Big Data Assignment 2 report.

Ruslan Izmailov, r.izmailov@innopolis.university, DS-01

Methodology

Changes made:

Docker compose file:
1. To be able to see logs I have exposed some ports on cluster-slave-node.
2. The same volume as for the master node was mounted to the slave-node.

Step 0. Starting services and setuping environment.

`app.sh` beginning

This step is done without any general changes from the provided template.

1. Start ssh service
2. Start hadoop and yarn services
3. Create environment, install dependences and pack

Step 1. Data preparation.

`prepare_data.sh`

To preprocess all data we have I have implemented the following steps:

1. Clears existing hdfs filesystem (to be able to write there without file collisions)
2. Runs `prepare_data.py` script
   - Samples N files (specified in script)
   - Creates documents in local file system
3. Writes created files to hdfs
4. Runs `prepare_data_indexing.py` script
   - Read files from `hdfs://data/`
   - Process each document (using RDD), so it can have the following structure:
     `document_id` `document_name` `document_body`
   - Save preprocessed documents to `hdfs://index/data`

Step 2. Creating an Index.

`index/index.sh`

1. Prepare directories for storing indexes: clear `hdfs://index/output`
2. Run mapreduce job:
   - Mapper:
     - Extracts words from each document (which length $\geqslant 3$)
     - Sends to stdout: `word \t document_id`
   - Reducer:
     - Extracts word & document_id from input
     - Stores data in dictionary with the following structure:
       word $\rightarrow$ (document_id, frequency)
     - After processing input sends to stdout: `word \t document_id \t frquency` that would be placed in the

3. Initialise cassandra schemas

Table: vocabulary_index		
Column	**Type**	**Key**
term	TEXT	K
doc_id	TEXT	C↑
tf	INT	

Table: document_info		
Column	**Type**	**Key**
doc_id	TEXT	K
title	TEXT	
doc_length	INT	

Table: collection_stats		
Column	**Type**	**Key**
key	TEXT	K
total_docs	INT	
avg_doc_length	DOUBLE	

Table: term_stats		
Column	**Type**	**Key**
term	TEXT	K
df	INT	

4. Store documents
   - Stores all documents, titles & documents sizes to the document_info table
5. Store to cassandra
   - Parses input generated by the mapreduce job and loading processed data to the vocabulary_index, term_stats
   - Writes a global statistics to the collection_stats.

2.1 Indexing file from not default path

Script is also working in a mode of adding & indexing files to the search engine. To do it you need to run a system without an entrypoint specified in the docker-compose.yaml file, but with basic start setup. Then you connect to the container, run the `start-services.sh` & `prepare_data.sh` scripts, basically it means that indexing is only available, when system started.

To your own file to the indexing you run `bash index/index.sh 123_my_own_file.txt` then system would prepare it, index and finally add to the cassandra tables. In case such file already indexed (*term* & *doc_id* present in cassandra) it would just ignore it.

Step 3. Ranking documents.

`search.sh`

Hyperparameters of the BM25:
k1 = 1.5
b = 0.75

Other notes:

- Spark cassandra adapter is used (due to the warnings I obtained)
- I was not able to manage to make script send its result to stdout using `--deploy-mode cluster` so `query.py` writes result to the `hdfs://query` and script `search.sh` aggregates them (just sends results of all jobs to from `hdfs://query` to stdout, as it is stores retrieval results ordered by default).

  In `--deploy-mode client` script just returns retrieval results in stdout without any other actions.

Steps to retrieve documents:

1. Retrieving main config

2. Loads the vocabulary index, term statistics, and document information from Cassandra into RDDs.
3. Filters the term statistics to include only the terms present in the query.
4. Computes the BM25 score for each document based on the tf & idf.
5. Ranks documents based on the scores, and cuts top 10 results.
6. Saves files to `hdfs://query` .

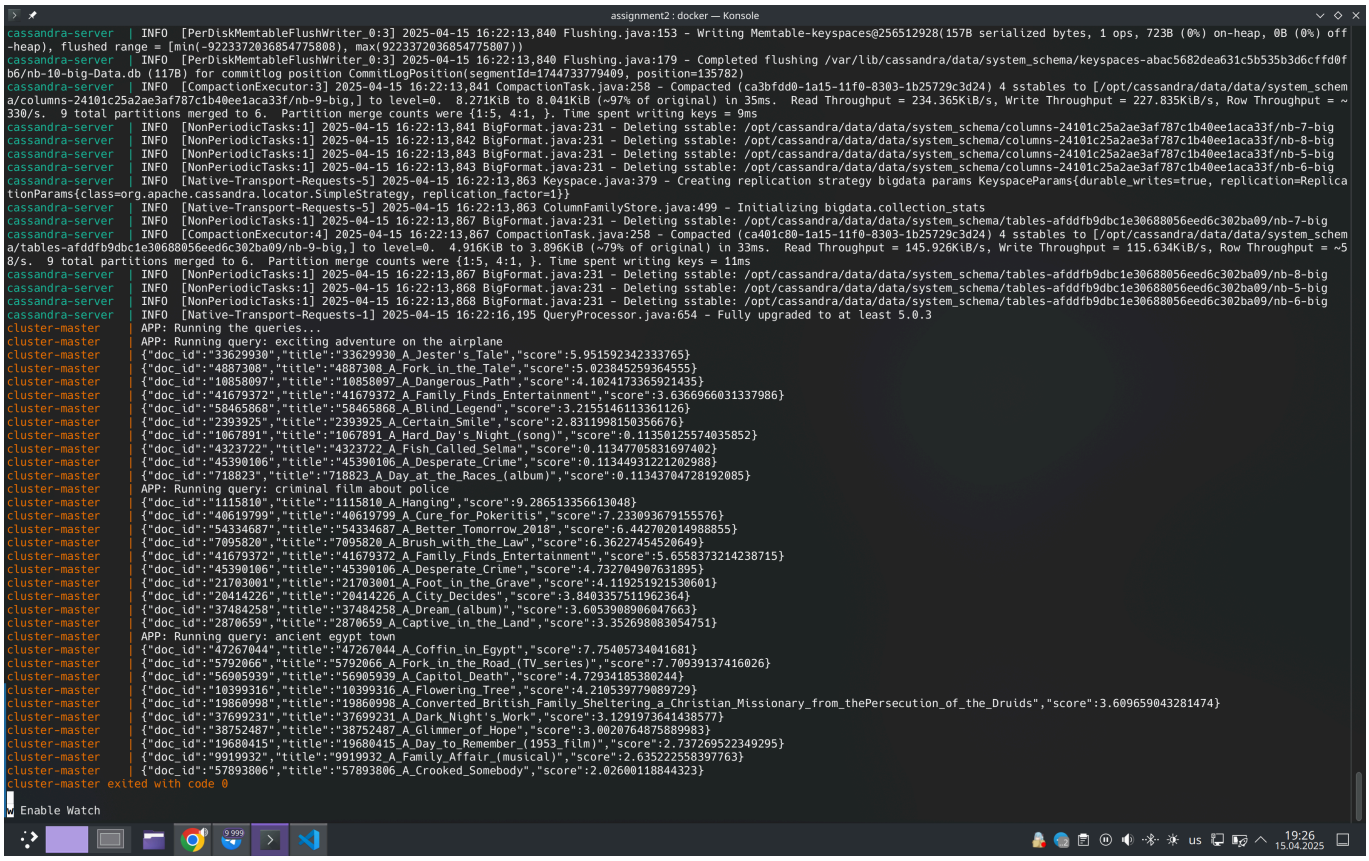
Demonstration

Guide how to run repository:

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

This will perform all needed actions to start, also it will run 3 predefined queries:

1. `exciting adventure on the airplane`
2. `criminal film about police`
3. `ancient egypt town`



I have explored top 1-3 documents for each query and they all contained all/most of words from the initial query. Also the score is highly grows, when target word (word from query) present several times in the documents.