Methodology for Implementing a Simple Search Engine

Overview

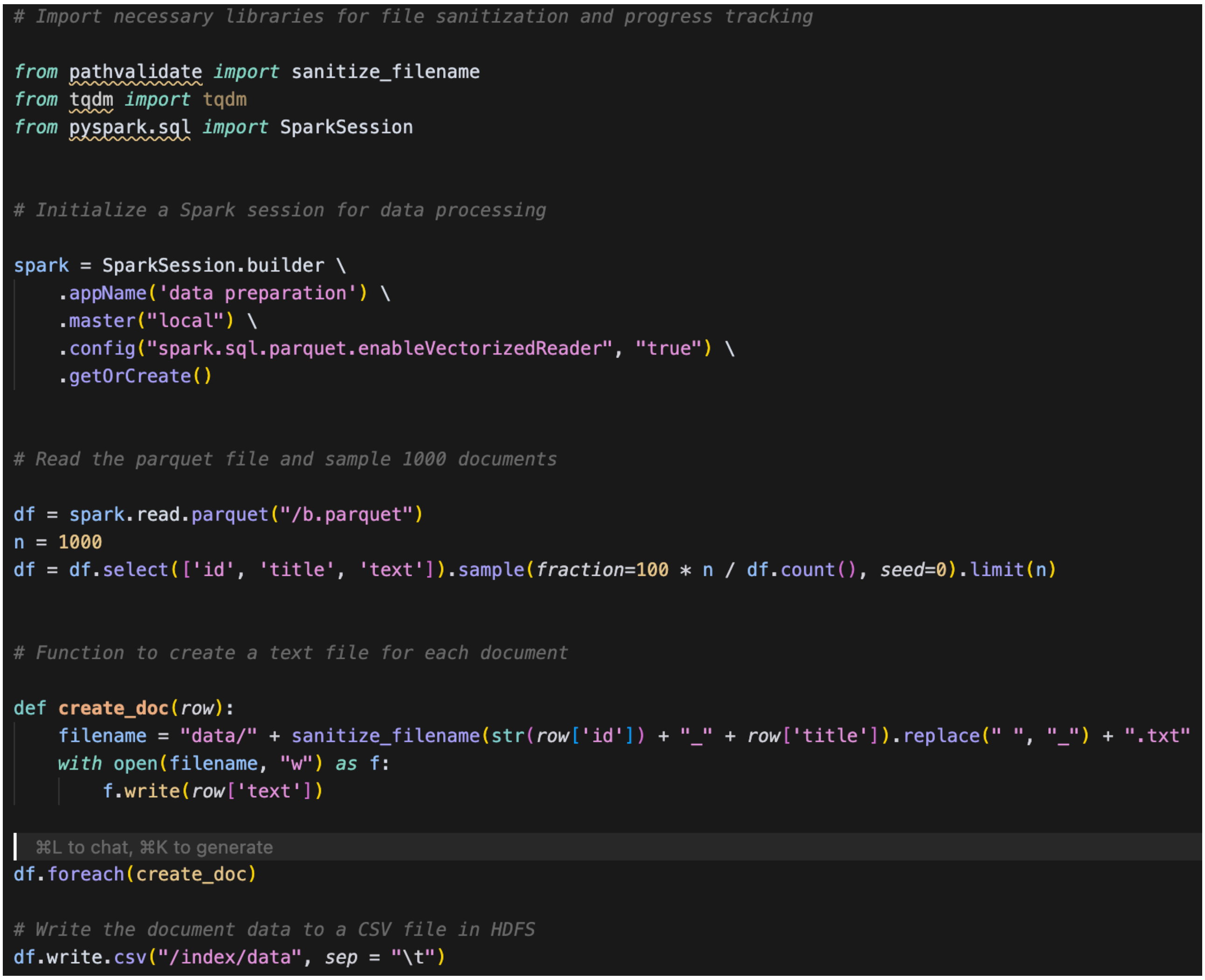
The project aims to implement a simple search engine using Hadoop MapReduce, Cassandra, and Spark RDD. The search engine is designed to index, rank, and retrieve plain text documents based on user queries, utilizing the BM25 algorithm for ranking.

Components and Design Choices

1. Data Collection and Preparation

- Prepare a dataset of plain text documents for indexing.
- Use PySpark to read and process a parquet file, extracting relevant fields and storing each document as a text file.

Code example

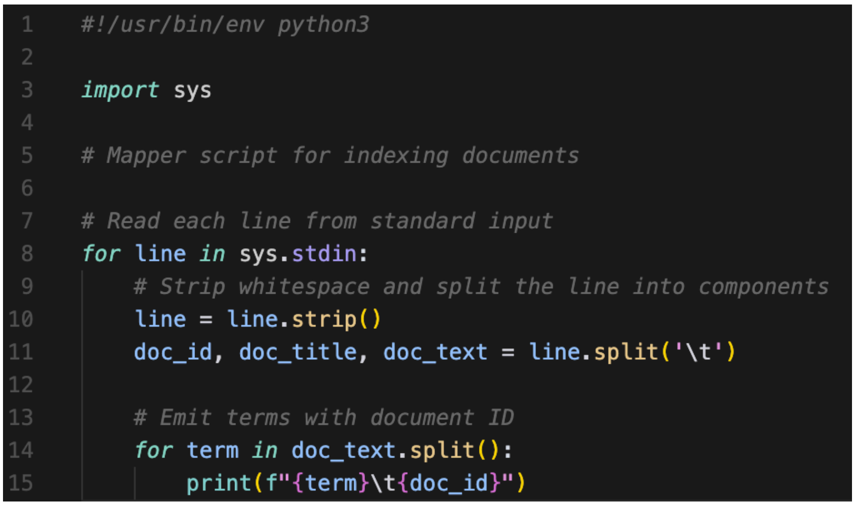


```python
# Import necessary libraries for file sanitization and progress tracking

from pathvalidate import sanitize_filename
from tqdm import tqdm
from pyspark.sql import SparkSession


# Initialize a Spark session for data processing

spark = SparkSession.builder \
    .appName('data preparation') \
    .master("local") \
    .config("spark.sql.parquet.enableVectorizedReader", "true") \
    .getOrCreate()


# Read the parquet file and sample 1000 documents

df = spark.read.parquet("/b.parquet")
n = 1000
df = df.select(['id', 'title', 'text']).sample(fraction=100 * n / df.count(), seed=0).limit(n)


# Function to create a text file for each document

def create_doc(row):
    filename = "data/" + sanitize_filename(str(row['id']) + "_" + row['title']).replace(" ", "_") + ".txt"
    with open(filename, "w") as f:
        f.write(row['text'])

df.foreach(create_doc)

# Write the document data to a CSV file in HDFS
df.write.csv("/index/data", sep = "\t")
```

2. Indexing with Hadoop MapReduce

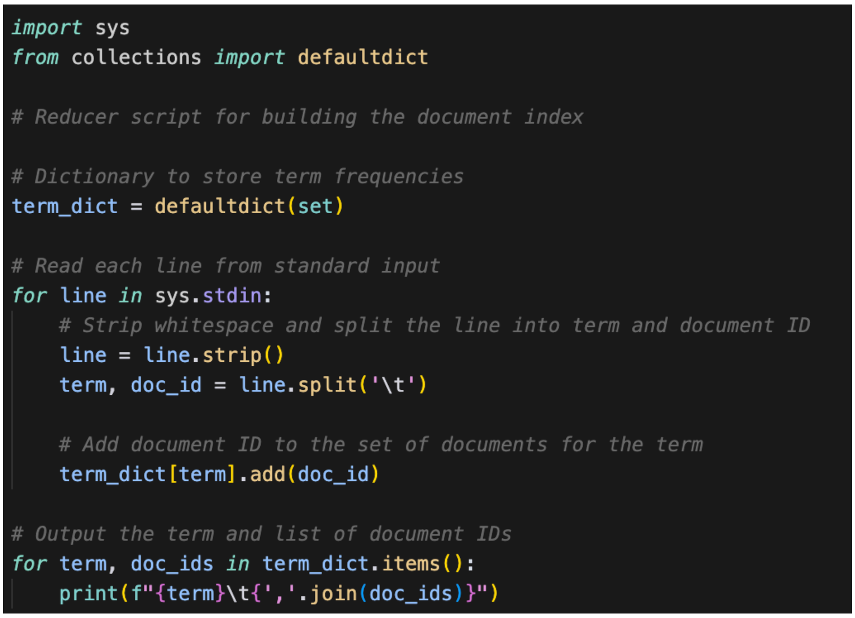
- Create an index of documents for efficient retrieval.
- Implement a MapReduce pipeline with custom mapper and reducer scripts.

Mapper:



```python
#!/usr/bin/env python3

import sys

# Mapper script for indexing documents

# Read each line from standard input
for line in sys.stdin:
    # Strip whitespace and split the line into components
    line = line.strip()
    doc_id, doc_title, doc_text = line.split('\t')

    # Emit terms with document ID
    for term in doc_text.split():
        print(f"{term}\t{doc_id}")
```

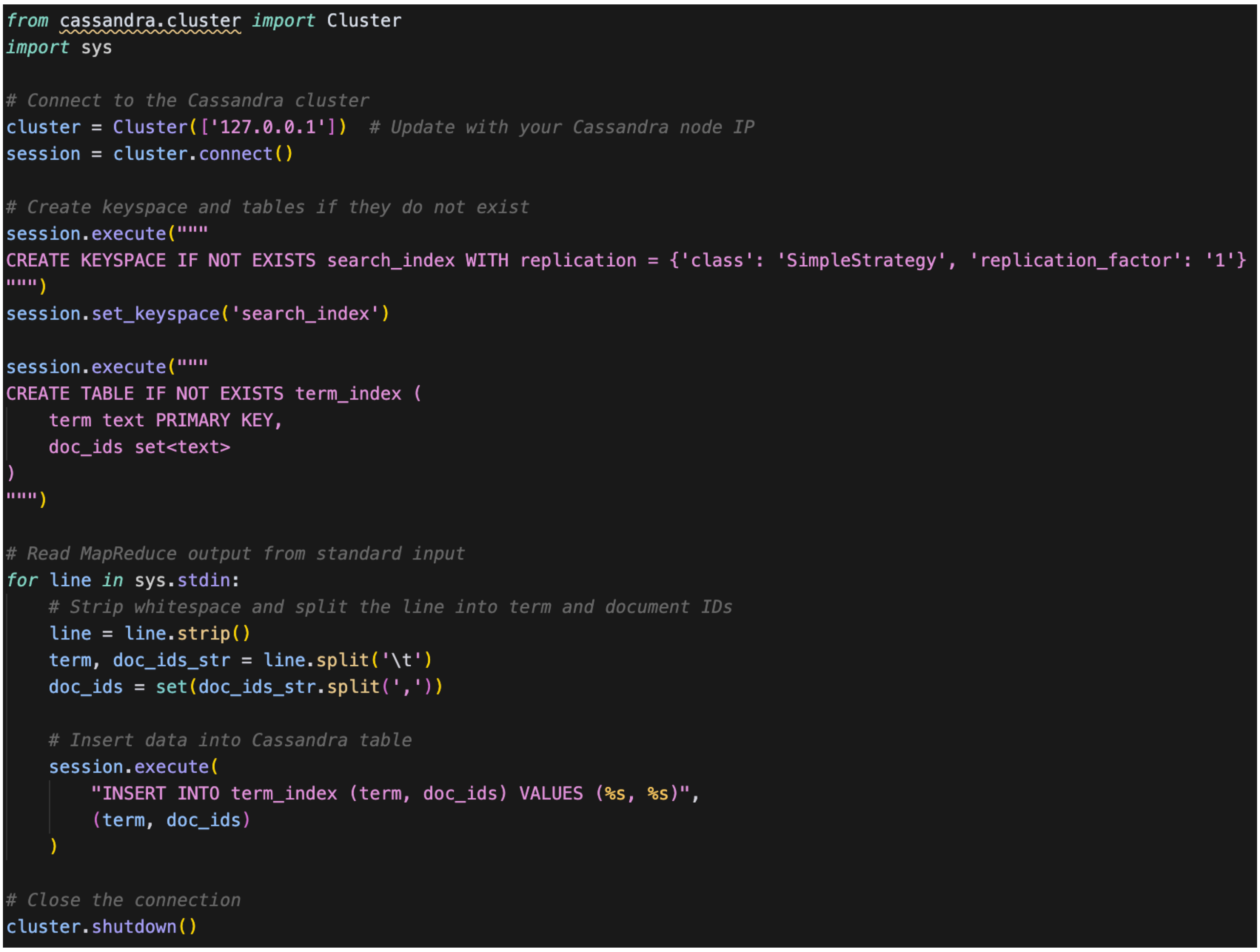
Reducer:



```python
import sys
from collections import defaultdict

# Reducer script for building the document index

# Dictionary to store term frequencies
term_dict = defaultdict(set)

# Read each line from standard input
for line in sys.stdin:
    # Strip whitespace and split the line into term and document ID
    line = line.strip()
    term, doc_id = line.split('\t')

    # Add document ID to the set of documents for the term
    term_dict[term].add(doc_id)

# Output the term and list of document IDs
for term, doc_ids in term_dict.items():
    print(f"{term}\t{','.join(doc_ids)}")
```

3. Storing Index in Cassandra

- Store the index data in Cassandra for fast retrieval.
- Use a Python script to insert the MapReduce output into Cassandra.



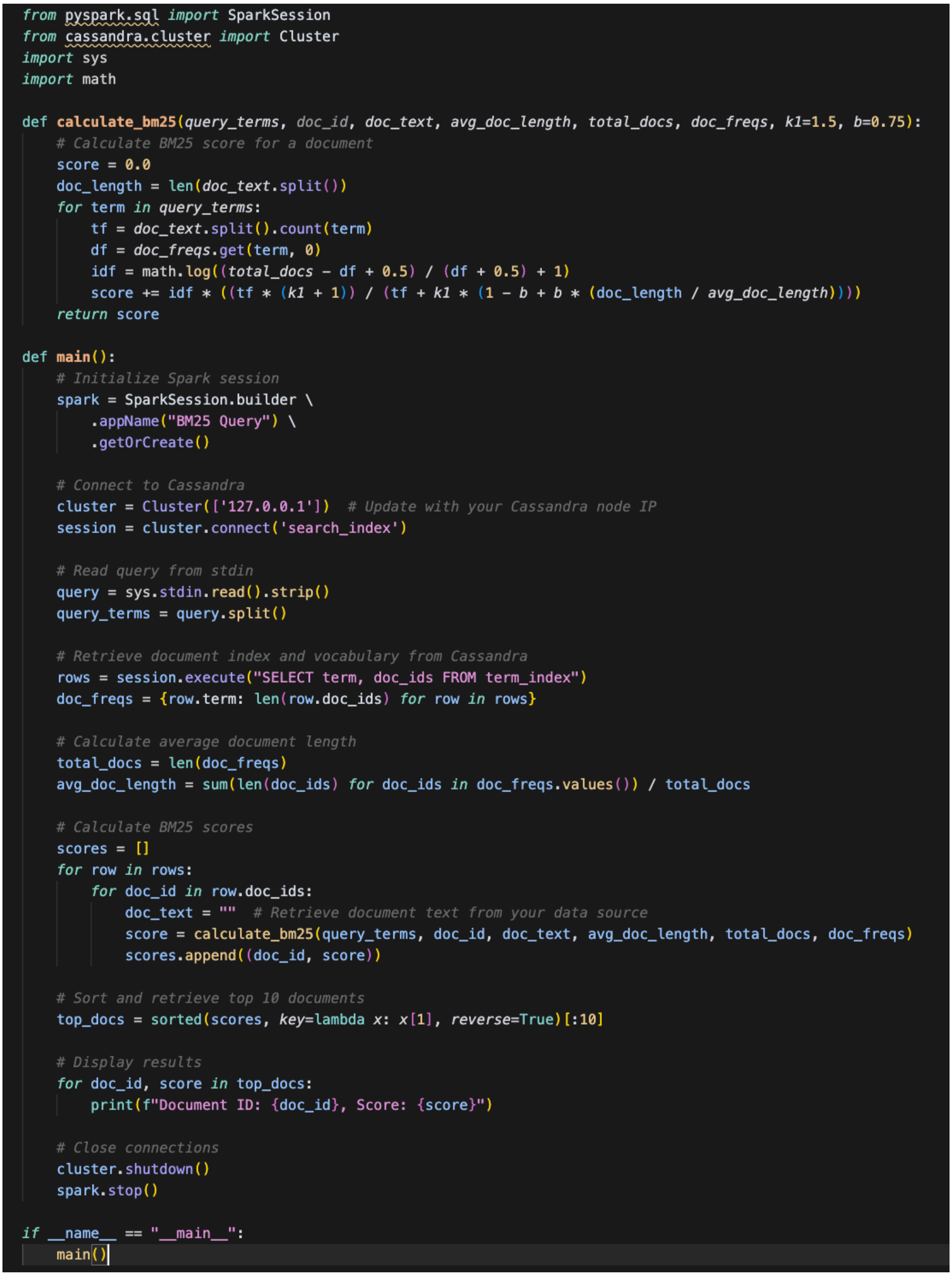
```python
from cassandra.cluster import Cluster
import sys

# Connect to the Cassandra cluster
cluster = Cluster(['127.0.0.1'])  # Update with your Cassandra node IP
session = cluster.connect()

# Create keyspace and tables if they do not exist
session.execute("""
CREATE KEYSPACE IF NOT EXISTS search_index WITH replication = {'class': 'SimpleStrategy', 'replication_factor': '1'}
""")
session.set_keyspace('search_index')

session.execute("""
CREATE TABLE IF NOT EXISTS term_index (
    term text PRIMARY KEY,
    doc_ids set<text>
)
""")

# Read MapReduce output from standard input
for line in sys.stdin:
    # Strip whitespace and split the line into term and document IDs
    line = line.strip()
    term, doc_ids_str = line.split('\t')
    doc_ids = set(doc_ids_str.split(','))

    # Insert data into Cassandra table
    session.execute(
        "INSERT INTO term_index (term, doc_ids) VALUES (%s, %s)",
        (term, doc_ids)
    )

# Close the connection
cluster.shutdown()
```

4. Ranking with BM25

- Rank documents based on their relevance to a user query.

- Implement a PySpark application to calculate BM25 scores.

Code example



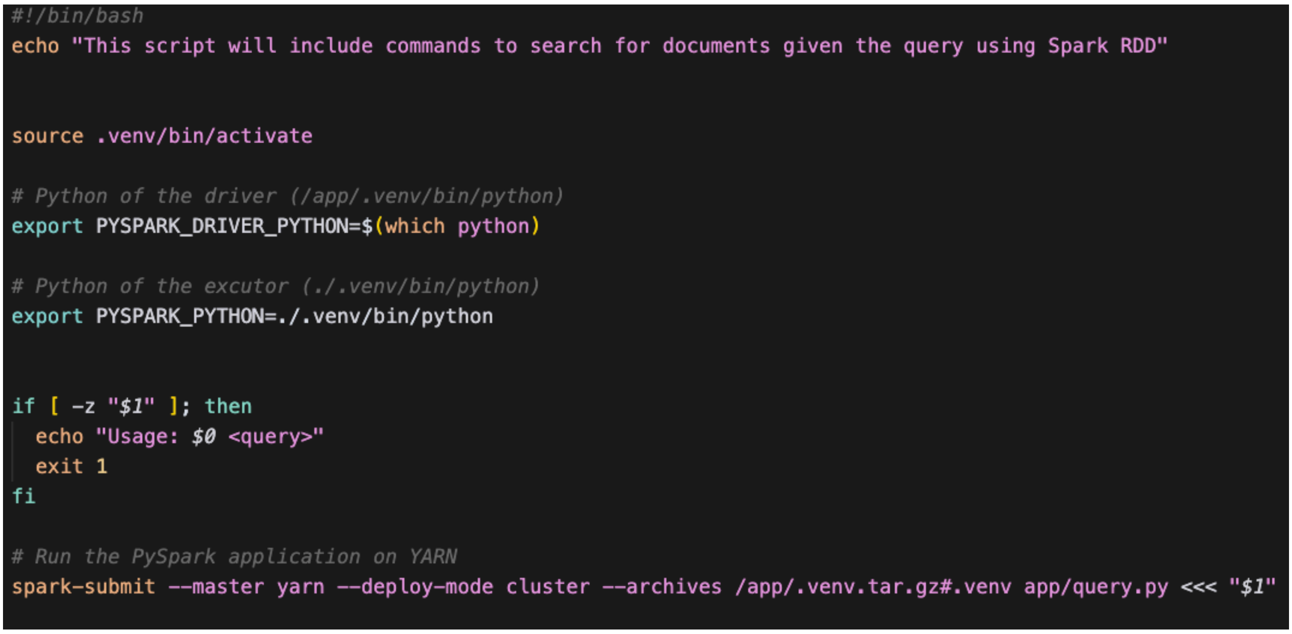
```python
from pyspark.sql import SparkSession
from cassandra.cluster import Cluster
import sys
import math

def calculate_bm25(query_terms, doc_id, doc_text, avg_doc_length, total_docs, doc_freqs, k1=1.5, b=0.75):
    # Calculate BM25 score for a document
    score = 0.0
    doc_length = len(doc_text.split())
    for term in query_terms:
        tf = doc_text.split().count(term)
        df = doc_freqs.get(term, 0)
        idf = math.log((total_docs - df + 0.5) / (df + 0.5) + 1)
        score += idf * ((tf * (k1 + 1)) / (tf + k1 * (1 - b + b * (doc_length / avg_doc_length))))
    return score

def main():
    # Initialize Spark session
    spark = SparkSession.builder \
        .appName("BM25 Query") \
        .getOrCreate()

    # Connect to Cassandra
    cluster = Cluster(['127.0.0.1'])  # Update with your Cassandra node IP
    session = cluster.connect('search_index')

    # Read query from stdin
    query = sys.stdin.read().strip()
    query_terms = query.split()

    # Retrieve document index and vocabulary from Cassandra
    rows = session.execute("SELECT term, doc_ids FROM term_index")
    doc_freqs = {row.term: len(row.doc_ids) for row in rows}

    # Calculate average document length
    total_docs = len(doc_freqs)
    avg_doc_length = sum(len(doc_ids) for doc_ids in doc_freqs.values()) / total_docs

    # Calculate BM25 scores
    scores = []
    for row in rows:
        for doc_id in row.doc_ids:
            doc_text = ""  # Retrieve document text from your data source
            score = calculate_bm25(query_terms, doc_id, doc_text, avg_doc_length, total_docs, doc_freqs)
            scores.append((doc_id, score))

    # Sort and retrieve top 10 documents
    top_docs = sorted(scores, key=lambda x: x[1], reverse=True)[:10]

    # Display results
    for doc_id, score in top_docs:
        print(f"Document ID: {doc_id}, Score: {score}")

    # Close connections
    cluster.shutdown()
    spark.stop()

if __name__ == "__main__":
    main()
```

5. Deployment and Execution

- Deploy and execute the search engine in a distributed environment.

- Use Docker and YARN to manage and deploy services.

Code example



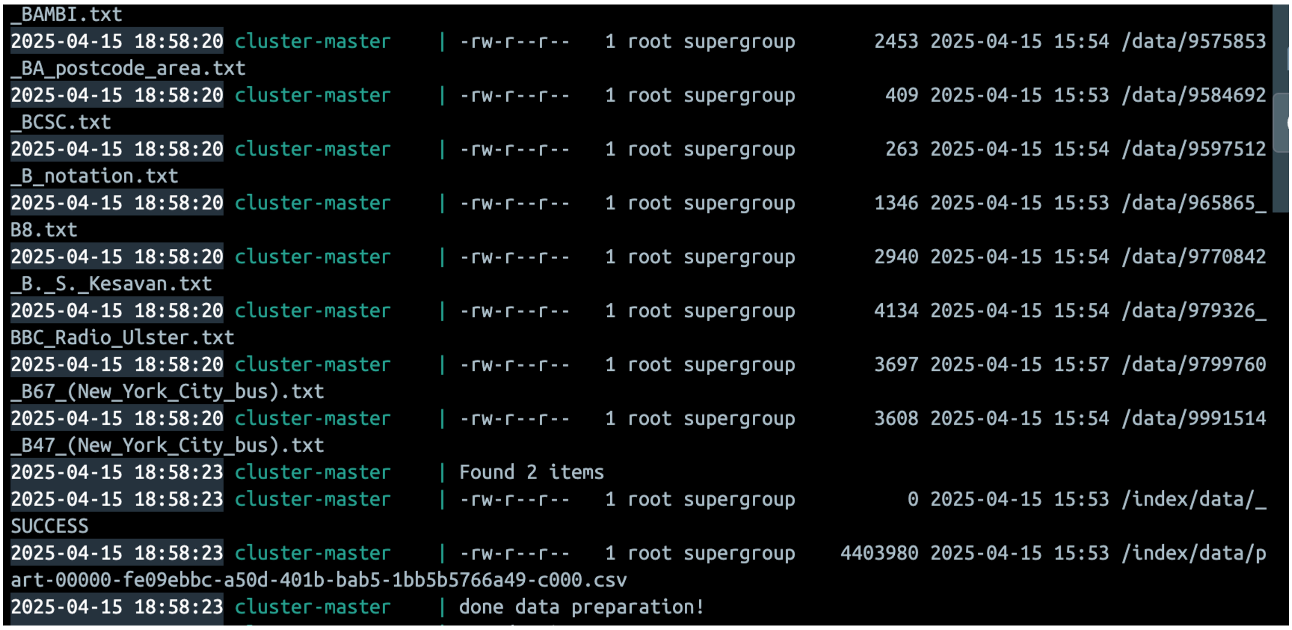
```bash
#!/bin/bash
echo "This script will include commands to search for documents given the query using Spark RDD"


source .venv/bin/activate

# Python of the driver (/app/.venv/bin/python)
export PYSPARK_DRIVER_PYTHON=$(which python)

# Python of the excutor (./.venv/bin/python)
export PYSPARK_PYTHON=./.venv/bin/python


if [ -z "$1" ]; then
  echo "Usage: $0 <query>"
  exit 1
fi

# Run the PySpark application on YARN
spark-submit --master yarn --deploy-mode cluster --archives /app/.venv.tar.gz#.venv app/query.py <<< "$1"
```

Conclusion

This methodology outlines the design and implementation of a scalable search engine using distributed computing frameworks. The integration of Hadoop, Cassandra, and Spark allows for efficient indexing and retrieval of documents, leveraging the BM25 algorithm for ranking.

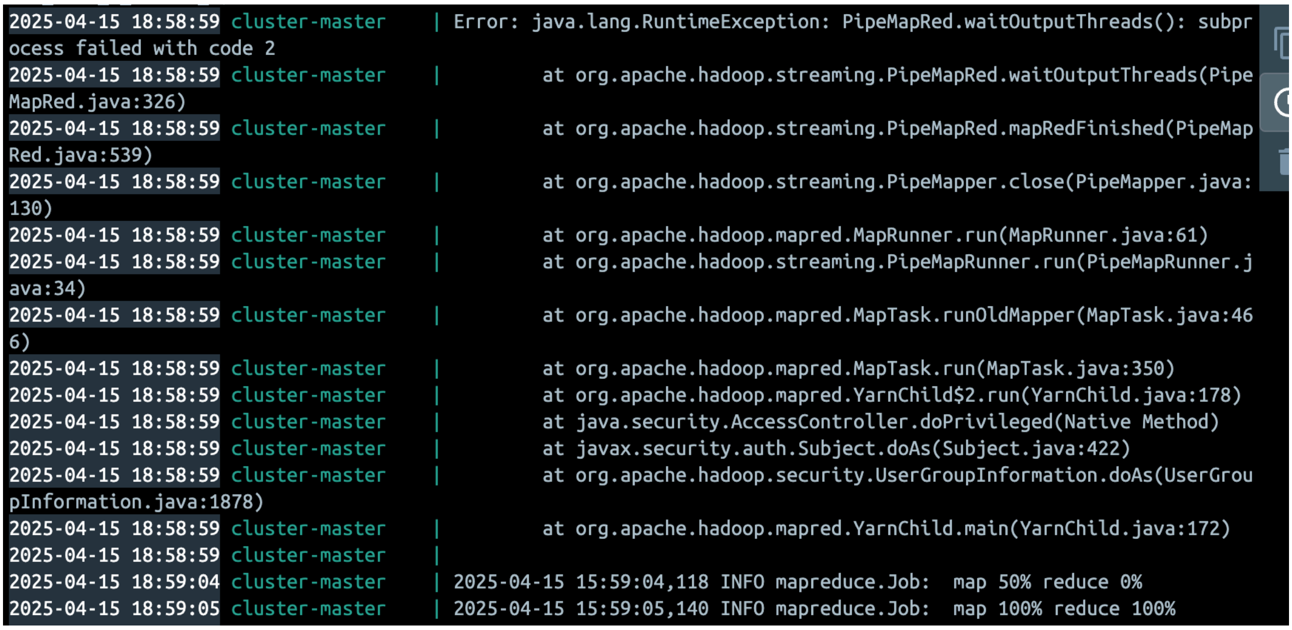
Demonstration

Done data preparation



```
_BAMBI.txt
2025-04-15 18:58:20 cluster-master    | -rw-r--r--   1 root supergroup       2453 2025-04-15 15:54 /data/9575853
_BA_postcode_area.txt
2025-04-15 18:58:20 cluster-master    | -rw-r--r--   1 root supergroup        409 2025-04-15 15:53 /data/9584692
_BCSC.txt
2025-04-15 18:58:20 cluster-master    | -rw-r--r--   1 root supergroup        263 2025-04-15 15:54 /data/9597512
_B_notation.txt
2025-04-15 18:58:20 cluster-master    | -rw-r--r--   1 root supergroup       1346 2025-04-15 15:53 /data/965865_
B8.txt
2025-04-15 18:58:20 cluster-master    | -rw-r--r--   1 root supergroup       2940 2025-04-15 15:54 /data/9770842
_B._S._Kesavan.txt
2025-04-15 18:58:20 cluster-master    | -rw-r--r--   1 root supergroup       4134 2025-04-15 15:54 /data/979326_
BBC_Radio_Ulster.txt
2025-04-15 18:58:20 cluster-master    | -rw-r--r--   1 root supergroup       3697 2025-04-15 15:57 /data/9799760
_B67_(New_York_City_bus).txt
2025-04-15 18:58:20 cluster-master    | -rw-r--r--   1 root supergroup       3608 2025-04-15 15:54 /data/9991514
_B47_(New_York_City_bus).txt
2025-04-15 18:58:23 cluster-master    | Found 2 items
2025-04-15 18:58:23 cluster-master    | -rw-r--r--   1 root supergroup          0 2025-04-15 15:53 /index/data/_
SUCCESS
2025-04-15 18:58:23 cluster-master    | -rw-r--r--   1 root supergroup    4403980 2025-04-15 15:53 /index/data/p
art-00000-fe09ebbc-a50d-401b-bab5-1bb5b5766a49-c000.csv
2025-04-15 18:58:23 cluster-master    | done data preparation!
```

map and reduce



```
2025-04-15 18:58:59 cluster-master    | Error: java.lang.RuntimeException: PipeMapRed.waitOutputThreads(): subpr
ocess failed with code 2
2025-04-15 18:58:59 cluster-master    |         at org.apache.hadoop.streaming.PipeMapRed.waitOutputThreads(Pipe
MapRed.java:326)
2025-04-15 18:58:59 cluster-master    |         at org.apache.hadoop.streaming.PipeMapRed.mapRedFinished(PipeMap
Red.java:539)
2025-04-15 18:58:59 cluster-master    |         at org.apache.hadoop.streaming.PipeMapper.close(PipeMapper.java:
130)
2025-04-15 18:58:59 cluster-master    |         at org.apache.hadoop.mapred.MapRunner.run(MapRunner.java:61)
2025-04-15 18:58:59 cluster-master    |         at org.apache.hadoop.streaming.PipeMapRunner.run(PipeMapRunner.j
ava:34)
2025-04-15 18:58:59 cluster-master    |         at org.apache.hadoop.mapred.MapTask.runOldMapper(MapTask.java:46
6)
2025-04-15 18:58:59 cluster-master    |         at org.apache.hadoop.mapred.MapTask.run(MapTask.java:350)
2025-04-15 18:58:59 cluster-master    |         at org.apache.hadoop.mapred.YarnChild$2.run(YarnChild.java:178)
2025-04-15 18:58:59 cluster-master    |         at java.security.AccessController.doPrivileged(Native Method)
2025-04-15 18:58:59 cluster-master    |         at javax.security.auth.Subject.doAs(Subject.java:422)
2025-04-15 18:58:59 cluster-master    |         at org.apache.hadoop.security.UserGroupInformation.doAs(UserGrou
pInformation.java:1878)
2025-04-15 18:58:59 cluster-master    |         at org.apache.hadoop.mapred.YarnChild.main(YarnChild.java:172)
2025-04-15 18:58:59 cluster-master    |
2025-04-15 18:59:04 cluster-master    | 2025-04-15 15:59:04,118 INFO mapreduce.Job:  map 50% reduce 0%
2025-04-15 18:59:05 cluster-master    | 2025-04-15 15:59:05,140 INFO mapreduce.Job:  map 100% reduce 100%
```