

Simple Search Engine using Hadoop MapReduce

1. Methodology

This search engine was built as a modular, distributed pipeline designed for indexing and querying large-scale documents using the BM25 ranking function. The implementation leverages Hadoop MapReduce for document preprocessing, Apache Cassandra for distributed storage, and Apache Spark for ranking search results.

1.1 Indexing Pipeline

MapReduce

I utilize a two-step MapReduce process:

- `mapper1.py`:
 - Parses each input line into `doc_id`, `doc_title`, and `doc_text`.
 - Tokenizes and normalizes text by removing punctuation and converting to lowercase.
 - Computes term frequencies (TF) per document.
 - Stores document metadata and term frequencies into Cassandra (docs and tfs tables).
 - Emits intermediate output in the format `<term>\t<doc_id>` for computing document frequencies.
- `reducer1.py`:
 - Aggregates the number of documents in which each term appears (document frequency, DF).
 - Stores DF values into the `dfs` table in Cassandra.

Index Creation Script (`index.sh`)

This script:

- Runs the Hadoop MapReduce job using the Hadoop streaming API.
- Uses a compressed Python virtual environment (`.venv.tar.gz`) for dependency isolation.
- After indexing, executes `app.py` to compute the average document length (`dl_avg`) across the corpus, storing it in a dedicated Cassandra table.

1.2 Cassandra Data Storage

The following Cassandra tables are used:

- docs(doc_id TEXT PRIMARY KEY, doc_title TEXT, dl INT): Stores titles and document lengths.
- tfs(doc_id TEXT, term TEXT, tf INT, PRIMARY KEY (doc_id, term)): Stores term frequencies.
- dfs(term TEXT PRIMARY KEY, df INT): Stores document frequencies.
- dl_avg(index_name TEXT PRIMARY KEY, dl_avg DOUBLE): Stores average document length.

1.3 Query Processing and Ranking

Queries are processed using query.py, which uses Apache Spark to:

- Load the list of doc_ids from Cassandra.
- Preprocess the query (tokenization, normalization).
- For each document, compute a BM25 score

1.4 Script Automation

- app.sh: Executes the indexing and retrieval pipelines.
- search.sh: Accepts a query string and triggers query.py.

1. Demonstration

How to run

Step 1: Install prerequisites:

Docker

Docker compose

Step 2: Run the command: docker compose up

This will create 3 containers, a master node and a worker node for Hadoop, and Cassandra server. The master node will run the script app/app.sh as an entry point.

```
app > $ app.sh
8 # Creating a virtual environment
9 python3 -m venv .venv
10 source .venv/bin/activate
11
12 # Install any packages
13 pip install -r requirements.txt
14
15 # Package the virtual env.
16 venv-pack -o .venv.tar.gz
17
18 # Collect data
19 bash prepare_data.sh
20
21
22 # Run the indexer
23 bash index.sh /index/data
24
25 # Run the ranker
26 bash search.sh "this is a query!"
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL CONSOLE

[+] Running 4/4

- ✓ Network big-data-assignment2-2025_spark-cluster Created 0.1s
- ✓ Container cassandra-server Started 0.4s
- ✓ Container cluster-slave-1 Started 0.4s
- ✓ Container cluster-master Started 0.6s

Restarting OpenBSD Secure Shell server sshd [OK]

Starting namenodes on [cluster-master]

Starting datanodes

cluster-slave-2: ssh: Could not resolve hostname cluster-slave-2: Temporary failure in name resolution

cluster-slave-3: ssh: Could not resolve hostname cluster-slave-3: Temporary failure in name resolution

cluster-slave-4: ssh: Could not resolve hostname cluster-slave-4: Temporary failure in name resolution

cluster-slave-5: ssh: Could not resolve hostname cluster-slave-5: Temporary failure in name resolution

Starting secondary namenodes [cluster-master]

Starting resourcemanagers

Starting nodeManagers

cluster-slave-2: ssh: Could not resolve hostname cluster-slave-2: Temporary failure in name resolution

cluster-slave-5: ssh: Could not resolve hostname cluster-slave-5: Temporary failure in name resolution

cluster-slave-3: ssh: Could not resolve hostname cluster-slave-3: Temporary failure in name resolution

286 org.apache.hadoop.hdfs.server.namenode.SecondaryNameNode

598 org.apache.hadoop.yarn.server.resourcemanager.ResourceManager

939 org.apache.hadoop.mapreduce.v2.hs.JobHistoryServer

972 jdk-17-linux-x86_64-jre-17

159 org.apache.hadoop.hdfs.server.namenode.NameNode

Cluster-master Configured Capacity: 13201507712 (122.95 GB)

Cluster-master Present Capacity: 12127244696 (112.94 GB)

Cluster-master DFS Remaining: 121272422400 (112.94 GB)

Cluster-master DFS Used: 24576 (24 KB)

Cluster-master DFS Used%: 0.0%

Cluster-master Replicated Blocks:

Cluster-master Under replicated blocks: 0

Cluster-master Blocks with corrupt replicas: 0

Cluster-master Missing blocks: 0

Cluster-master Missing blocks (with replication factor 1): 0

Cluster-master Low redundancy blocks with highest priority to recover: 0

Cluster-master Pending deletion blocks: 0

Here's an example of indexer work:

```
app > query.py
4
5 from cassandra.cluster import Cluster
6 from pyspark.sql import SparkSession
7
8 cluster = Cluster(['cassandra-server'])
9 session = cluster.connect('bm25_index')
10
11 spark = SparkSession.builder \
12     .appName('query') \
13     .master("local") \
14     .config("spark.sql.parquet.enableVectorizedReader", "true") \
15     .getOrCreate()
16
17 # Save as mapper1.py
18 alphabet = set(string.ascii_lowercase)
19 query_terms = []
20 for term in sys.argv[1].split():
21     term = ''.join(c for c in term.lower() if c in alphabet)
22     if len(term) != 0:
23         query_terms.append(term)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL CONSOLE

--rw-r--r-- 1 root supergroup 3555833 2025-04-20 13:06 /index/data/part-00000-b824f750-a472-4597-9c28-ee55bccc68a4-c000.csv

(venv) root@cluster-master:app# bash index.sh /index/data

This script include commands to run mapreduce jobs using hadoop streaming to index documents

Input file is:

/index/data

packageJobJar: [/tmp/hadoop-unjar74026379473737018]/[] /tmp/streamjob6918744022015633560.jar tmpDir=null

2025-04-20 13:57:24,242 INFO client.DefaultHadoopRMFailoverProxyProvider: Connecting to ResourceManager at cluster-master/172.22.0.4:8032

2025-04-20 13:57:24,366 INFO client.DefaultHadoopRMFailoverProxyProvider: Connecting to ResourceManager at cluster-master/172.22.0.4:8032

2025-04-20 13:57:24,536 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/root/.staging/job_1745151969659_0001

2025-04-20 13:57:25,544 INFO mapred.FileInputFormat: Total input files to process : 1

2025-04-20 13:57:25,571 INFO mapreduce.JobSubmitter: number of splits:2

2025-04-20 13:57:25,662 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1745151969659_0001

2025-04-20 13:57:25,662 INFO mapreduce.JobSubmitter: Executing with tokens: []

2025-04-20 13:57:25,782 INFO conf.Configuration: resource-types.xml not found

2025-04-20 13:57:25,782 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.

2025-04-20 13:57:25,991 INFO impl.YarnClientImpl: Submitted application application_1745151969659_0001

2025-04-20 13:57:26,031 INFO mapreduce.Job: The url to track the job: http://cluster-master:8088/proxy/application_1745151969659_0001/

2025-04-20 13:57:26,032 INFO mapreduce.Job: Running job: job_1745151969659_0001

2025-04-20 13:57:38,134 INFO mapreduce.Job: Job job_1745151969659_0001 running in uber mode : false

2025-04-20 13:57:38,135 INFO mapreduce.Job: map 0% reduce 0%

2025-04-20 13:57:44,255 INFO mapreduce.Job: map 10% reduce 0%

2025-04-20 13:58:00,282 INFO mapreduce.Job: map 14% reduce 0%

2025-04-20 13:58:06,318 INFO mapreduce.Job: map 20% reduce 0%

2025-04-20 13:58:12,337 INFO mapreduce.Job: map 26% reduce 0%

2025-04-20 13:58:18,364 INFO mapreduce.Job: map 33% reduce 0%

2025-04-20 13:58:24,389 INFO mapreduce.Job: map 41% reduce 0%

2025-04-20 13:58:30,417 INFO mapreduce.Job: map 45% reduce 0%

2025-04-20 13:58:36,439 INFO mapreduce.Job: map 50% reduce 0%

2025-04-20 13:58:42,466 INFO mapreduce.Job: map 59% reduce 0%

2025-04-20 13:58:48,486 INFO mapreduce.Job: map 64% reduce 0%

2025-04-20 13:58:54,513 INFO mapreduce.Job: map 67% reduce 0%

2025-04-20 13:58:55,527 INFO mapreduce.Job: map 83% reduce 0%

2025-04-20 13:58:56,537 INFO mapreduce.Job: map 100% reduce 0%

2025-04-20 13:59:11,579 INFO mapreduce.Job: map 100% reduce 95%

2025-04-20 13:59:17,598 INFO mapreduce.Job: map 100% reduce 95%

2025-04-20 13:59:20,668 INFO mapreduce.Job: map 100% reduce 100%

2025-04-20 13:59:21,614 INFO mapreduce.Job: Job job_1745151969659_0001 completed successfully

```
def bm25_score(doc_id):
    for term in query_terms:
        if tf is None:
            tf = tf.tf
            idf = math.log10(N / df)
            numerator = (k1 + 1) * tf
            denominator = k1 * ((1 - b) + b * (dl / d_avg)) + tf
            score += idf * (numerator / denominator)
    return (doc_id, score)

bm25_rdd = doc_ids_rdd.map(bm25_score)
bm25_top10_rdd = bm25_rdd.takeOrdered(10, key=lambda x: -x[1])
for doc_id, score in bm25_top10_rdd:
    print(doc_id, score, session.execute("SELECT doc_title FROM docs WHERE doc_id=%s", (doc_id,)).one()).doc_title
```

And we see the output (top-10).I entered the first sentence of a 843 document as a query:

```
def bm25_score(doc_id):
    for term in query_terms:
        if tf is None:
            tf = tf.tf
            idf = math.log10(N / df)
            numerator = (k1 + 1) * tf
            denominator = k1 * ((1 - b) + b * (dl / d_avg)) + tf
            score += idf * (numerator / denominator)
    return (doc_id, score)

bm25_rdd = doc_ids_rdd.map(bm25_score)
bm25_top10_rdd = bm25_rdd.takeOrdered(10, key=lambda x: -x[1])
for doc_id, score in bm25_top10_rdd:
    print(doc_id, score, session.execute("SELECT doc_title FROM docs WHERE doc_id=%s", (doc_id,)).one()).doc_title
```