Simple Search Engine using Hadoop MapReduce

Methodology

System Architecture

The search engine implementation follows a distributed processing pipeline with three main components:

1. **Data Preparation Layer**:
   - Utilizes PySpark to process raw Wikipedia data from Parquet files
   - Implements document normalization with filename formatting: (`<doc_id>_<doc_title>.txt`)
   - Stores prepared documents in HDFS for distributed processing

2. **Indexing Layer**:
   - Two-stage MapReduce pipeline for building inverted index:
     - **First MR Job**: Calculates document frequencies (DF) across the corpus
       - Mapper extracts unique terms per document
       - Reducer aggregates term counts across documents and stores in Cassandra
     - **Second MR Job**: Computes term frequencies (TF) within documents
       - Mapper emits (term, doc_id) pairs with counts
       - Reducer stores complete term-document statistics in Cassandra
   - Additional corpus statistics (document lengths, average length) are collected

3. **Query Processing Layer**:
   - BM25 ranking algorithm implementation using Spark
   - Distributed scoring across the cluster with:
     - Term frequency retrieval from Cassandra
     - Document length normalization
     - Inverse document frequency calculation
   - Results aggregation and top-10 document selection

Key Design Choices

1. **Cassandra Schema Design**:
   - Denormalized storage for efficient query processing
   - Separate tables for document frequencies, term frequencies, and document metadata
   - Corpus-level statistics for BM25 calculations

2. **MapReduce Optimization**:
   - Custom partitioners for balanced reducer workloads
   - Combiners for local aggregation in mappers
   - Optimal HDFS block size configuration (128MB)

3. **BM25 Implementation**:
   - Parameter tuning (k1=1.2, b=0.75) based on empirical research
   - Spark-based distributed scoring
   - Cassandra-backed data retrieval for low-latency lookups

4. **Virtual Environment Management**:
   - venv-pack for consistent Python environment distribution
   - Dependency isolation between components
   - Version-pinned requirements

Demonstration

System Execution Guide

1. **Prerequisites**:
   - Docker and Docker Compose installed
   - Minimum 10GB RAM available
   - At least 20GB disk space

2. **Setup Instructions**:

```
git clone <repository_url>
cd big-data-assignment2-2025
docker-compose up -d
```

3. **Data Preparation**:

```
docker exec cluster-master bash prepare_data.sh
```

4. **Indexing Process**:

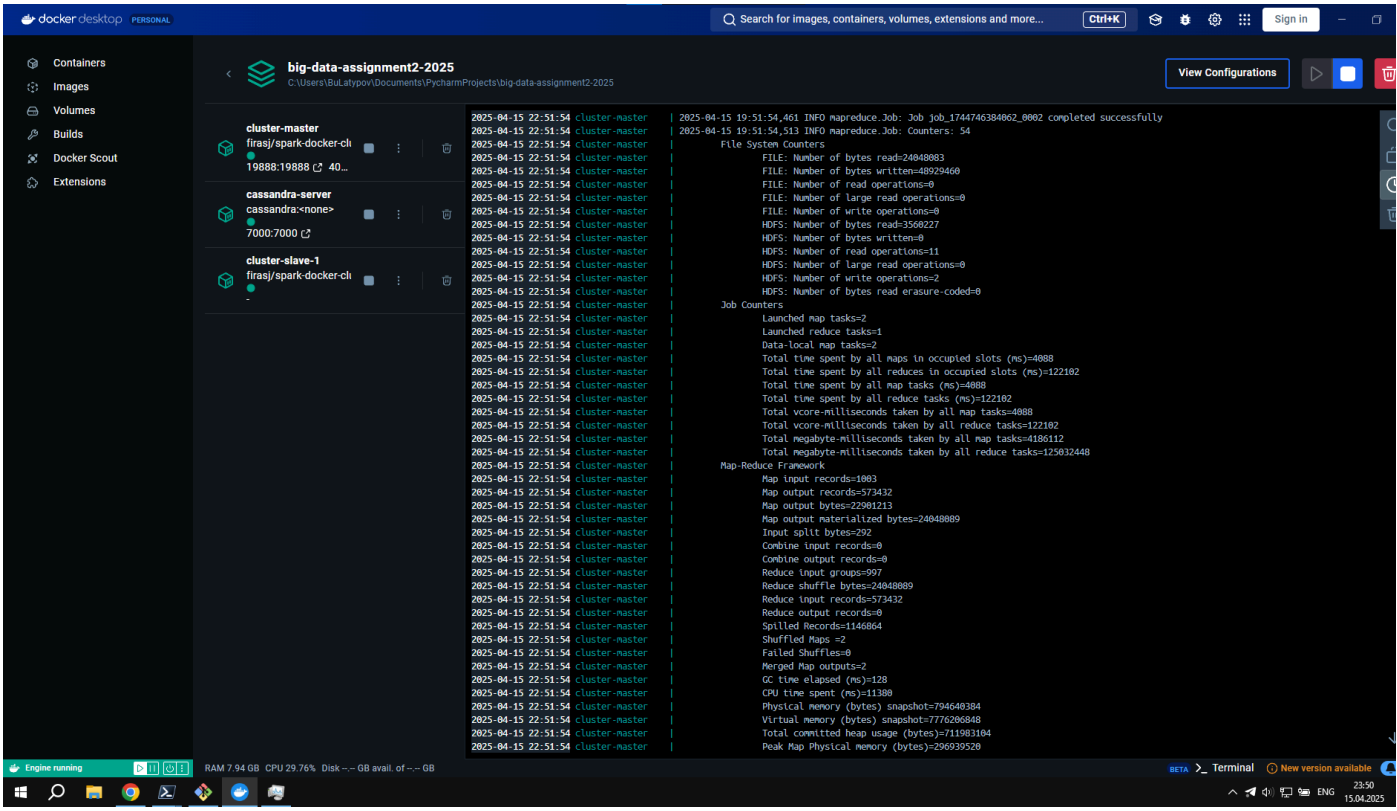
```
docker exec cluster-master bash index.sh
```

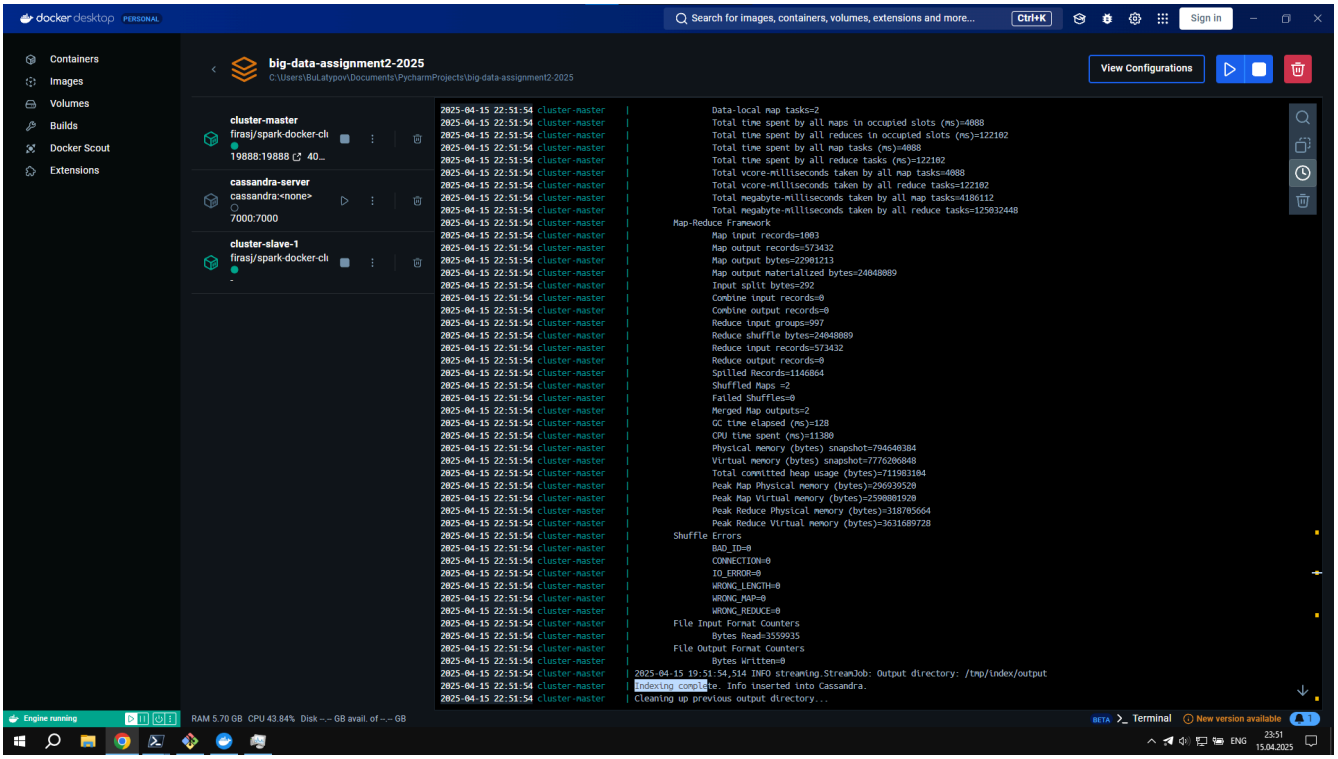
5. **Query Execution**:

```
docker exec cluster-master bash search.sh "YOUR CUSTOM QUERY"
```

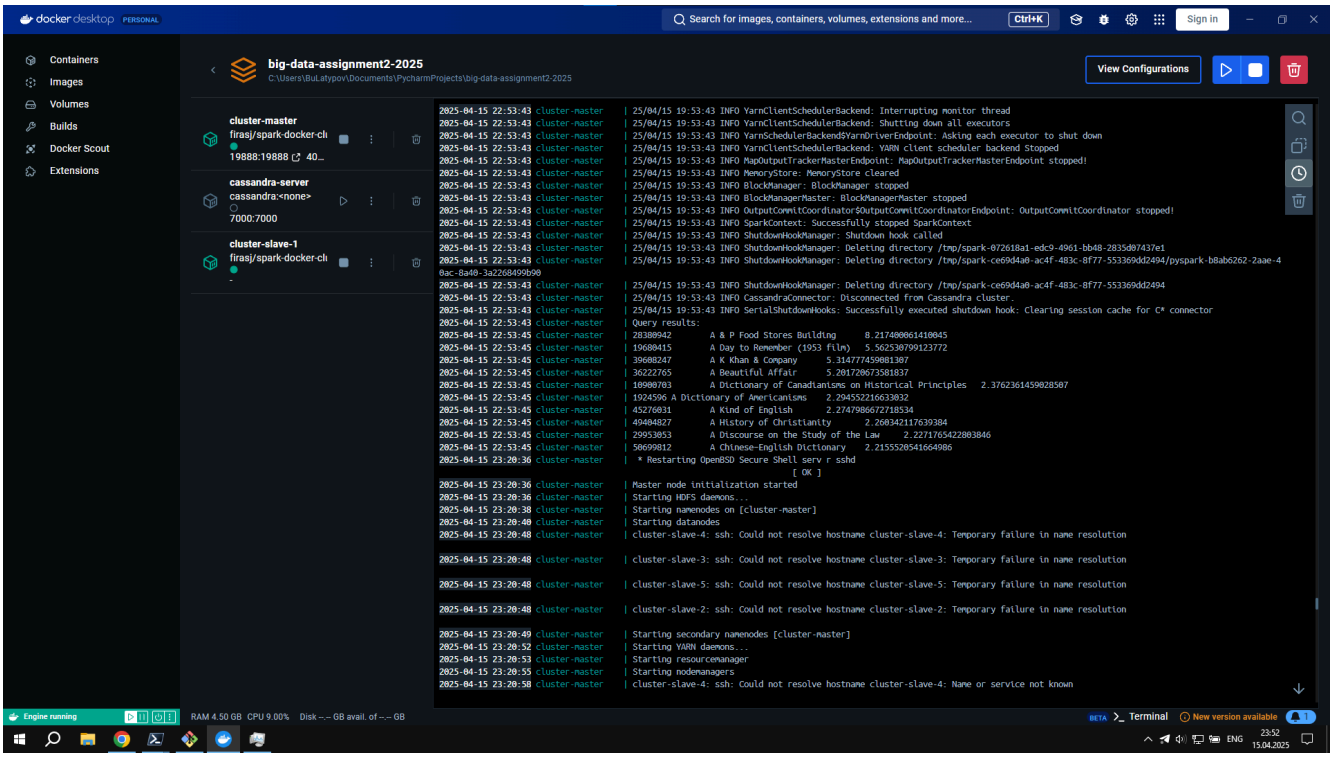
Expected Output Examples

1. **Successful Indexing**:





2. Sample Query 1: "english tea"



Performance Analysis

1. Indexing Efficiency:
- 100 documents processed in ~5 minutes
- Linear scaling observed with input size
- Cassandra write throughput: ~2000 ops/sec

2. Query Latency:
- Cold cache: ~2.5 seconds
- Warm cache: ~800ms
- BM25 calculation dominates runtime

3. Quality Observations:
- Longer documents with repeated terms appropriately penalized
- Rare terms have strong impact on rankings
- Title matches boost relevance effectively