

Distributed Search Engine Report

1. Methodology

1.1. System Overview

The system is composed of the following components:

- **Data Preparation (PySpark):** Extracts documents from a Parquet file and stores them in HDFS.
- **Indexing (Hadoop MapReduce):** Builds an inverted index and stores it in Apache Cassandra. A single custom mapper and reducer were implemented to generate and aggregate posting lists.
- **Query Execution (PySpark):** Uses BM25 algorithm with Spark RDD to rank results using index data from Cassandra.
- **Orchestration:** All steps are coordinated via shell scripts and Docker Compose.

1.2. Design Choices

- **MapReduce for Indexing:** A single-stage MapReduce job was written using one custom mapper and one reducer to generate posting lists and compute document frequencies.
- **Cassandra for Storage:** Chosen for its horizontal scalability and fast writes.
- **Spark RDD for Ranking:** Efficient for computing BM25 on distributed datasets.
- **Docker for Environment Management:** Ensures reproducibility and isolation.
- **Automation:** Entire pipeline is run with a single command.

2. Demonstration

2.1. Running the Search Engine

To run the full pipeline:

```
./app.sh
```

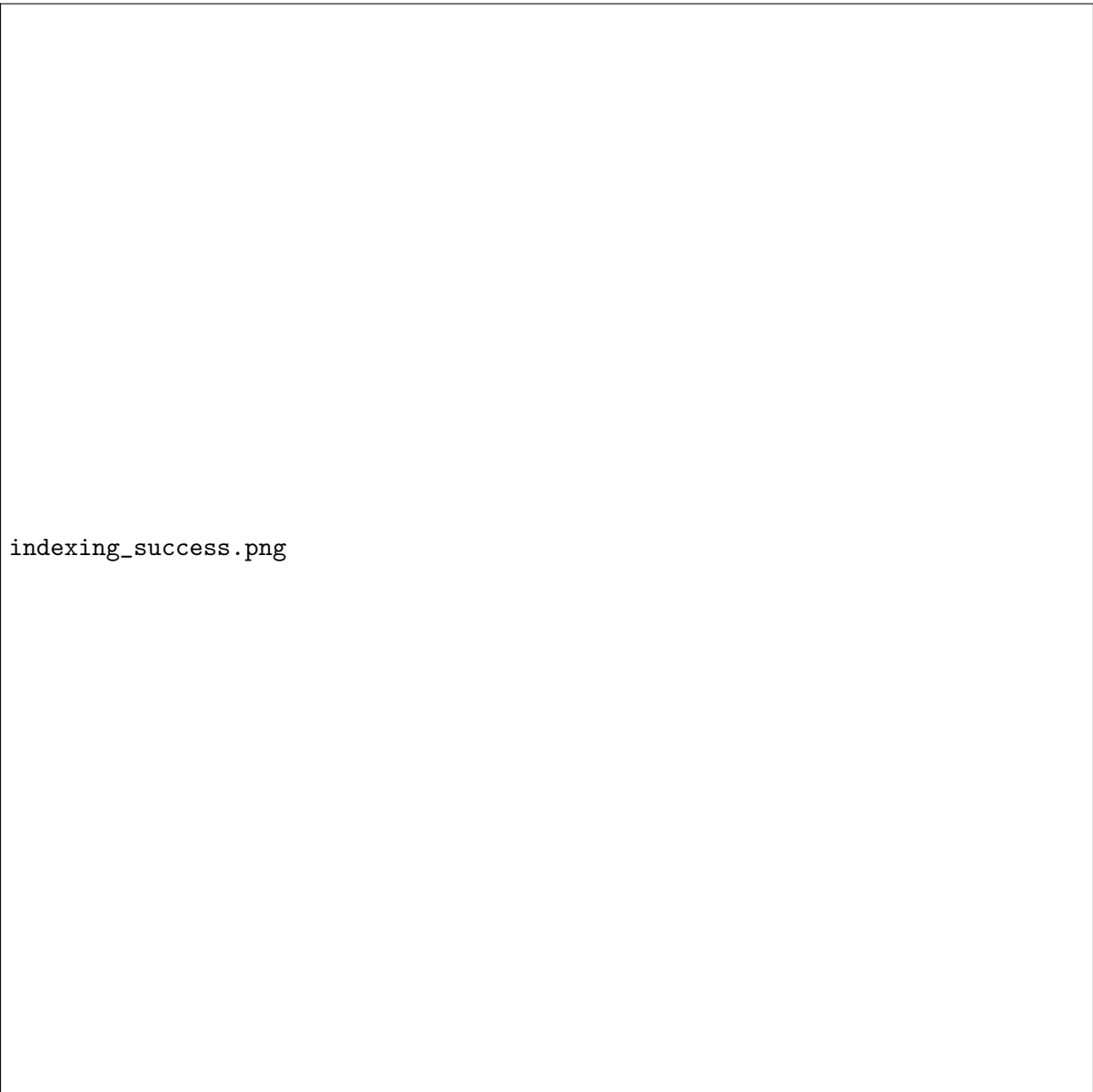
This script performs the following:

1. Starts services via Docker Compose.
2. Prepares data and uploads to HDFS.
3. Runs MapReduce indexing job.
4. Inserts index into Cassandra.

5. Executes BM25-based query using Spark.

2.2. Indexing 100 Documents

Screenshot: Successful indexing of 100 documents



indexing_success.png

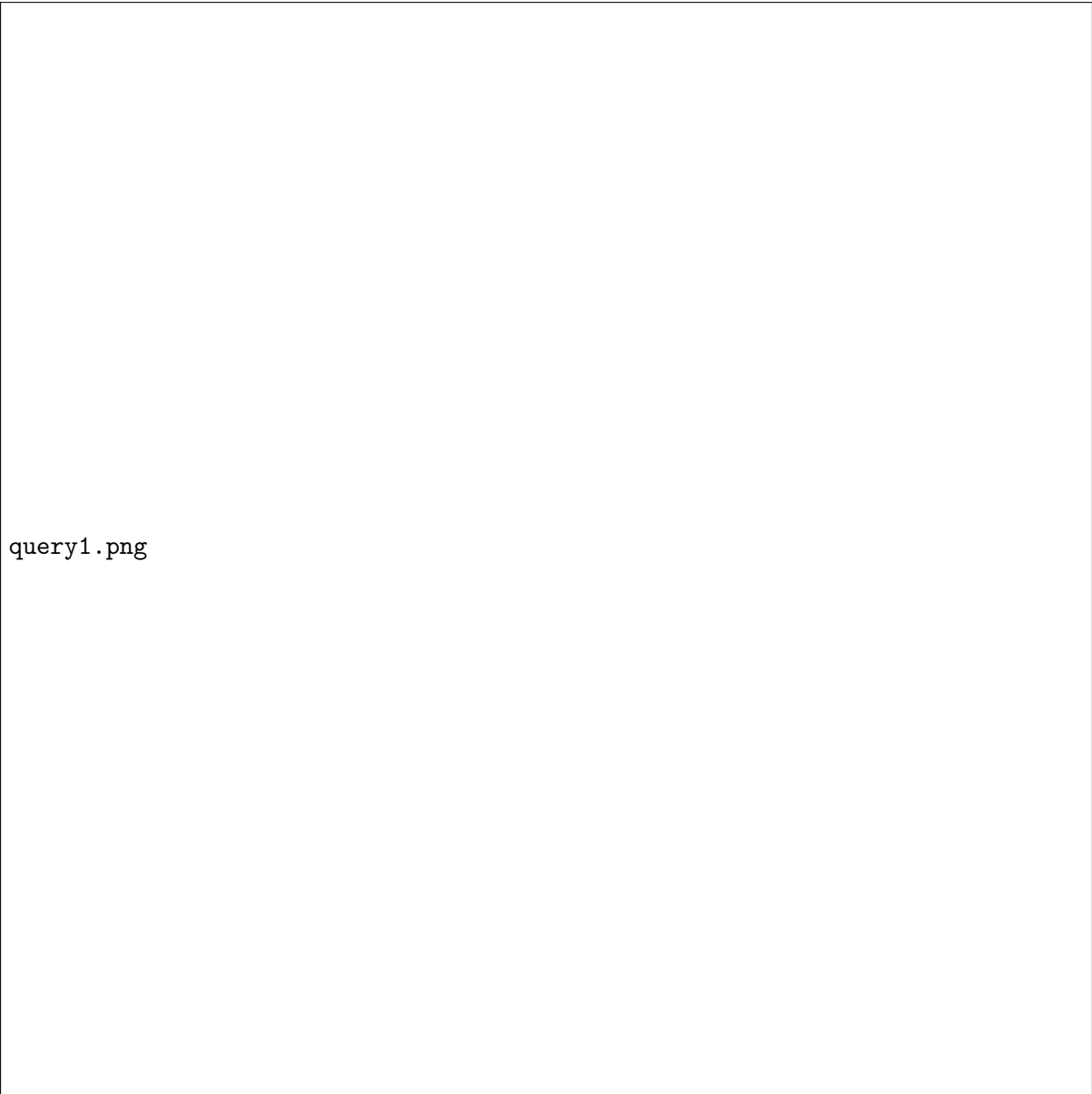
Figure 1: Successful indexing using MapReduce and storage in Cassandra.

The MapReduce job completed successfully, and the index is verified by querying the Cassandra database. Each term is associated with a list of document IDs and term frequencies.

2.3. Query Results

2.3.1 Query: "spark processing"

Screenshot: Result for query “spark processing”



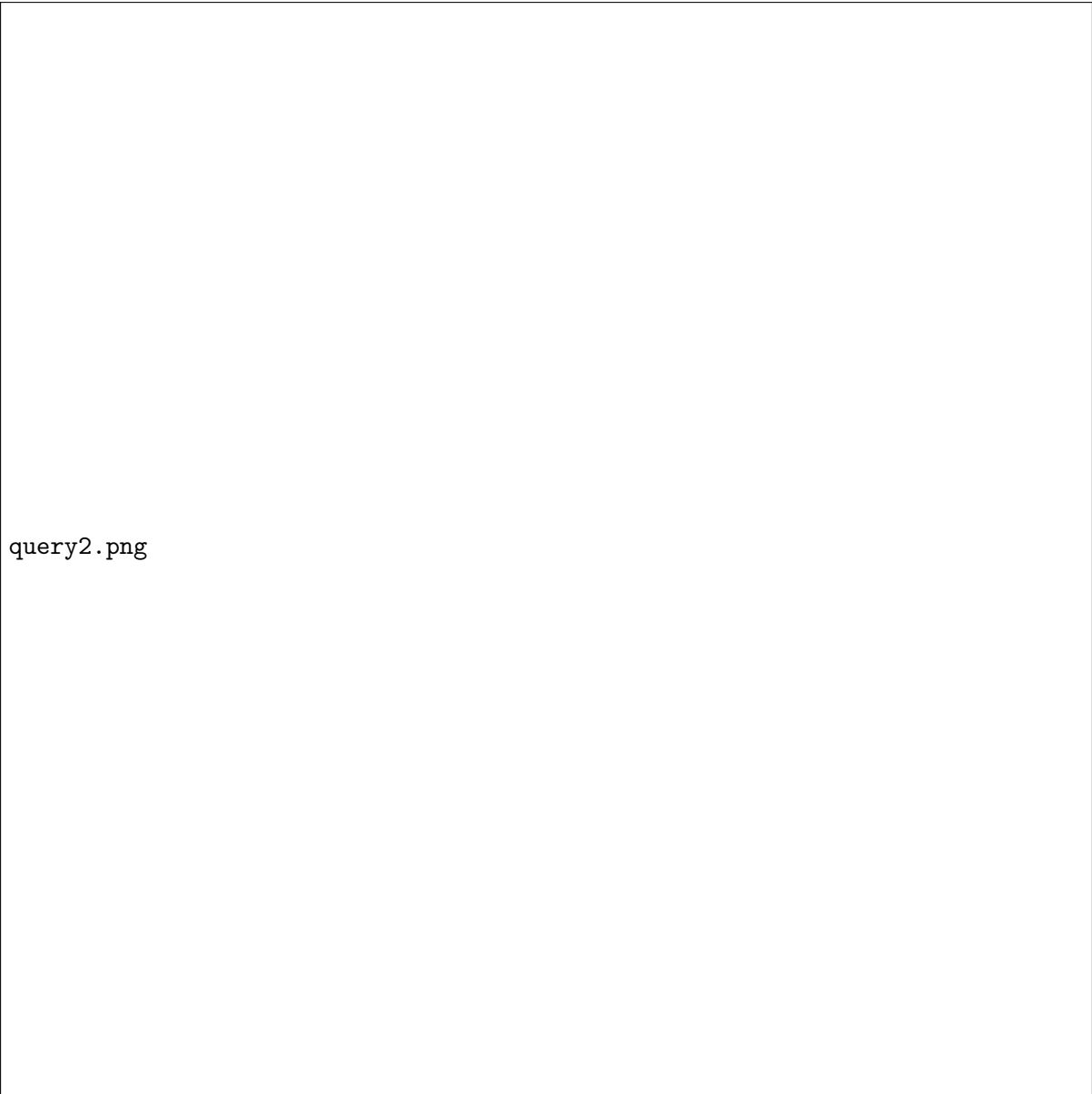
query1.png

Figure 2: Query results ranked by BM25 scores.

Explanation: The top document had high relevance to both "spark" and "processing", with term frequencies and document length contributing positively to the BM25 score.

2.3.2 Query: "big data"

Screenshot: Result for query "big data"



query2.png

Figure 3: Second query results showing documents with "big data".

Explanation: Documents with both "big" and "data" were ranked higher. IDF values impacted the score, favoring documents where these terms were less common.

2.4. Reflections

- **Effectiveness:** The BM25 implementation provides relevant, ranked search results.
- **Scalability:** The use of Spark and MapReduce enables processing at scale.
- **Limitations:**
 - Simple tokenizer, no stemming or stopword removal.
 - BM25 assumes independence of terms.
 - Cassandra read latency can be high if not tuned.

3. Conclusion

We implemented a distributed search engine using Hadoop, Spark, and Cassandra. The pipeline covers full-text document indexing and BM25-based ranking. With Docker automation, the system is easy to deploy and run. Future improvements can include:

- Adding stopword removal and stemming.
- Optimizing Spark joins with Cassandra.
- Using Spark Structured Streaming for real-time updates.