Big Data Search Engine Implementation

Methodology

Data Preparation

The script samples 1000 documents, normalizes their titles, creates text files in a local directory, and writes the dataset to HDFS for MapReduce processing

```
# Normalize document text and create files
def normalize_text(text):
    return re.sub(r'\s+', ' ', text).strip()

# Create sanitized filenames for HDFS compatibility
title = row['title'].encode('ascii', 'ignore').decode('ascii')
title = re.sub(r'[^a-zA-Z0-9_\-.]', '_', title)
```

Using prepare_data.py provided by Firas Jolha, I faced the problem: many of titles contain non-ascii symbols, so I decided to substitude them

```
PS C:\Users\ahmat\OneDrive\Документы\big-data-assignment2-2025> & C:/Python313/python.exe c:/Users/ahmat/OneDrive/Документы/big-data-assignment2-2025/test.py
Found 981 files in HDFS /data directory
Found 1000 files in local app/data directory

Found 22 files in local directory missing from HDFS:
 - 14404655_A_Coruña_(Congress_of_Deputies_constituency).txt
 - 22186504_A_Caridá.txt
 - 25475273_A_Dolorosa_Raiz_do_Micondó.txt
 - 26928604_A_Canção_da_Saudade.txt
 - 27534215_A_Corazón_Abierto.txt
 - 34933447_A_Jamaâ.txt
 - 36902574_A_Dónde.txt
 - 37789341_A_History_(1982–1985).txt
 - 37789456_A_History_(1986–1989).txt
 - 42546120_A_Grupė.txt
 - 42585025_A_Grande_Vitória.txt
 - 44388687_A_Checklist_of_Painters_c1200-1994.txt
 - 46577795_A_Coruña_railway_station.txt
 - 50699812_A_Chinese-English_Dictionary.txt
 - 50813860_A_History_of_Garage_and_Frat_Bands_in_Memphis_1960-1975,_Volume_1.txt
 - 60102787_A_Jakállan_Intrigue.txt
 - 61041026_A_Dona_do_Pedaço.txt
 - 65213306_A_Droga_da_Obediência.txt
 - 65926201_A_Franklin_kézi_lexikona.txt
 - 7013812_A_Gudiña.txt
 - 73467161_A_History_of_the_Negro_Troops_in_the_War_of_the_Rebellion,_1861-1865.txt
 - 7662078_A_Just_Russia_–_For_Truth.txt

Found 3 files in HDFS not present in local directory:
 - 19860998_A_Converted_British_Family_Sheltering_a_Christian_Missionary_from_thePersecution_of_the_Druids.txt
 - 6773012_A_Huge_Ever_Growing_Pulsating_Brain_That_Rules_from_the_Centre_of_theUltraworld.txt
 - 69979031_A_History_of_Science,_Technology,_and_Philosophy_in_the_16th_and_17thCenturies.txt
```

Two-Stage MapReduce Indexing

1. **First Stage**: Tokenizes documents, counts term frequencies, outputs document lengths

```
for token, count in counter.items():
    print(f"{token}::{doc_id}\t{count}")
print(f"DOCLEN_{doc_id}\t{len(tokens)}")
```

2. **Second Stage**: Calculates inverse document frequency (IDF) for ranking

```
# for each term:
idf = math.log(total_docs / doc_count)
print(f"{term}\t{doc_count}\t{idf:.6f}")
```

Cassandra Storage

The system uses three Cassandra tables to store the index:

- `inverted_index`: Maps terms to documents with term frequency

```
CREATE TABLE IF NOT EXISTS inverted_index (
  term text,
  doc_id text,
  tf int,
  PRIMARY KEY (term, doc_id)
)
```

- `doc_stats`: Stores document metadata

```
CREATE TABLE IF NOT EXISTS doc_stats (
  doc_id text PRIMARY KEY,
  doc_length int,
  title text
)
```

- `term_stats`: Contains corpus statistics

```
CREATE TABLE IF NOT EXISTS term_stats (
  term text PRIMARY KEY,
  doc_count int,
  idf double
)
```

BM25 Search Algorithm

```
def calculate_bm25(tf, doc_len, avg_doc_len, idf, k1=1.0, b=0.75):
    """Calculate BM25 score for a term-document pair"""
    return idf * (tf * (k1 + 1)) / (tf + k1 * (1 - b + b * (doc_len /
avg_doc_len)))
```

The search component implements BM25 ranking using Spark, with optimizations like broadcast variables to efficiently share term statistics across worker nodes. Default k1 and b are basic - 1 and 0.75.

Demonstration

Running the System

The entire pipeline can be executed with a single command:

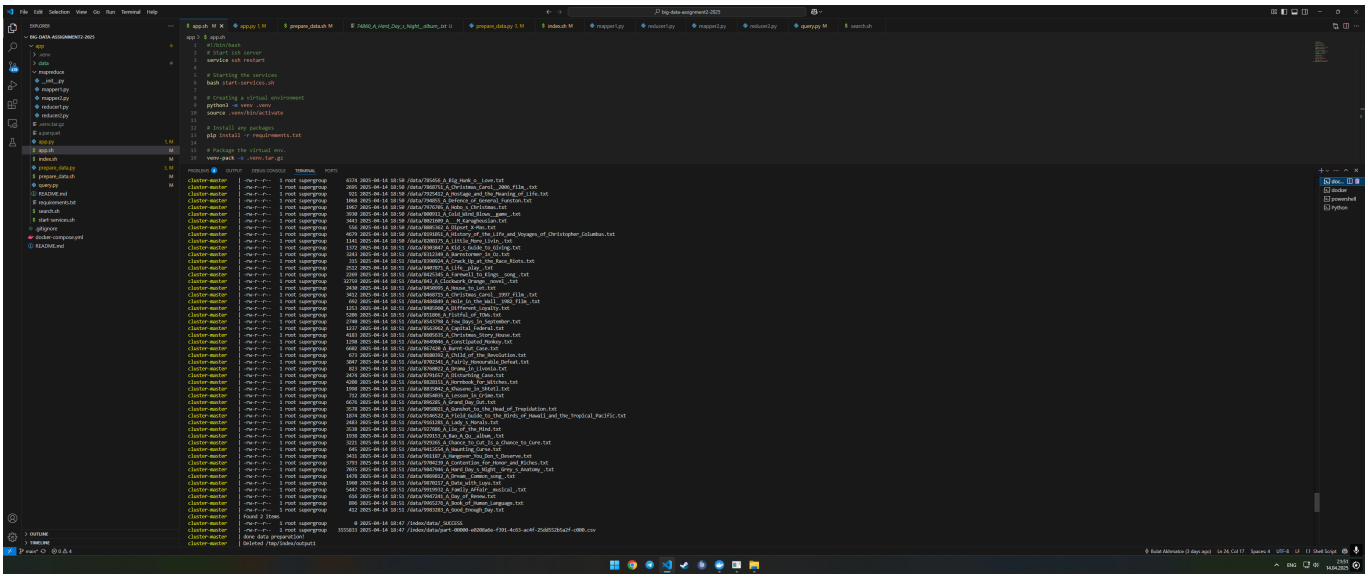
```
docker compose up
```

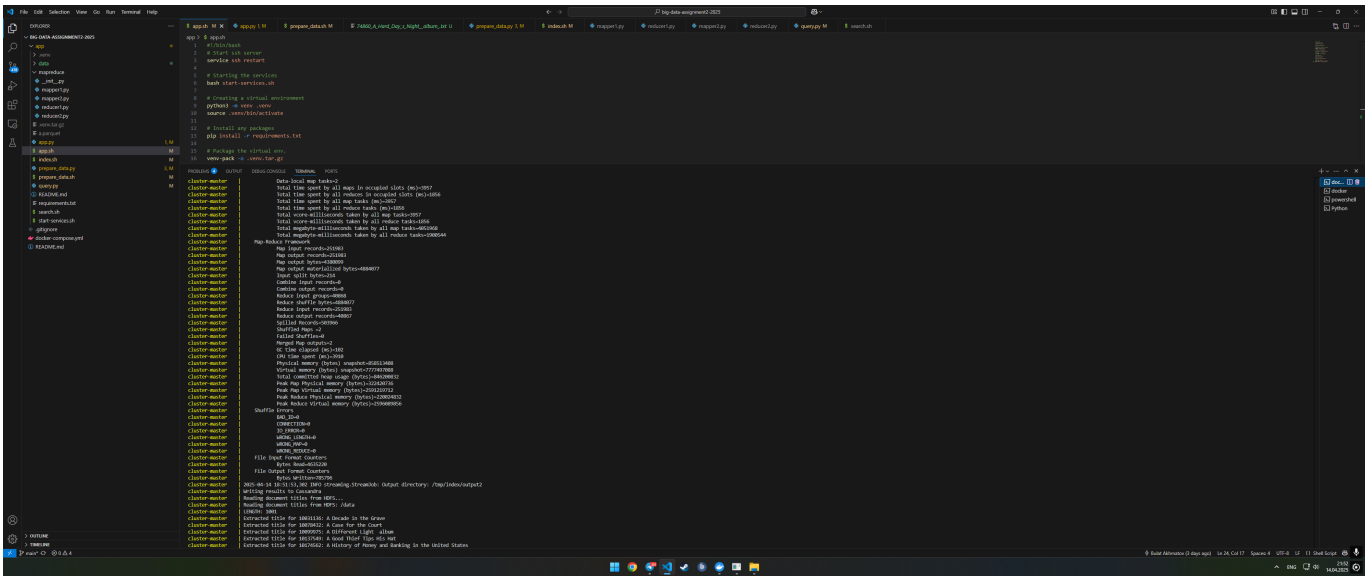
docker-compose.yml contains entrypoint to app/app.sh that:

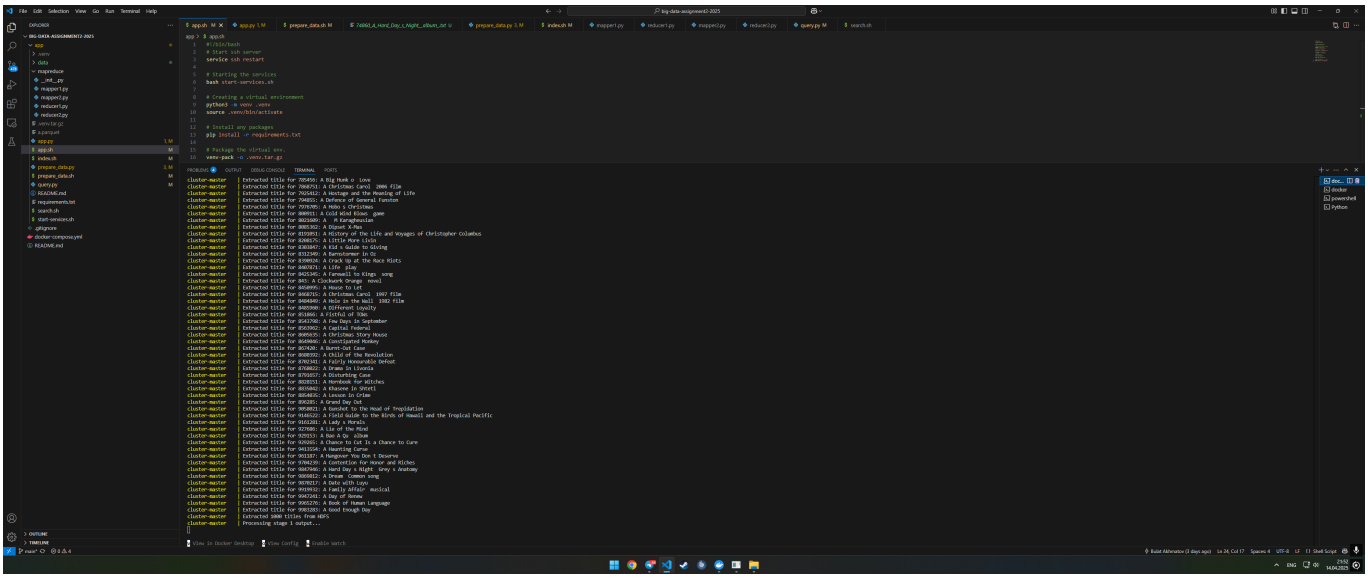
1. Starts required services
2. Sets up a Python virtual environment
3. Prepares document data (1000 documents)
4. Runs the MapReduce indexing pipeline
5. Executes sample searches

Screenshots and Results

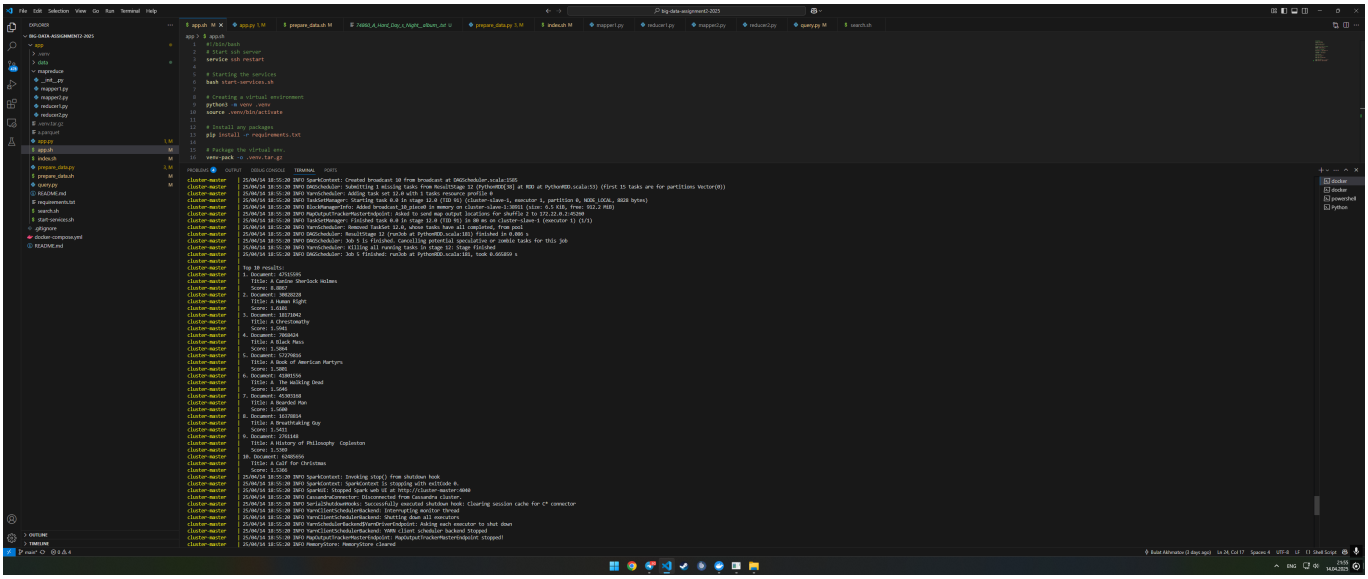
Data preparation and indexing:







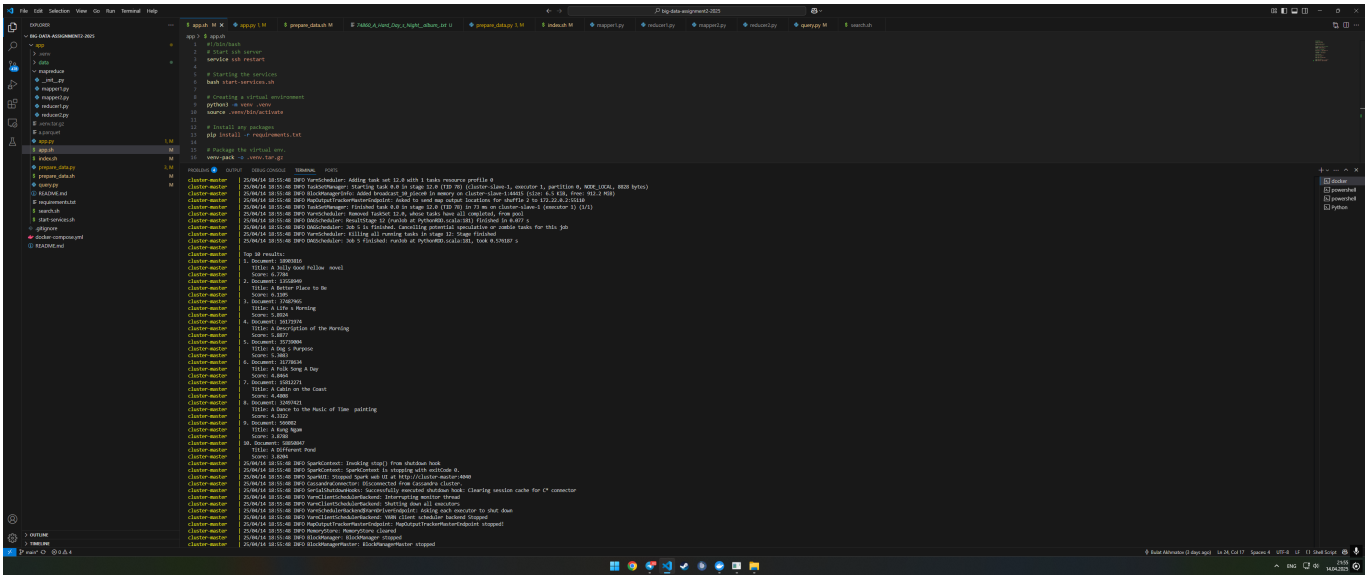
Search Results for "this is a query!":



First document contains 1 'query', 0 'this', 4 'is', 7 'a', and 0 '!'

Despite the fact that there are 0 'this' and '!', this words are much more common than 'query', so importance of word query should be high, and it is - score is much higher than for other documents, since they don't have this important for query word.

Search Results for "good morning":



First document contains 3 'good' and 1 'morning'

Second document contains the same amount of 'good' and 'morning' as first one, but number of words is higher - 634 against 484.

Key Findings

Search results for both queries are pretty interpretable.

The distributed architecture allows efficient processing of indexing and search. For me it was about 20 seconds for mapreduce and 0.5s for search. I think it's pretty good for cpu.

Challenges encountered included handling special characters in filenames and handling errors. I do like an ability to parallelize code with MapReduce, but this java errors aren't really interpretable. A lot of memory heap and garbage collector errors were encountered.