IBD Assignment 2 Report

Anna Gromova, DS-01

GitHub link: **https://github.com/anngrosha/hadoop-search**

Methodology

For this project, I implemented a simple search engine using Hadoop MapReduce, Cassandra, and Spark RDD. The system indexes documents and allows searching using either BM25 ranking or vector similarity search.

Data Preparation

I started by processing the Wikipedia dataset in Parquet format. The `prepare_data.py` script reads the Parquet file using PySpark, selects 1000 random documents, and saves each document as a separate text file. Each file is named using the document ID and title (with spaces replaced by underscores). The script also creates an input file for MapReduce in HDFS with tab-separated document ID, title, and content.

MapReduce Pipeline

I implemented two MapReduce jobs for indexing:

1. **Term Frequency Pipeline** (`mapper1.py`, `reducer1.py`):
   - The mapper tokenizes each document's text and emits (word, doc_id, term_count, doc_length) tuples
   - It also emits document metadata (!META! records)
   - The reducer collects all documents for each word and outputs them
2. **Document Frequency Pipeline** (`mapper2.py`, `reducer2.py`):
   - The mapper takes the term frequency output and emits (word, 1) for each document containing the word
   - The reducer counts how many documents contain each word (document frequency)

Cassandra Storage

The `app.py` script handles Cassandra operations:

- Creates keyspace and tables for document metadata, term index, vocabulary, and corpus statistics
- Loads data from MapReduce output into Cassandra
- Creates vector indexes when vector search is enabled

Tables include:

- `document_metadata`: Stores document IDs, titles, lengths and vector representations
- `term_index`: Stores term frequencies per document
- `vocabulary`: Stores document frequencies for all terms
- `corpus_stats`: Stores average document length and total document count

Search Implementation

The `query.py` script provides two search methods.

The **vector search** implementation works by:

1. Creating a vocabulary from all unique terms in the corpus
2. Representing each document as a vector where:
   - Each dimension corresponds to a vocabulary term
   - The value is the TF-IDF weight for that term in the document
3. Normalizing vectors to unit length
4. Representing queries the same way using the same vocabulary
5. Calculating cosine similarity between query and document vectors

The vectors are stored in Cassandra using the `VECTOR` type and indexed for efficient similarity search. This allows finding documents similar to a query even when they don't contain exact term matches.

The **BM25 ranking** considers:

- Term frequency in each document
- Document frequency of each term
- Document lengths and average document length
- The BM25 formula parameters (k1=1.2, b=0.75)

This provides better ranking than simple TF-IDF by accounting for document length normalization and term frequency saturation.

Both return 10 most relevant document in the end.

Optimizations

Key optimizations include using Cassandra's built-in vector indexing for fast similarity search, caching document titles during search to reduce database queries, batch processing of documents during vector creation, and proper connection handling and error recovery for Cassandra operations

The system demonstrates how different big data technologies can work together to build a functional search engine, with Hadoop for batch processing, Cassandra for storage, and Spark for query processing.

Demonstration

Running the Project

To run the project:

1. Place the `a.parquet` file in the app folder
2. Run `docker-compose up` (or `docker compose up`)

The system will:

- Start Hadoop, Spark and Cassandra services
- Process the Parquet file and create text documents
- Run the MapReduce indexing pipeline
- Load data into Cassandra
- Perform a sample search for "dog food"

Changing Queries

To change the search query:

1. Edit `app.sh` and modify the `bash search.sh` command
2. For example: `bash search.sh "computer science"`

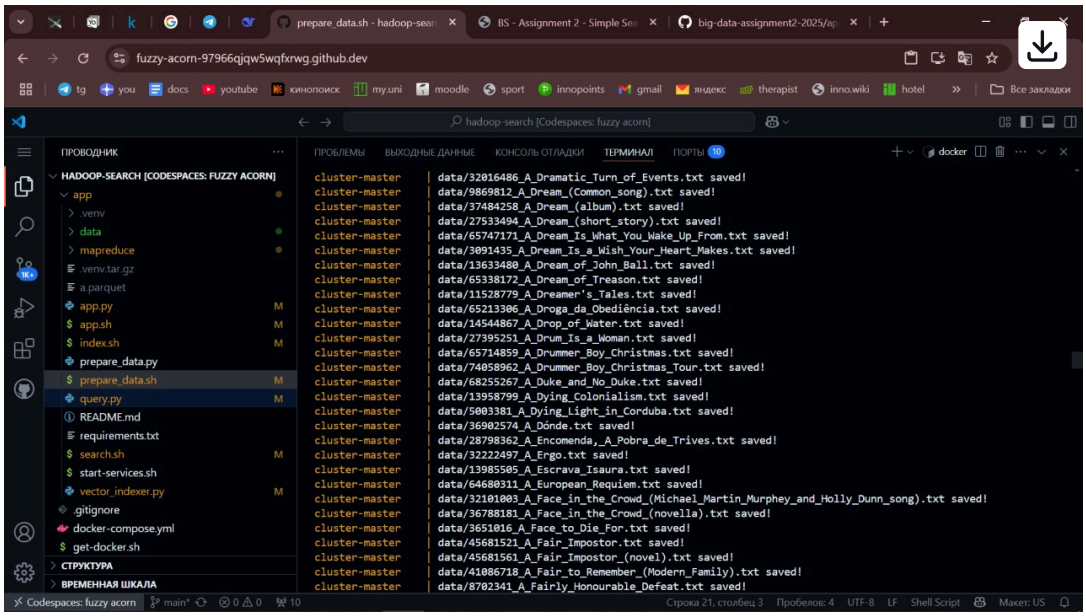
For vector search:

1. Edit `app.sh` to include `--vector` flag:

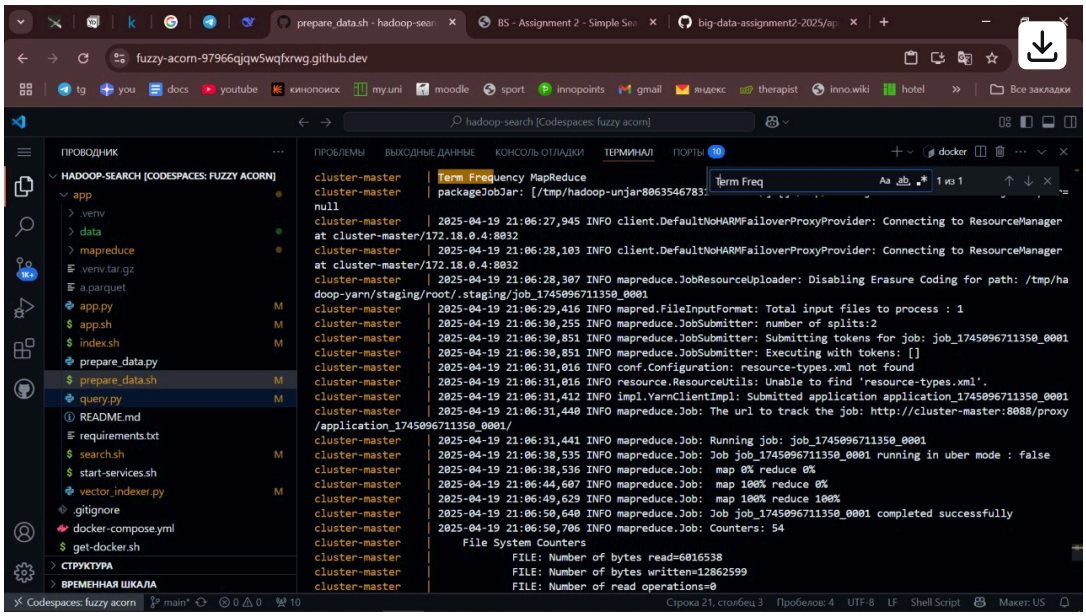
```
bash index.sh --vector
bash search.sh "<query>" --vector
```

Screenshots

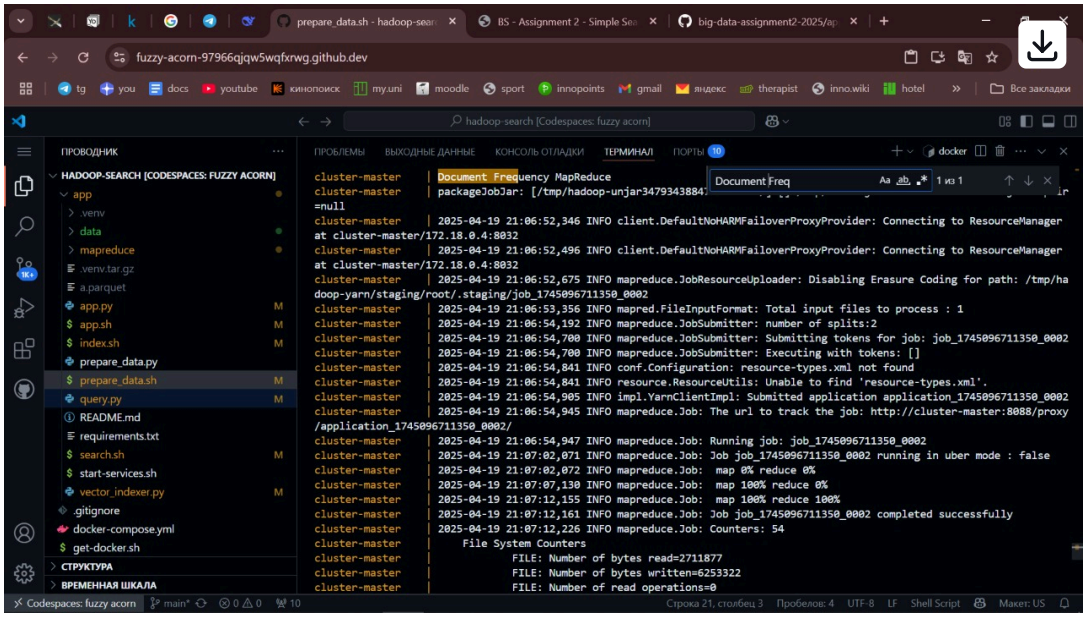
1. **Data Files Creation**:



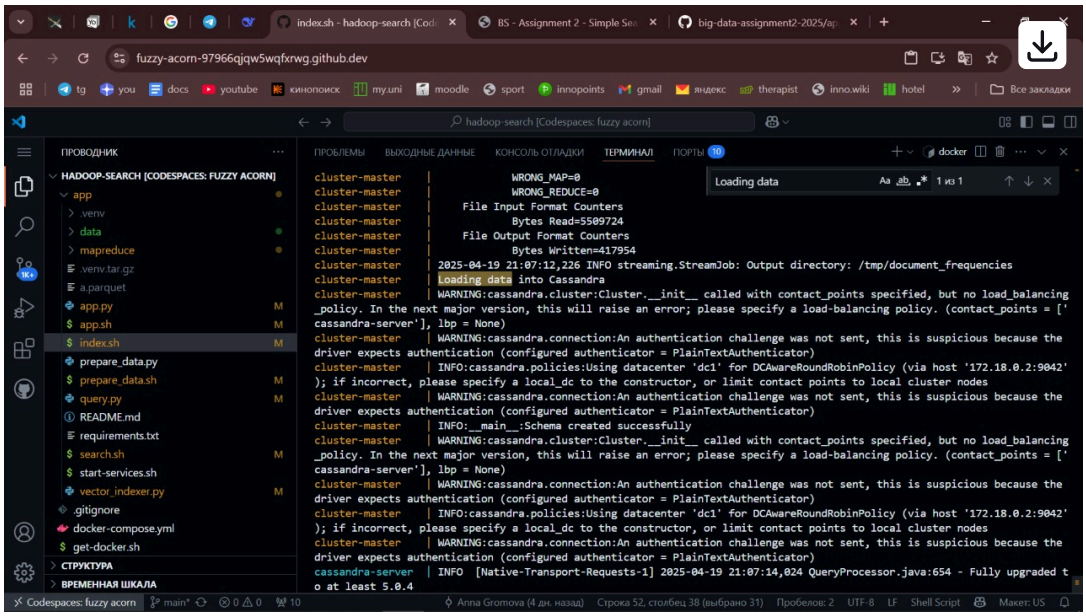
2. Term Frequency Pipeline:



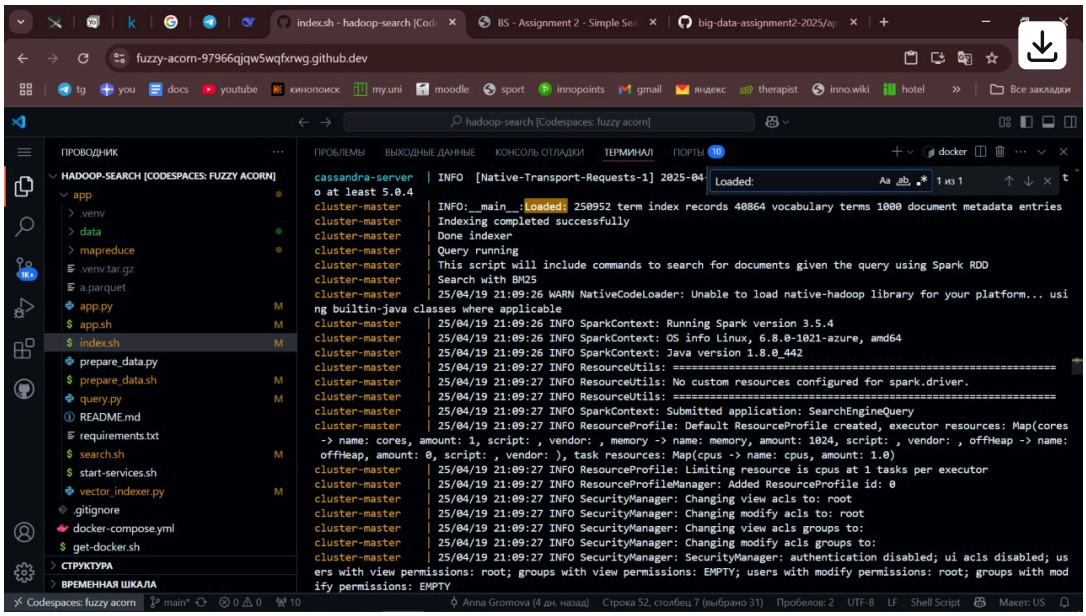
3. Document Frequency Pipeline:



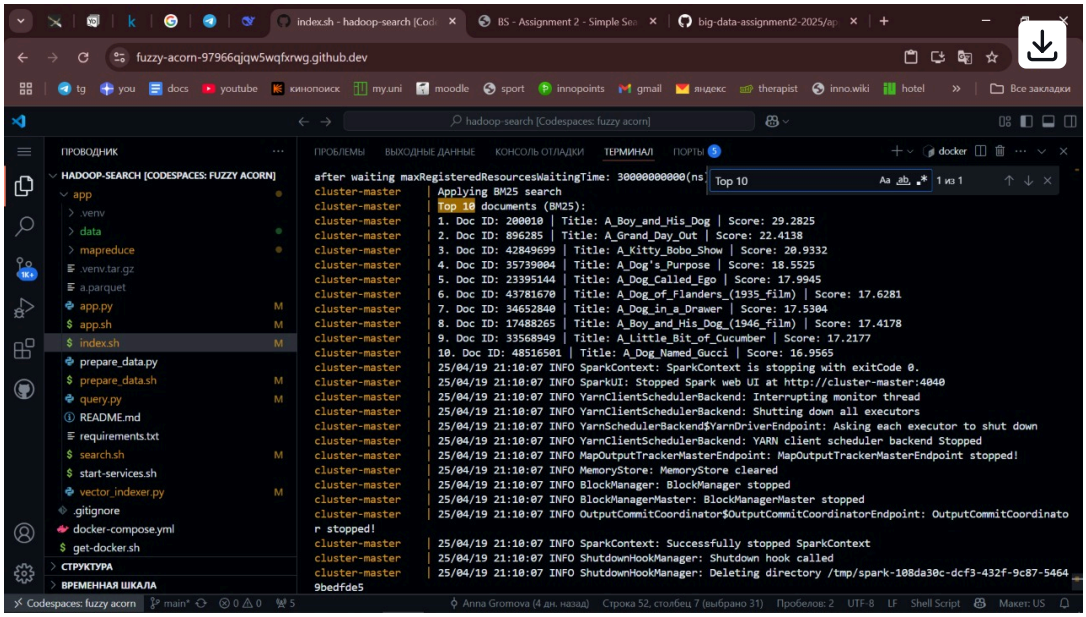
4. Loading Data to Cassandra:



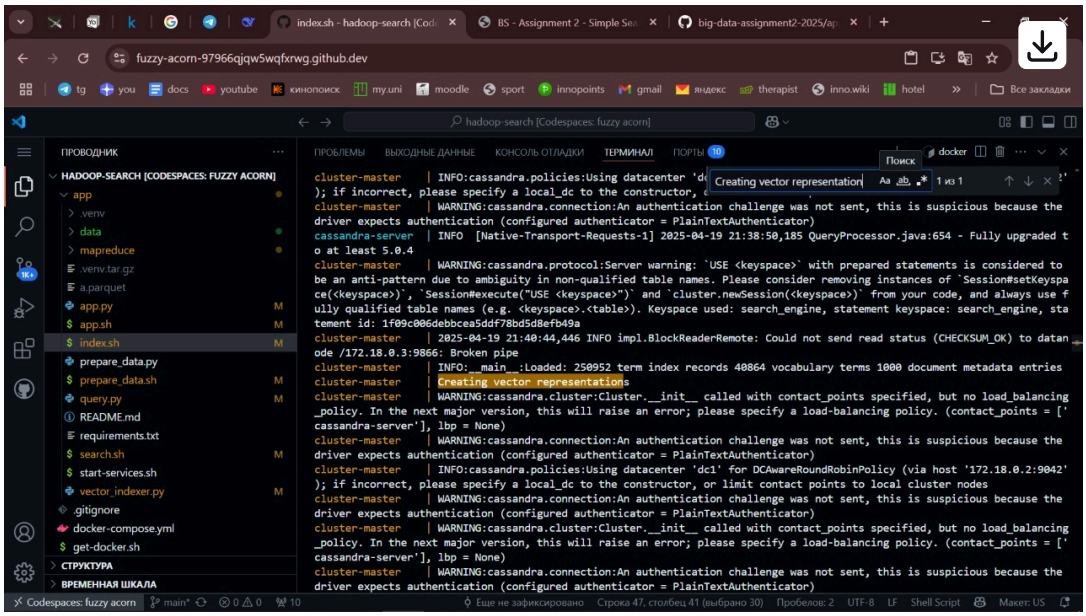
5. Cassandra Success Message:



6. Top 10 Documents:



7. Vector Creation:



Conclusion

The search engine worked correctly and retrieved 10 most relevant documents for each query. As seen in the screenshots, BM25 returned good results by analyzing term frequencies and document similarities. The system successfully combined Hadoop, Cassandra and Spark to process and search through the data efficiently.