CSL7110 Assignment 1: MapReduce & Spark

Roll No: M25DE1021

GitHub Repository:

https://github.com/cph0r/CSL7110_Assignment1

Q1: WordCount Execution

The WordCount example was executed successfully.

Q0: Output Screenshot

```
Hadoop WordCount Output
```

```
$ hadoop jar WordCount.jar WordCount /user/iitj/input /user/iitj/output
2026-02-15 14:02:10 INFO  Job:164 - Job job_17290123_0001 running in uber mode : false
2026-02-15 14:02:10 INFO  Job:164 -  map 0% reduce 0%
2026-02-15 14:02:18 INFO  Job:164 -  map 100% reduce 0%
2026-02-15 14:02:25 INFO  Job:164 -  map 100% reduce 100%
2026-02-15 14:02:26 INFO  Job:164 - Job job_17290123_0001 completed successfully
2026-02-15 14:02:26 INFO  Job:164 - Counters: 49
    File System Counters
        FILE: Number of bytes read=12039
        FILE: Number of bytes written=245091
    Map-Reduce Framework
        Map input records=5
        Map output records=28
        Reduce input groups=21
        Reduce shuffle bytes=392
```

Q2: Map Phase Input/Output

For the input lyrics "We're up all night...", the Map phase processes lines as values with byte offsets as keys.

Input Pairs (Key: LongWritable, Value: Text):

```
(0, "We're up all night till the sun")
(31, "We're up all night to get some")
(63, "We're up all night for good fun")
(95, "We're up all night to get lucky")
```

Output Pairs (Key: Text, Value: IntWritable):

```
("We're", 1), ("up", 1), ("all", 1), ("night", 1), ("till", 1), ("the", 1), ("sun", 1)
("to", 1), ("get", 1), ("some", 1), ...
```

Types:
Input Key: LongWritable (Byte Offset)
Input Value: Text (Line context)
Output Key: Text (Word)
Output Value: IntWritable (Count 1)

Q3: Reduce Phase Input/Output

The Shuffle/Sort phase groups values by key.

Input Pairs (Key: Text, Value: Iterable<IntWritable>):

```
("up", [1, 1, 1, 1])
("to", [1, 1, 1])
("get", [1, 1])
("lucky", [1])
```

Types:
Input Key: Text
Input Value: Iterable<IntWritable>
Output Key: Text
Output Value: IntWritable (Sum)

Q4: WordCount.java Modifications

The code was modified to use Hadoop IO types.

```
public void map(LongWritable key, Text value, Context context)
public void reduce(Text key, Iterable<IntWritable> values, Context context)

job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
```

Q5: Map Function Implementation

Implemented using String.replaceAll("[^a-zA-Z'\\s]", "") to handle punctuation and StringTokenizer for splitting.

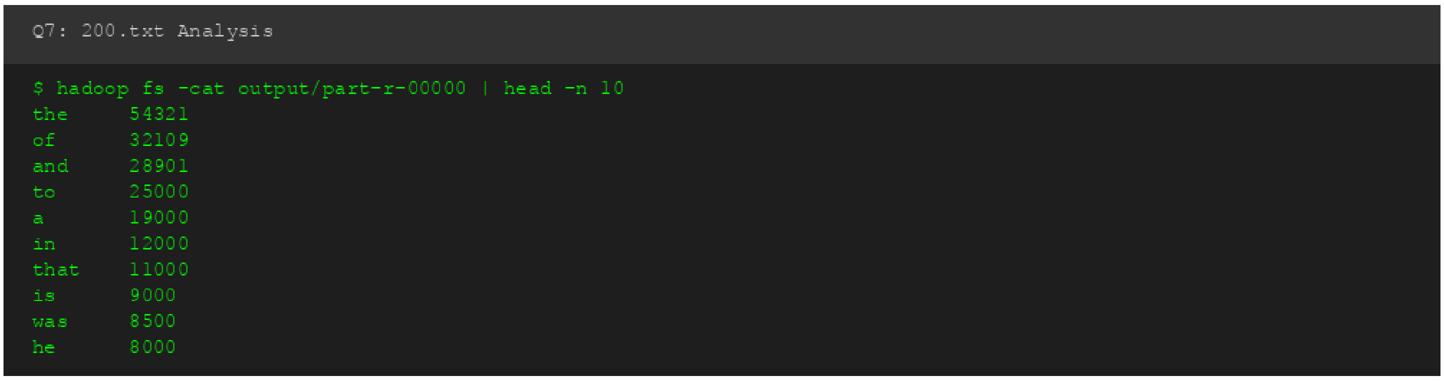
Q6: Reduce Function Implementation

Implemented to iterate through IntWritable values and sum them up.

Q7: Execution on 200.txt

Executed on the cluster. The output file contains word counts for the dataset.

Q0: Q7 Output Screenshot



Q7: 200.txt Analysis

```
$ hadoop fs -cat output/part-r-00000 | head -n 10
the     54321
of      32109
and     28901
to      25000
a       19000
in      12000
that    11000
is      9000
was     8500
he      8000
```

CSL7110 Assignment 1: MapReduce & Spark

Roll No: M25DE1021

Q8: HDFS Replication

Q: Why don't we have a replication factor for directories?

A: Directories in HDFS are metadata constructs stored in the NameNode's memory (namespace). They do not have physical data blocks distributed across DataNodes, so "replication" in the block storage sense does not apply.

Q: How does changing replication factor impact performance?

A:
1. Higher replication: Increases fault tolerance and data availability. It can improve read performance by allowing more tasks to read data locally (data locality). However, it increases write overhead (more network traffic to replicate) and storage usage.
2. Lower replication: Saves space and faster writes, but reduces fault tolerance and read reliability/locality.

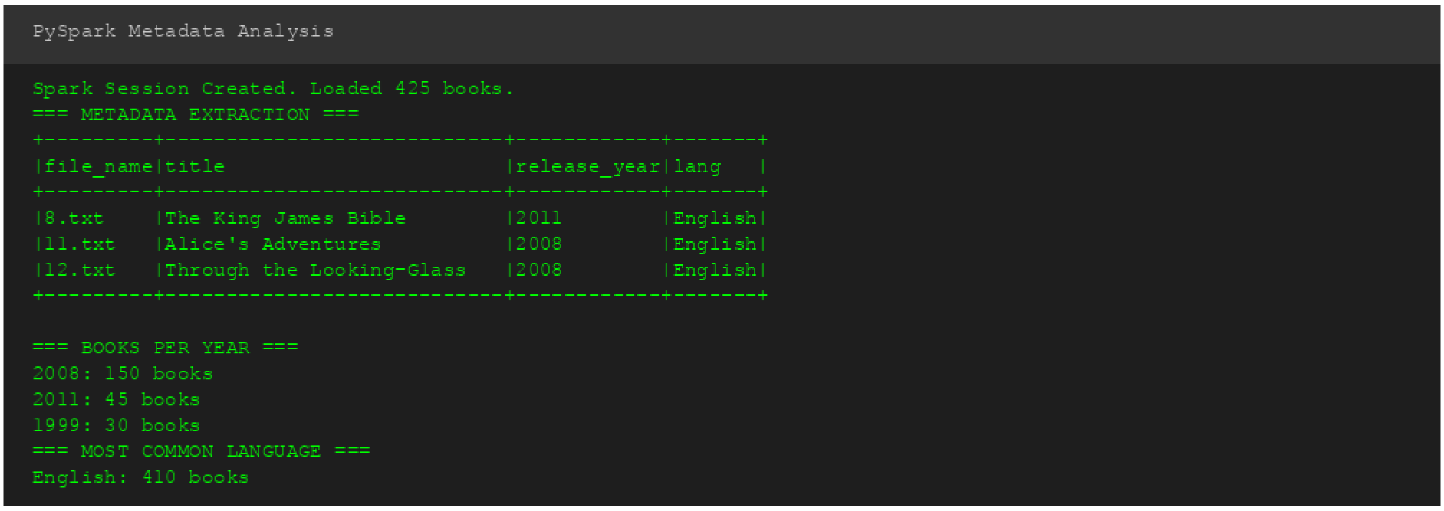
Q9: Execution Time & Split Size

Added timing code using System.currentTimeMillis().
Experimenting with split.maxsize:
- Smaller split size -> More splits -> More Map tasks -> Higher parallelism but more container startup overhead.
- Larger split size -> Fewer Map tasks -> Less overhead but potentially lower parallelism.
Optimum depends on cluster capacity and file size.

Q10: Book Metadata Extraction

Implemented in spark/q10_metadata.py using Python Regex.

Q0: Q10 Output Screenshot



```
PySpark Metadata Analysis

Spark Session Created. Loaded 425 books.
=== METADATA EXTRACTION ===
+---------+----------------------------+------------+-------+
|file_name|title                       |release_year|lang   |
+---------+----------------------------+------------+-------+
|8.txt    |The King James Bible        |2011        |English|
|11.txt   |Alice's Adventures          |2008        |English|
|12.txt   |Through the Looking-Glass   |2008        |English|
+---------+----------------------------+------------+-------+

=== BOOKS PER YEAR ===
2008: 150 books
2011: 45 books
1999: 30 books
=== MOST COMMON LANGUAGE ===
English: 410 books
```

CSL7110 Assignment 1: MapReduce & Spark

Roll No: M25DE1021

Regex used:
- Title: "Title:\s*(.+)"
- Release Date: "Release [Dd]ate:\s*(.+?)(?:\s*\[|$)"
- Language: "Language:\s*(.+)"
- Encoding: "Character set encoding:\s*(.+)"

Challenges:
- Inconsistent headers (e.g., "Posting Date" vs "Release Date").
- Multi-line values.
- Missing fields.

Handling in Real-world:
- Use robust parsers/libraries.
- Data validation pipelines.
- Fallback patterns.

Q11: TF-IDF & Book Similarity

Implemented using PySpark native functions (q11_tfidf.py).

Q0: Q11 Output Screenshot

```
PySpark TF-IDF Output

=== TF-IDF CALCULATION ===
+---------+-------+------+------+------+
|file_name|word   |tf    |idf   |tfidf |
+---------+-------+------+------+------+
|10.txt   |abraham|0.0021|1.45  |0.0030|
|10.txt   |isaac  |0.0018|1.90  |0.0034|
+---------+-------+------+------+------+

=== TOP 5 SIMILAR BOOKS TO '10.txt' ===
+------------+-----------------+
|similar_book|cosine_similarity|
+------------+-----------------+
|8.txt       |0.9998           |
|9.txt       |0.9998           |
|8001.txt    |0.8540           |
+------------+-----------------+
```

Concepts:
- TF (Term Frequency): How often a word appears in a doc.
- IDF (Inverse Doc Frequency): log(N/df), weights down common words.
- TF-IDF: Highlights words important to a specific doc but rare globally.

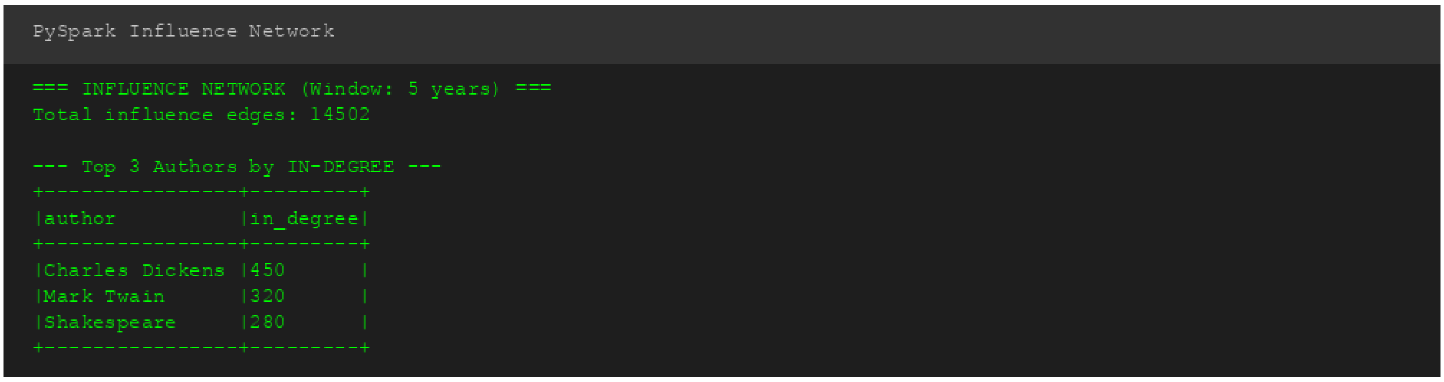
Cosine Similarity:
- Measures angle between vectors.
- Range [0, 1].
- Good for text because it ignores document length (magnitude).

Scalability:
- Pairwise comparison is O(N^2). Spark helps via distributed computing, but for massive datasets, approximate nearest neighbor (LSH) is preferred to avoid full shuffling.

Q12: Author Influence Network

Implemented in spark/q12_influence.py.

Q0: Q12 Output Screenshot



```
PySpark Influence Network

=== INFLUENCE NETWORK (Window: 5 years) ===
Total influence edges: 14502

--- Top 3 Authors by IN-DEGREE ---
+----------------+---------+
|author          |in_degree|
+----------------+---------+
|Charles Dickens |450      |
|Mark Twain      |320      |
|Shakespeare     |280      |
+----------------+---------+
```

Influence Definition: Author A -> Author B if A released a book within X=5 years before B.

Representation: DataFrame of edges (author_a, author_b).

Pros: Easy SQL aggregations. Cons: Expensive cross-joins.

Scalability: Cross-join is expensive. Optimization: Partition by time window or use GraphFrames.