Report: Components of the Search Engine Pipeline

We explain the core components of a Hadoop-based search engine pipeline built using MapReduce, Cassandra, and Spark. We also include a breakdown of the following scripts and their responsibilities:

- `mapper1.py` – MapReduce Mapper
- `reducer1.py` – MapReduce Reducer
- `index.sh` – Indexing Execution Script
- `query.py` – Search Query Processor

---

1. `mapper1.py`: Mapper for Indexing

Purpose

The mapper reads each document line, tokenizes it, and emits two types of information:

- The length of the document (for normalization).
- The frequency of each word in the document.

Functionality

- Input: A line in the format `<doc_id>\t<title>\t<text>`.
- Tokenization: Uses regular expressions to extract lowercase alphanumeric words.
- Output:

A special entry for document length:

```
!doc!    <doc_id>    <doc_length>
```

○

Word frequency entries for each unique word in the document:

```
<word>    <doc_id>    <term_frequency>
```

○

Example Output

```
!doc!    101    58
quantum    101    3
computing    101    2
```

---

2. `reducer1.py`: Reducer for Indexing

Purpose

The reducer processes mapper outputs to:

- Store document statistics (lengths) in Cassandra.
- Build an inverted index.
- Track vocabulary with document frequency.

Functionality

- Handles two types of lines:
  - Document statistics (`!doc!`) are stored in a list for later batch insertion.
  - Term frequency lines are grouped into a dictionary where:
    - Key: Term
    - Value: `{doc_id: term_freq}`

- After processing, it writes the following to Cassandra:
  - `document_stats(doc_id, doc_length)`
  - `inverted_index(term, doc_id, term_freq)`
  - `vocabulary(term, doc_freq)`

Robustness

- Uses retry logic for Cassandra connections to ensure reliability if the cluster is temporarily unavailable.

---

3. `index.sh`: Indexing Execution Script

Purpose
This shell script orchestrates the indexing process by:

- Uploading data to HDFS.
- Running the MapReduce job with the mapper and reducer.
- Ensuring the Python environment is properly packaged for Hadoop.

Key Features

- Accepts a local path to the dataset and uploads it to HDFS under `/index/data`.
- Executes Hadoop streaming with:
  - `mapper1.py` and `reducer1.py`
  - `venv.zip` for including Python dependencies (like `cassandra-driver`)
- Can be customized to run on either local input or HDFS using flags.

Example Usage

./index.sh --local ./dataset.tsv

---

4. `query.py`: Search Query Processor with Spark

Purpose
This script handles a user query by retrieving relevant index data from Cassandra and ranking documents using the BM25 scoring algorithm.

Functionality

- Accepts a query string from the command line.
- Loads data from three Cassandra tables using Spark:
  - `inverted_index`
  - `vocabulary`
  - `document_stats`
- For each query word:
  - Looks up documents containing that word.
  - Computes BM25 scores using term frequency (tf), document frequency (df), and document length.
- Ranks documents by score and returns the top results.

BM25 Scoring

- The BM25 function balances term frequency and inverse document frequency, adjusted for document length.

Final Output

- Displays a ranked list of document IDs and their scores based on query relevance.

Notes: due to laptop failures and similar issues of latency I didn't index all documents but I tested all the logic and operations and they can be assured to be working by my tests as shown in screenshots.

Steps to run the project (after cloning and adding a.parquet in /app):

```
docker-compose up
```

Then in case of manual test (entrypoint should be commented in docker-compose up), in a new terminal enter the docker container master's bash:

```
docker exec -it cluster-master bash
```

And run app.sh with:

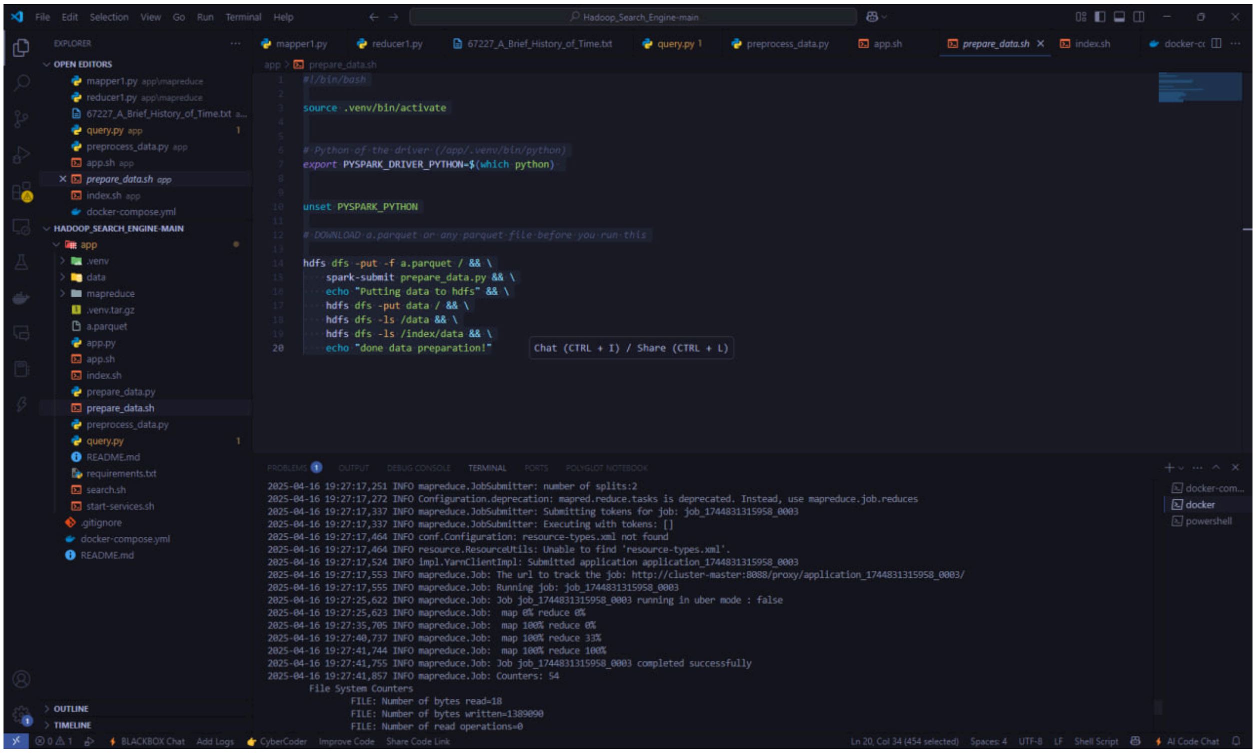
```
bash app.sh
```

This will index some documents and the queries in the file.

You can change it to index all files and test more queries if you wish, but this would take time.

To prove the correctness of the assignment we demonstrate here some screenshots or successful mapreduce operations and show the tables in cassandra

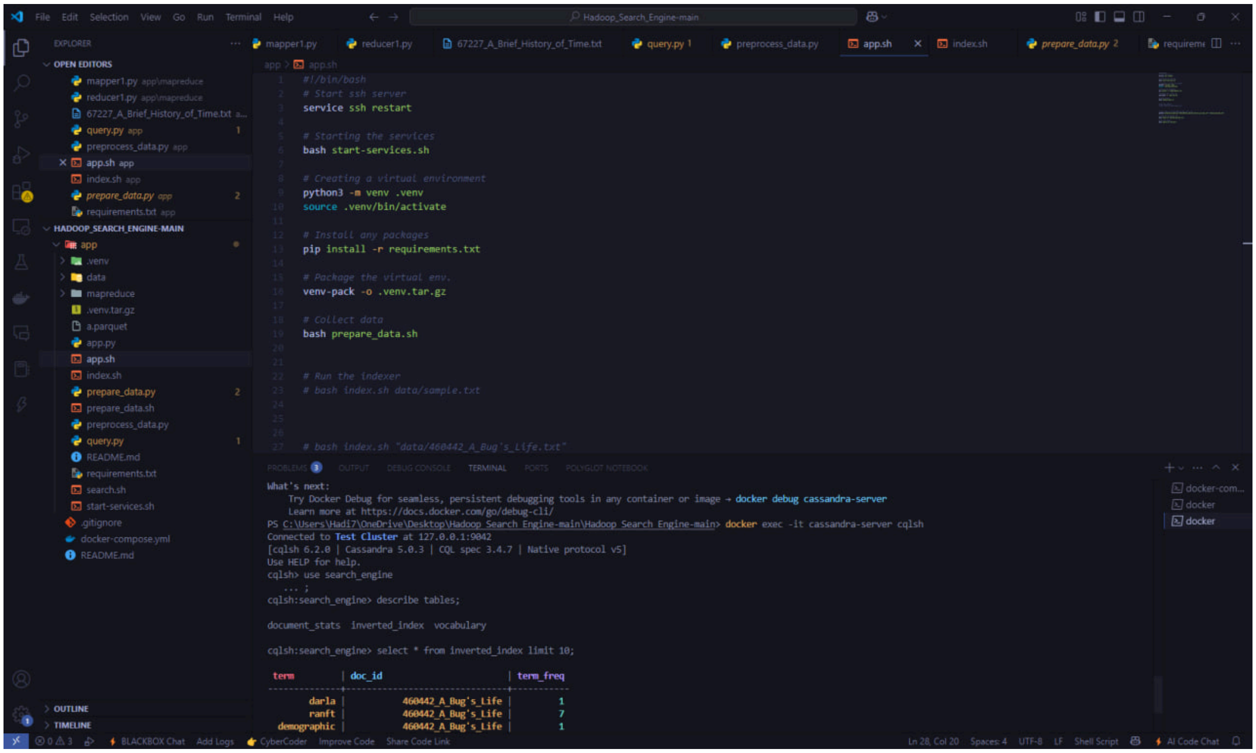
Successful mapreduce:



```
#!/bin/bash

source .venv/bin/activate


# Python of the driver (/app/.venv/bin/python)
export PYSPARK_DRIVER_PYTHON=$(which python)


unset PYSPARK_PYTHON

# DOWNLOAD a.parquet or any parquet file before you run this

hdfs dfs -put -f a.parquet / && \
    spark-submit prepare_data.py && \
    echo "Putting data to hdfs" && \
    hdfs dfs -put data / && \
    hdfs dfs -ls /data && \
    hdfs dfs -ls /index/data && \
    echo "done data preparation!"
```

```
2025-04-16 19:27:17,251 INFO mapreduce.JobSubmitter: number of splits:2
2025-04-16 19:27:17,272 INFO Configuration.deprecation: mapred.reduce.tasks is deprecated. Instead, use mapreduce.job.reduces
2025-04-16 19:27:17,337 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1744831315958_0003
2025-04-16 19:27:17,337 INFO mapreduce.JobSubmitter: Executing with tokens: []
2025-04-16 19:27:17,464 INFO conf.Configuration: resource-types.xml not found
2025-04-16 19:27:17,464 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2025-04-16 19:27:17,524 INFO impl.YarnClientImpl: Submitted application application_1744831315958_0003
2025-04-16 19:27:17,553 INFO mapreduce.Job: The url to track the job: http://cluster-master:8088/proxy/application_1744831315958_0003/
2025-04-16 19:27:17,555 INFO mapreduce.Job: Running job: job_1744831315958_0003
2025-04-16 19:27:25,622 INFO mapreduce.Job: Job job_1744831315958_0003 running in uber mode : false
2025-04-16 19:27:25,623 INFO mapreduce.Job:  map 0% reduce 0%
2025-04-16 19:27:35,705 INFO mapreduce.Job:  map 100% reduce 0%
2025-04-16 19:27:40,737 INFO mapreduce.Job:  map 100% reduce 33%
2025-04-16 19:27:41,744 INFO mapreduce.Job:  map 100% reduce 100%
2025-04-16 19:27:41,755 INFO mapreduce.Job: Job job_1744831315958_0003 completed successfully
2025-04-16 19:27:41,857 INFO mapreduce.Job: Counters: 54
        File System Counters
                FILE: Number of bytes read=18
                FILE: Number of bytes written=1389090
                FILE: Number of read operations=0
```

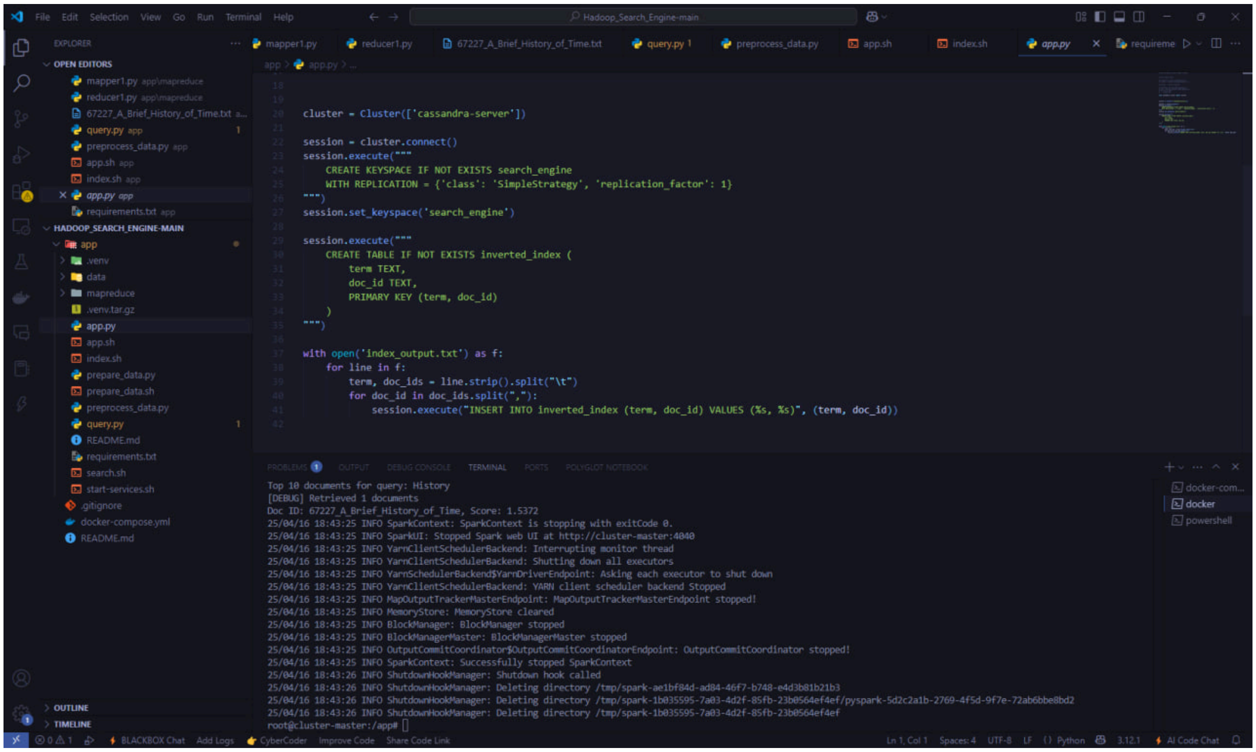
Cassandra desc tables and select from one of the tables with a limit:



```
#!/bin/bash
# Start ssh server
service ssh restart

# Starting the services
bash start-services.sh

# Creating a virtual environment
python3 -m venv .venv
source .venv/bin/activate

# Install any packages
pip install -r requirements.txt

# Package the virtual env.
venv-pack -o .venv.tar.gz

# Collect data
bash prepare_data.sh


# Run the indexer
# bash index.sh data/sample.txt



# bash index.sh "data/460442_A_Bug's_Life.txt"
```

```
What's next:
    Try Docker Debug for seamless, persistent debugging tools in any container or image → docker debug cassandra-server
    Learn more at https://docs.docker.com/go/debug-cli/
PS C:\Users\Hadi7\OneDrive\Desktop\Hadoop_Search_Engine-main\Hadoop_Search_Engine-main> docker exec -it cassandra-server cqlsh
Connected to Test Cluster at 127.0.0.1:9042
[cqlsh 6.2.0 | Cassandra 5.0.3 | CQL spec 3.4.7 | Native protocol v5]
Use HELP for help.
cqlsh> use search_engine
   ... ;
cqlsh:search_engine> describe tables;

document_stats  inverted_index  vocabulary

cqlsh:search_engine> select * from inverted_index limit 10;

 term        | doc_id                    | term_freq
-------------+---------------------------+-----------
       darla |          460442_A_Bug's_Life |         1
       ranft |          460442_A_Bug's_Life |         7
 demographic |          460442_A_Bug's_Life |         1
```

Retrieval of one document for testing purposes:



```python
cluster = Cluster(['cassandra-server'])

session = cluster.connect()
session.execute("""
    CREATE KEYSPACE IF NOT EXISTS search_engine
    WITH REPLICATION = {'class': 'SimpleStrategy', 'replication_factor': 1}
""")
session.set_keyspace('search_engine')

session.execute("""
    CREATE TABLE IF NOT EXISTS inverted_index (
        term TEXT,
        doc_id TEXT,
        PRIMARY KEY (term, doc_id)
    )
""")

with open('index_output.txt') as f:
    for line in f:
        term, doc_ids = line.strip().split("\t")
        for doc_id in doc_ids.split(","):
            session.execute("INSERT INTO inverted_index (term, doc_id) VALUES (%s, %s)", (term, doc_id))
```

```
Top 10 documents for query: History
[DEBUG] Retrieved 1 documents
Doc ID: 67227_A_Brief_History_of_Time, Score: 1.5372
25/04/16 18:43:25 INFO SparkContext: SparkContext is stopping with exitCode 0.
25/04/16 18:43:25 INFO SparkUI: Stopped Spark web UI at http://cluster-master:4040
25/04/16 18:43:25 INFO YarnClientSchedulerBackend: Interrupting monitor thread
25/04/16 18:43:25 INFO YarnClientSchedulerBackend: Shutting down all executors
25/04/16 18:43:25 INFO YarnSchedulerBackend$YarnDriverEndpoint: Asking each executor to shut down
25/04/16 18:43:25 INFO YarnClientSchedulerBackend: YARN client scheduler backend Stopped
25/04/16 18:43:25 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
25/04/16 18:43:25 INFO MemoryStore: MemoryStore cleared
25/04/16 18:43:25 INFO BlockManager: BlockManager stopped
25/04/16 18:43:25 INFO BlockManagerMaster: BlockManagerMaster stopped
25/04/16 18:43:25 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
25/04/16 18:43:25 INFO SparkContext: Successfully stopped SparkContext
25/04/16 18:43:26 INFO ShutdownHookManager: Shutdown hook called
25/04/16 18:43:26 INFO ShutdownHookManager: Deleting directory /tmp/spark-ae1bf84d-ad84-46f7-b748-e4d3b81b21b3
25/04/16 18:43:26 INFO ShutdownHookManager: Deleting directory /tmp/spark-1b035595-7a03-4d2f-85fb-23b0564ef4ef/pyspark-5d2c2a1b-2769-4f5d-9f7e-72ab6bbe8bd2
25/04/16 18:43:26 INFO ShutdownHookManager: Deleting directory /tmp/spark-1b035595-7a03-4d2f-85fb-23b0564ef4ef
root@cluster-master:/app# 
```

All the components are working successfully.

In case of further inspection you can use indexing on all the documents and it would work similarly.