and normalized, and Cassandra tables were loaded into broadcast RDDs. BM25 scores per document were computed:

$$\text{BM25}(q,d) = \sum_{t \in q} \log \frac{N}{\text{df}(t)} \cdot \frac{(k_1+1)\cdot \text{tf}(t,d)}{k_1 \left[(1-b) + b\frac{\text{dl}(d)}{\text{dl}_{avg}}\right] + \text{tf}(t,d)}$$

with hyperparameters $k_1 = 1$ and $b = 0.75$, total documents N, and average document length dl_{avg}. The scores were sorted to retrieve the top 10 documents (IDs and titles).

1.4 Scripts and Services

Services and pipelines were orchestrated via shell scripts:

- `start-services.sh`: used to start Hadoop and Cassandra clusters using Docker Compose.
- `index.sh`: used to run MapReduce pipelines on HDFS `/index/data` and update Cassandra tables.
- `search.sh`: used to submit the PySpark job on YARN to execute `query.py` with a user-provided query.
- `app.sh`: provided as a wrapper to start services, index sample documents, and perform sample searches.

2 Demonstration

2.1 Indexing

Indexing of 1000 documents was demonstrated. The index was summarized in the following table:

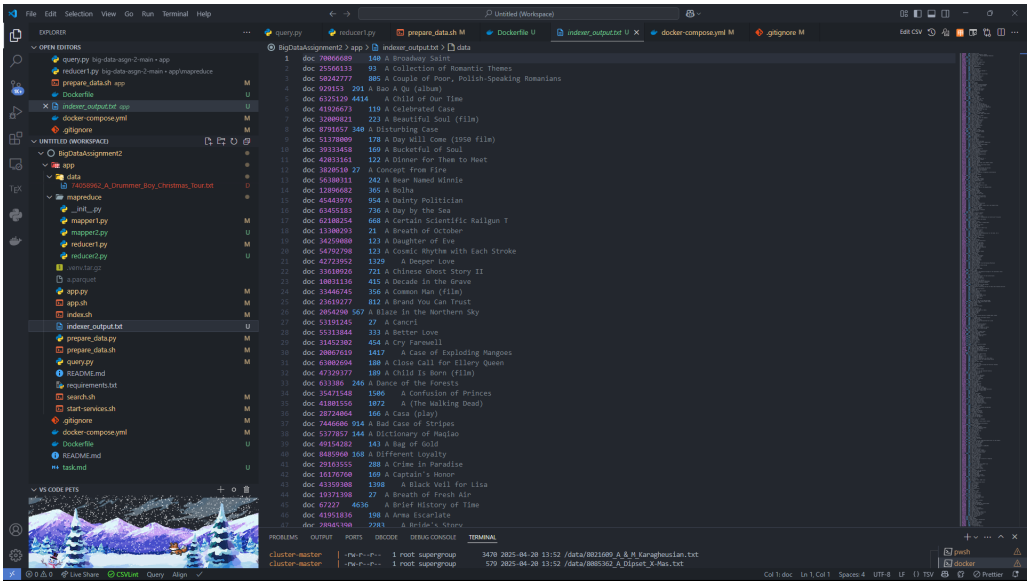

Figure 1: Index statistics and Cassandra table snapshots after indexing.

2.2 Search Results

Sample queries were run using `search.sh`. Results for the queries `"this is a query!"`, `"artificial intelligence"`, and `"big data"` were shown in Figure 2, listing the top 10 document IDs and titles.

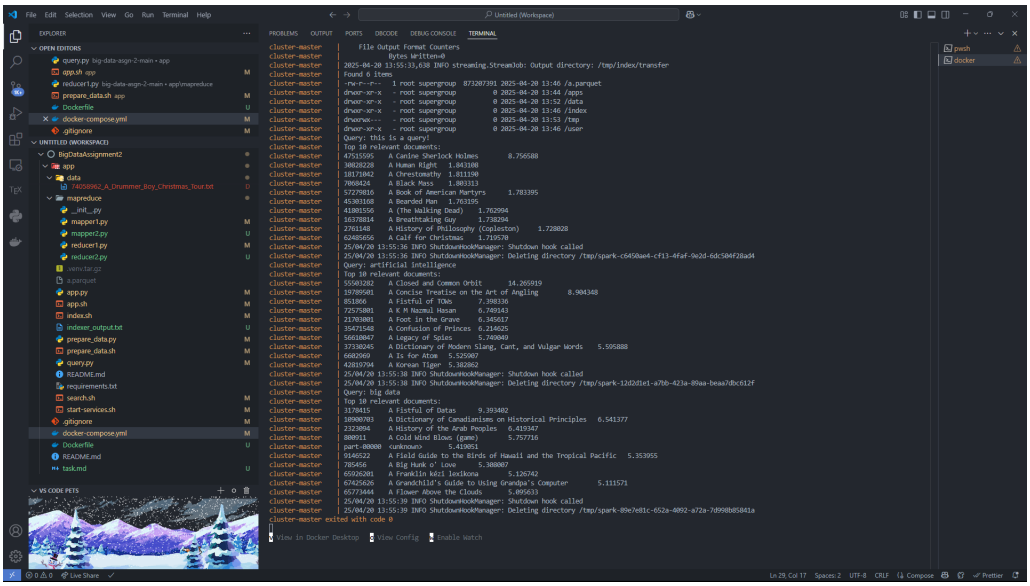

Figure 2: Sample search outputs for user queries.

Relevant documents were returned by the BM25-based ranking; for example, documents containing high term frequency and moderate length were ranked at the top.