

Big Data Assignment 2 - Search Engine on Hadoop and Cassandra

Osama Orabi

April 20, 2025

1. Methodology

This project implements a basic search engine using a big data pipeline with Hadoop MapReduce, Apache Spark, and Cassandra, all orchestrated via Docker.

Architecture Overview

The search engine pipeline consists of the following components:

- **Data Preparation:** Converts documents from a Parquet file to plain text.
- **Indexing (MapReduce):** Builds an inverted index using Hadoop Streaming with custom mapper and reducer scripts.
- **Index Storage (Cassandra):** Stores document metadata, postings, and term statistics for fast retrieval.
- **Query Engine (PySpark):** Processes a search query and computes BM25 scores to rank documents.

Data Flow

1. Documents are extracted from a Parquet file using `prepare_data.py` and saved into the `data/` folder.
2. The data is uploaded to HDFS using `prepare_data.sh`.
3. `index.sh` runs a Hadoop Streaming job to compute term frequencies, document lengths, and inverted index structure.
4. The MapReduce output is stored in Cassandra via `app.py`.
5. `query.py` retrieves necessary statistics from Cassandra and computes BM25 scores for a given query using PySpark.

BM25 Ranking Formula

We used the BM25 formula:

$$score(q, d) = \sum_{i \in q} IDF(i) \cdot \frac{f_{i,d} \cdot (k + 1)}{f_{i,d} + k \cdot (1 - b + b \cdot \frac{dl}{avgdl})}$$

Where:

- $f_{i,d}$: frequency of term i in document d
- dl : document length
- $avgdl$: average document length
- $k = 1.5, b = 0.75$

Environment and Setup

The system uses Docker Compose to deploy:

- Spark Hadoop master node (driver and resource manager)
- One Spark slave node
- Cassandra server

All dependencies are installed in a virtual environment and packed with `venv-pack` to run on YARN.

2. Demonstration

Running the System

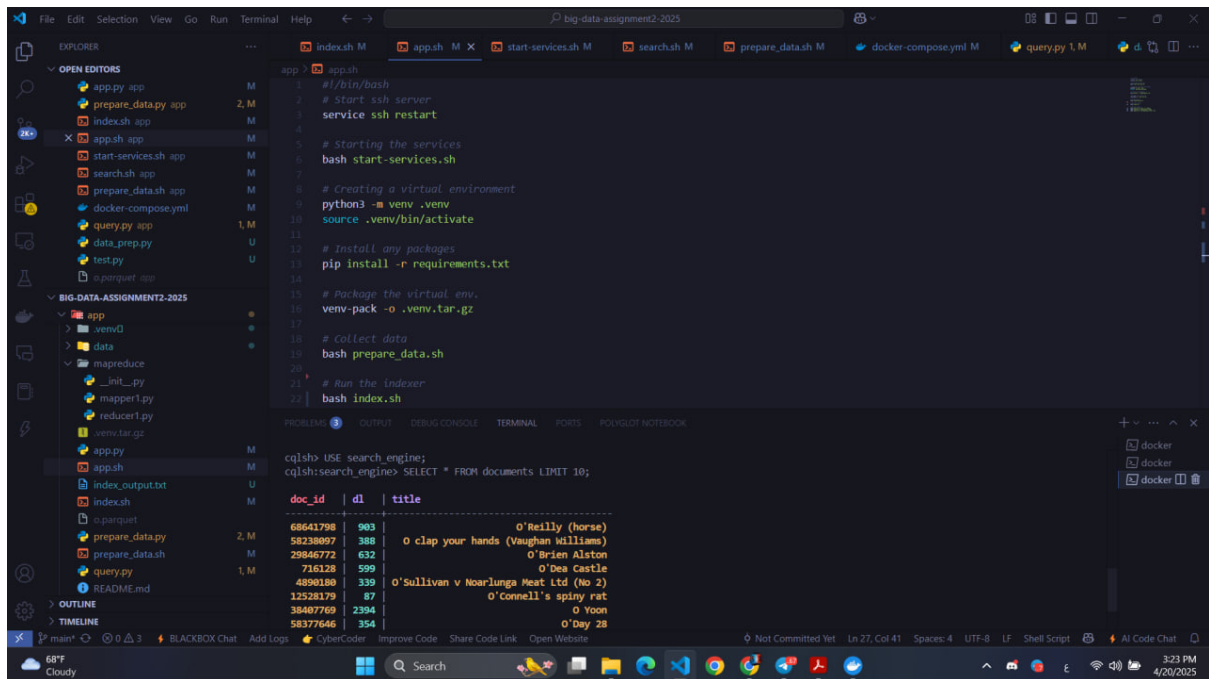
To run the system, execute:

```
docker compose up
```

This starts three containers: master, worker, and Cassandra. The master container runs the full pipeline defined in `app.sh`, which:

- Installs dependencies
- Runs the data preparation and HDFS upload
- Runs MapReduce indexing
- Loads index into Cassandra
- Executes three example queries

Screenshot 1: Successful Indexing



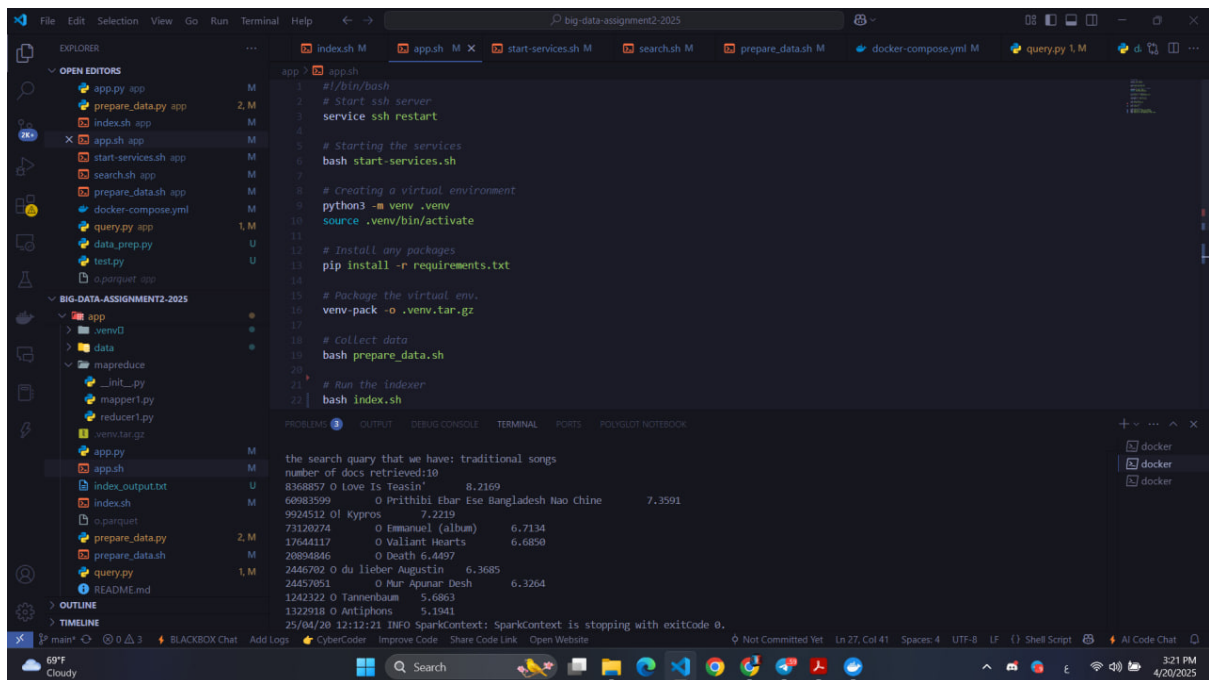
```
app > app.sh
1 #!/bin/bash
2 # Start ssh server
3 service ssh restart
4
5 # Starting the services
6 bash start-services.sh
7
8 # Creating a virtual environment
9 python3 -m venv .venv
10 source .venv/bin/activate
11
12 # Install any packages
13 pip install -r requirements.txt
14
15 # Package the virtual env.
16 venv-pack -o .venv.tar.gz
17
18 # Collect data
19 bash prepare_data.sh
20
21 # Run the indexer
22 bash index.sh
```

```
curlsh> USE search_engine;
curlsh:search_engine> SELECT * FROM documents LIMIT 10;
```

doc_id	dl	title
68641798	903	O'Reilly (horse)
58238097	308	O clap your hands (Vaughan Williams)
29846772	632	O'Brien Aliston
716128	599	O'Dea Castle
4890180	339	O'Sullivan v Noarlunga Meat Ltd (No 2)
12528179	87	O'Connell's spiny rat
38407769	2394	O Yoon
58377646	354	O'Day 28

This screenshot shows that documents were successfully indexed and uploaded into Cassandra.

Screenshot 2: Query Result – "traditional songs"



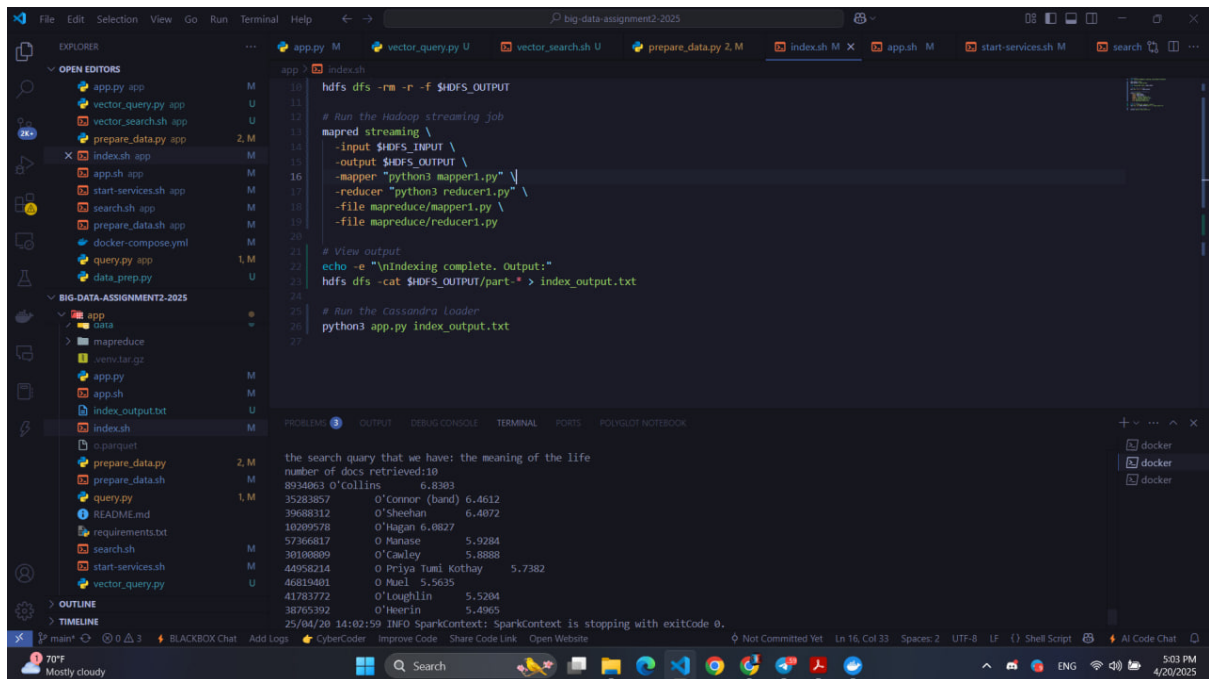
```
app > app.sh
1 #!/bin/bash
2 # Start ssh server
3 service ssh restart
4
5 # Starting the services
6 bash start-services.sh
7
8 # Creating a virtual environment
9 python3 -m venv .venv
10 source .venv/bin/activate
11
12 # Install any packages
13 pip install -r requirements.txt
14
15 # Package the virtual env.
16 venv-pack -o .venv.tar.gz
17
18 # Collect data
19 bash prepare_data.sh
20
21 # Run the indexer
22 bash index.sh
```

```
the search query that we have: traditional songs
number of docs retrieved:10
8368857 O Love Is Teasin' 8.2169
60983599 O Prithibi Ebar Ese Bangladesh Nao Chine 7.3591
9924512 Ol Kypros 7.2219
73120274 O Emmanuel (album) 6.7134
17644117 O Valiant Hearts 6.6850
20894846 O Death 6.4497
24467802 O du lieber Augustin 6.3685
24457951 O Mur Apunar Desh 6.3264
1242322 O Tannenbaum 5.6863
1322918 O Antiphons 5.1941
```

```
25/04/20 12:12:21 INFO SparkContext: SparkContext is stopping with exitCode 0.
```

Sample output showing top 10 ranked documents using BM25.

Screenshot 3: Query Result – "the meaning of the life"



```
hdfs dfs -rm -r -f $HDFS_OUTPUT
# Run the Hadoop streaming job
mapred streaming \
  -input $HDFS_INPUT \
  -output $HDFS_OUTPUT \
  -mapper "python3 mapper1.py" \
  -reducer "python3 reducer1.py" \
  -file mapreduce/mapper1.py \
  -file mapreduce/reducer1.py
# View output
echo -e "\nindexing complete. Output:"
hdfs dfs -cat $HDFS_OUTPUT/part-* > index_output.txt
# Run the Cassandra Loader
python3 app.py index_output.txt
```

the search query that we have: the meaning of the life
number of docs retrieved:10
8934963 O'collins 6.8383
35283857 O'Connor (band) 6.4612
39688312 O'Sheehan 6.4872
10209578 O'Hagan 6.0827
57366817 O'Rourke 5.9284
30108809 O'Cauley 5.8888
44958214 O Priya Tumi Kothay 5.7382
46819481 O Muel 5.5635
41783772 O'Loughlin 5.5204
38765392 O'Heerin 5.4965

Explanation of Results

The retrieved documents for queries like traditional songs or the meaning of the life show relevant matches, with higher BM25 scores assigned to documents that contain those query terms more frequently and concisely.

Reflection

This project demonstrates the power of distributed systems in building scalable search engines. Challenges included integrating multiple tools (HDFS, Spark, Cassandra) and optimizing BM25 efficiently. In future work, we can:

- Support phrase search
- Add stemming/token normalization
- Implement a frontend interface