Big Data Search Engine

using Hadoop MapReduce

Amine Trabelsi

Submitted for Big Data
Assignment 2

a.trabelsi@innopolis.university
group DS-01
Innopolis University
2025

Contents

Methodology

MapReduce Indexing Pipeline

The search engine relies on a Hadoop MapReduce pipeline to tokenize documents and build an inverted index stored in Cassandra.

Mapper (mapper1.py)

Each line of the input dataset (stored in HDFS) is expected to follow this tab-separated format:

`<doc_id>\t<title>\t<text>`

The mapper performs the following operations:

- Splits each line into `doc_id`, `title`, and `text`.
- Tokenizes the text into lowercase alphanumeric terms using a regular expression (`\w+`).
- Emits a key-value pair for each term occurrence in the format:

  `<term>\t<doc_id>\t1`

This effectively prepares the data for counting how many times each term appears in each document.

Reducer (reducer1.py)

The reducer receives all emitted term-document pairs sorted by key, and:

- Aggregates the total frequency (`tf`) of each term per document.
- Inserts the final result into a Cassandra table called `inverted_index`, with the schema:

```
(term TEXT, doc_id TEXT, tf INT,
 PRIMARY KEY (term, doc_id))
```

- All Cassandra table creation and insertion is handled directly within the reducer.
- The reducer also logs inserted terms to standard error for debugging purposes.

Query Processing with BM25 Ranking

After the inverted index is stored in Cassandra, a PySpark application (`query.py`) is used to process user queries and retrieve the top 10 most relevant documents using the BM25 scoring algorithm.

Workflow:

1. The user provides a query via command-line input.
2. The PySpark application connects to Cassandra and retrieves relevant entries from the `inverted_index` table using a `SELECT` query with all query terms.
3. Because only `(term, doc_id, tf)` data is available, the script:
   - Estimates document lengths by summing term frequencies (TFs) per document.
   - Assigns placeholder document titles (e.g., `"Document_<doc_id>"`).
   - Uses static placeholder values for inverse document frequency (IDF) and average document length.
4. Using the PySpark RDD API:
   - The BM25 score is computed for each document–term pair using the formula:

```
BM25 = idf * (tf * (k1 + 1)) / (tf + k1 * (1 - b + b * (dl / avgdl)))
```

- Scores are aggregated per document.
- The top 10 documents are selected based on the total BM25 score.

5. The final output displays the document IDs, their generated titles, and BM25 scores in descending order.

Execution: The PySpark application is launched in a distributed fashion on the Hadoop YARN cluster via the `search.sh` script:

```
bash search.sh "search engine indexing"
```

Demonstration

Running the container

To run the code it is sufficiant to run:

```
docker-compose up --build
```

this will build three images: cluster-master, cluster-slave, and cassandra-server

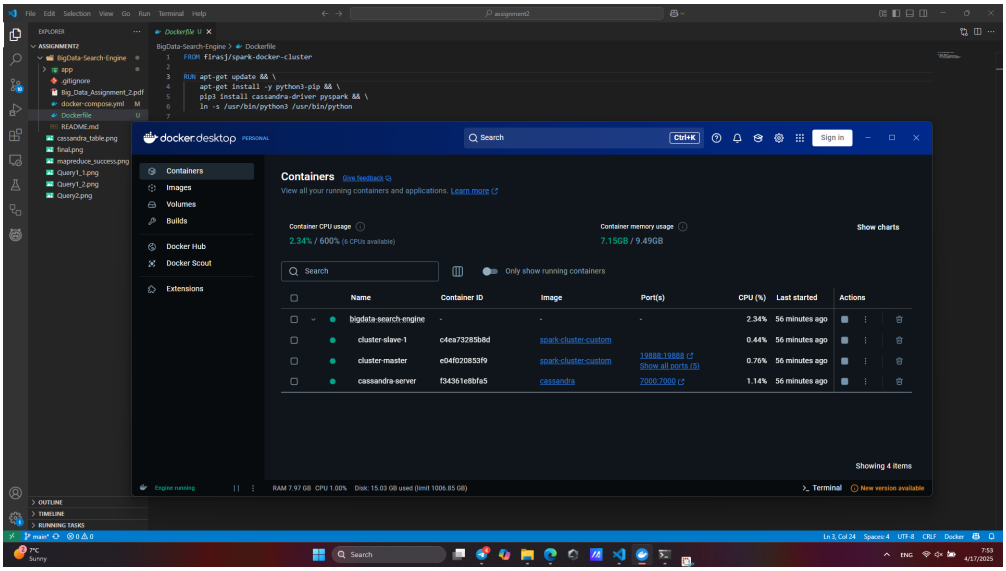

Figure 1: Running docker container

Indexing Documents with Hadoop MapReduce

The following screenshot shows the successful execution of the MapReduce pipeline over 993 documents stored in HDFS. The mapper and reducer jobs were executed in a fully distributed environment using Hadoop YARN.

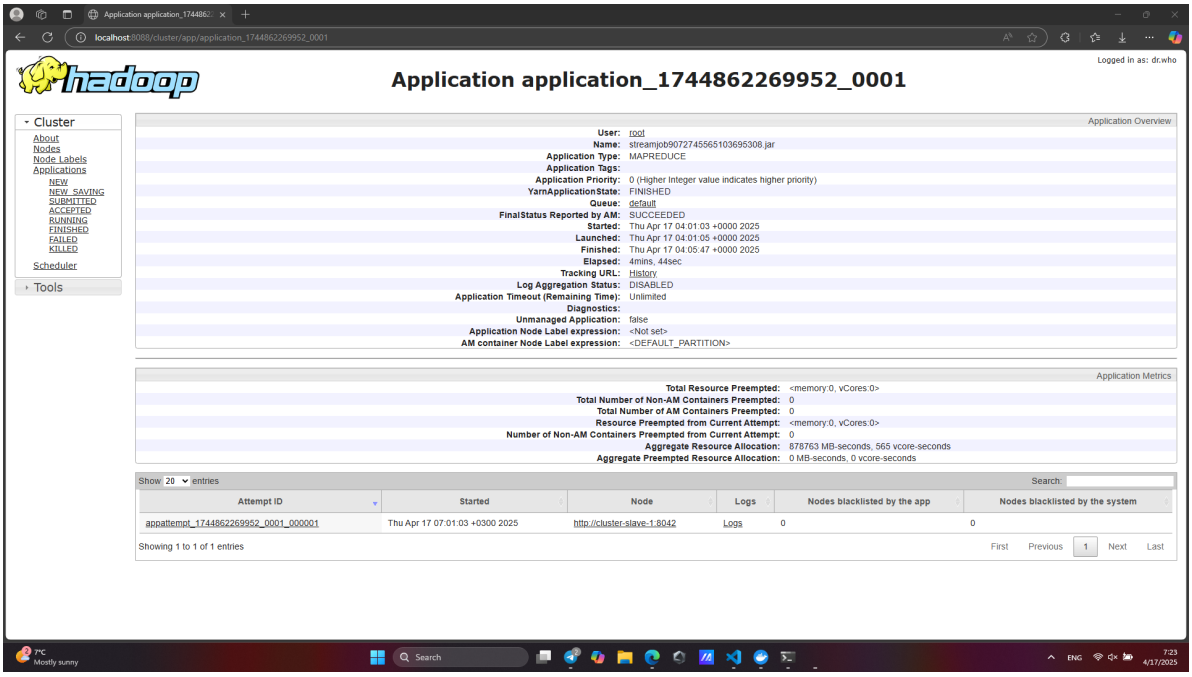

Figure 2: Hadoop YARN UI: MapReduce job completed successfully

Inverted Index Stored in Cassandra

After the reducer finishes, the term frequencies are stored in the `inverted_index` table in the Cassandra database. Below is a screenshot of a sample query run using `cqlsh`.

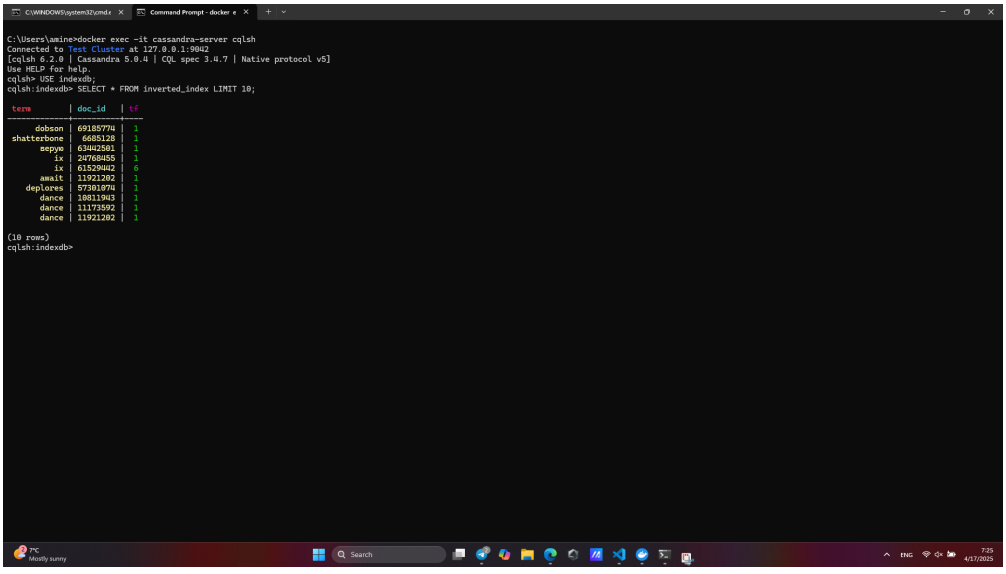

Figure 3: Sample rows from inverted_index table in Cassandra

Search Results for Example Queries

The following screenshots show the results of running the BM25-based search engine using `query.py` and retrieving the top 10 most relevant documents for each query.

Query 1: `search engine indexing`

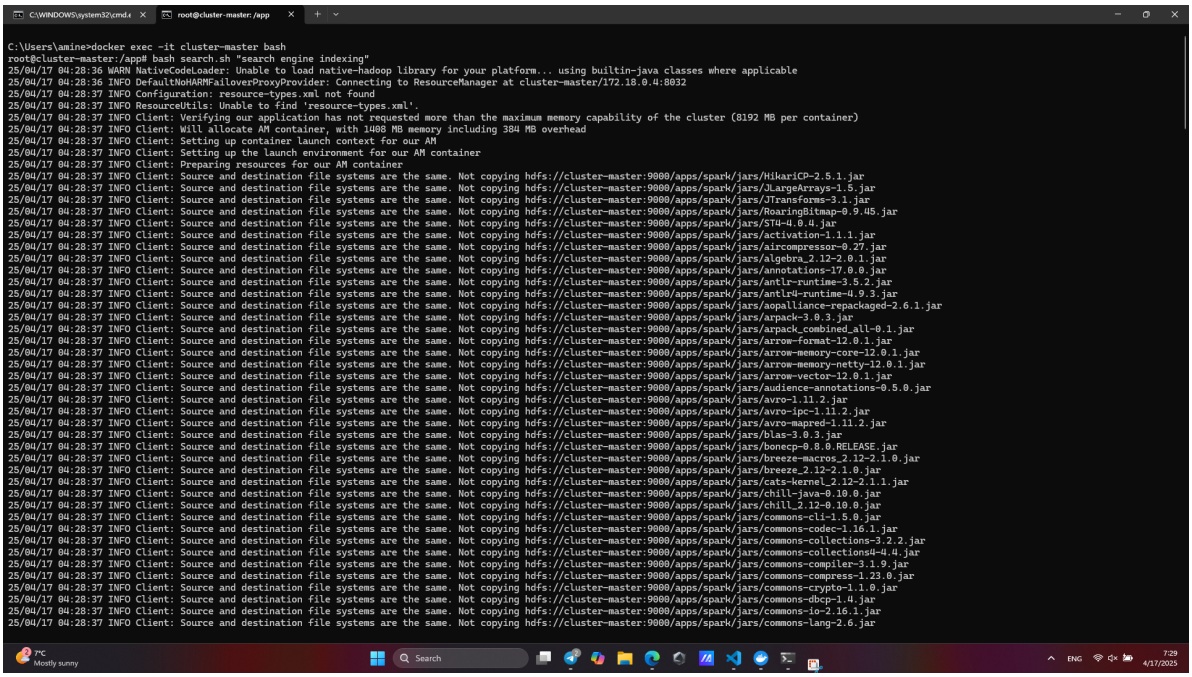

Figure 4: Bash results for query: ”search engine indexing”

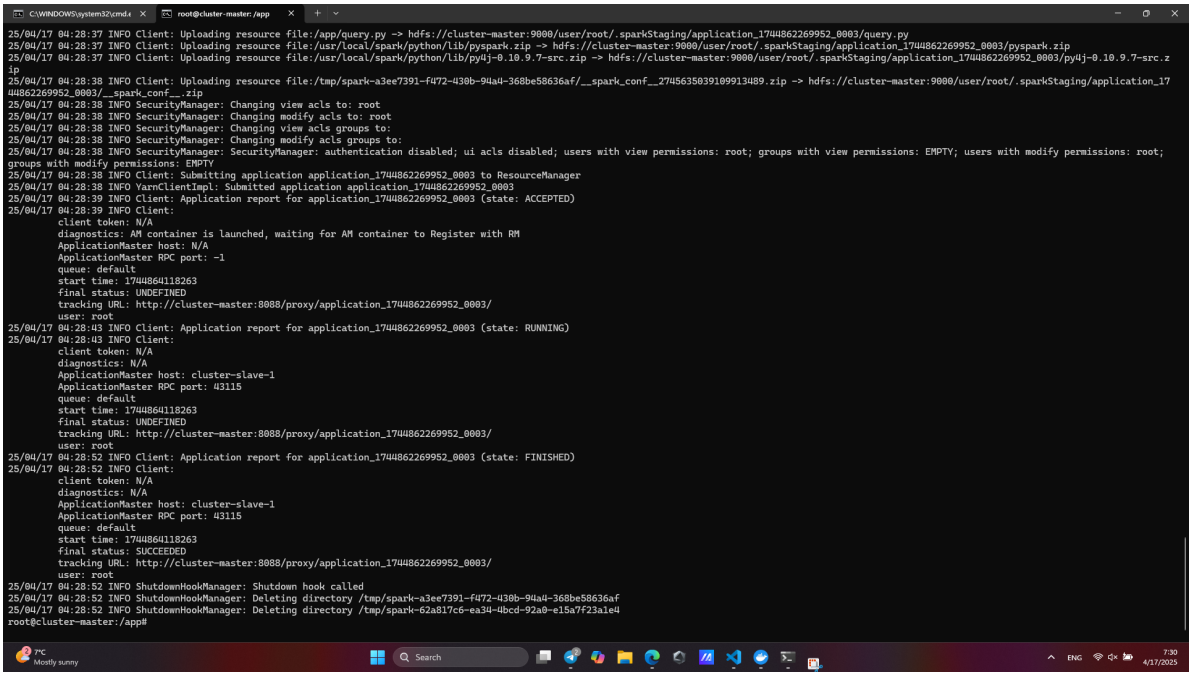

Figure 5: Continuation of bash results for query: "search engine indexing"

Query 2: `natural language processing`

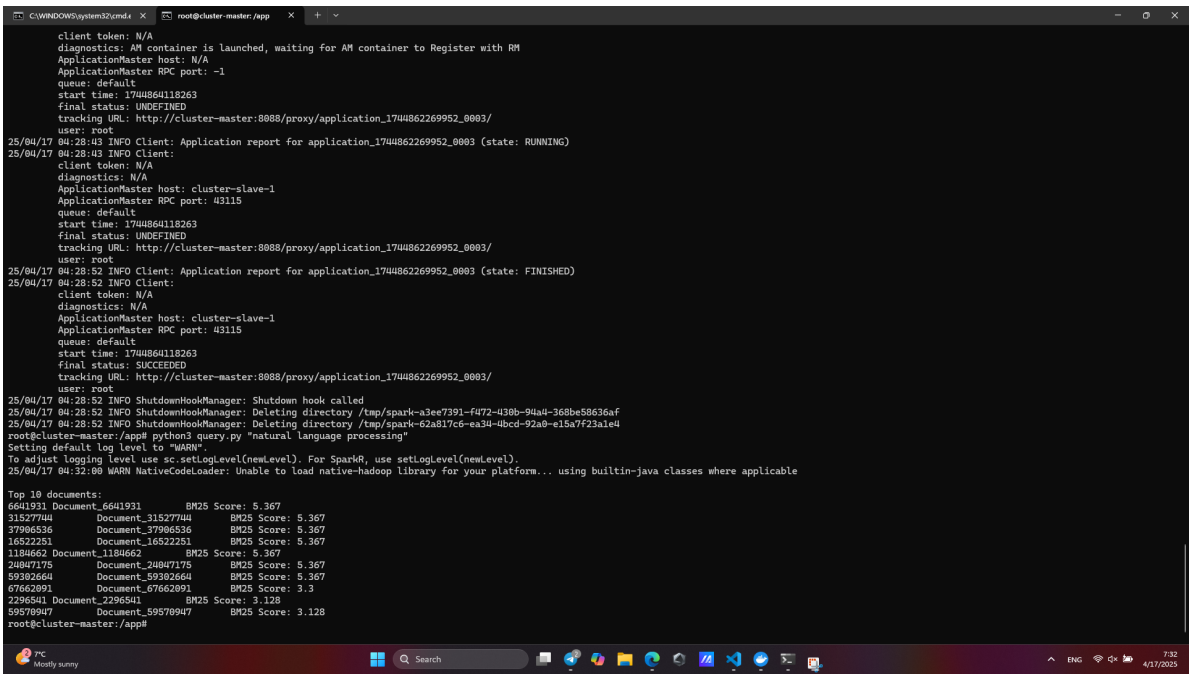

Figure 6: BM25 scores for query: "natural language processing"

Dashboard

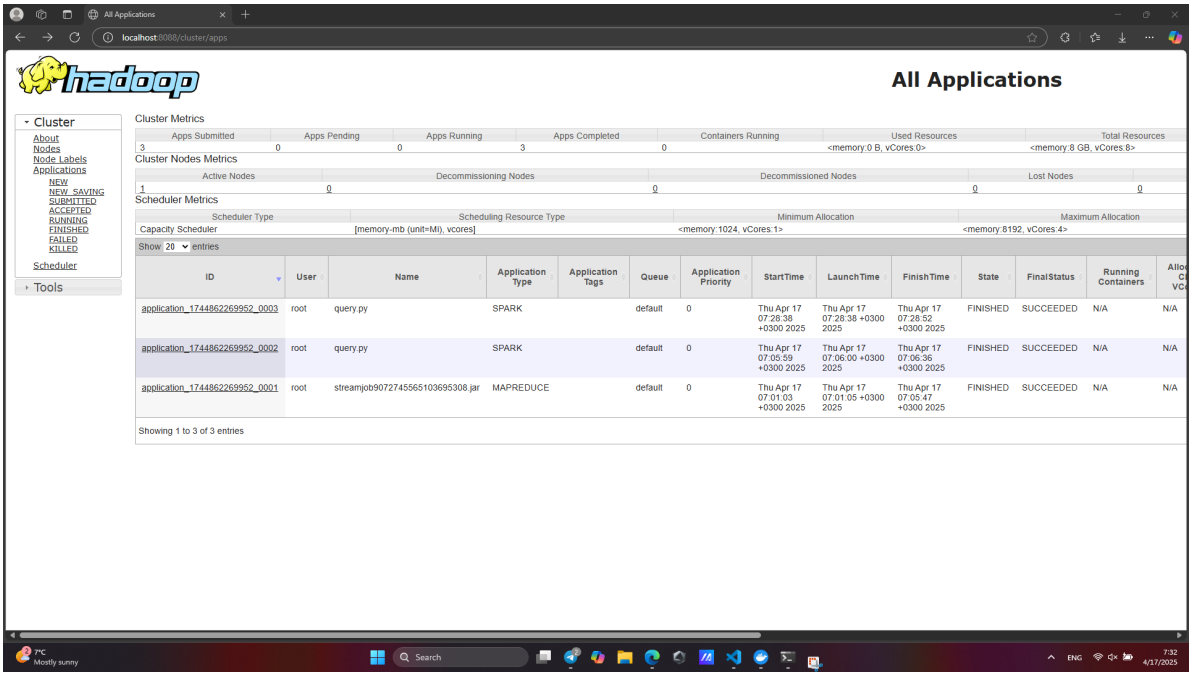

Figure 7: Successful Execution of Applications

Observations

- For the container to work, I had to keep increasing resources. I ended up with a setup of 10GB of ram allocated for docker and 6 CPU cores.
- I have faced an issue with pip3 missing, and cassandra-driver and pyspark not installed. I fixed this by adding a dockerfile with installation commands in the image.

```
FROM firasj/spark-docker-cluster

RUN apt-get update && \
    apt-get install -y python3-pip && \
    pip3 install cassandra-driver pyspark && \
    ln -s /usr/bin/python3 /usr/bin/python
```