Name:
Luis Castellanos
Student ID:
33489855

Purdue University (Fall 2025)
CS44000: Large-scale Data Analytics
Homework 1

IMPORTANT:

- Upload a pdf file with answers to Gradescope.
- Please use the either the latex template or word template to write down your answers and generate a pdf file.
  - Latex template: `https://www.cs.purdue.edu/homes/csjgwang/CS440/template.tex`
  - Word template: `https://www.cs.purdue.edu/homes/csjgwang/CS440/template.docx`

Problem	Score
1	
2	
3	
4	
5	
Total	

Problem 1 [20 points]

1. During the map phase we read each document and tokenize the words in the following key-value (word, docID), since we know there are no duplicates within each document there is no need to count within one document.

   During the shuffle phase we proceed to group same words from all files (word, list of docID)

   Finally we enter the reduce phase, we aggregate docIDs and remove duplicates if any, and we get (word, v3), in our case v3 is a list of values

   Example: (Hello, 1), (World, 1), (Hello, 2), (Data, 2), (Big, 3), (Data, 3) → (Hello, list(1,2)), (World, list(1)), (Data, list(2,3)), (Big, list(3))
   → (Hello, 1,2), (World, 1), (Data, 2,3), (Big, 3) (I started counting at 1 but in the example they start at 0)

```
Map(docId, documentText):
    words = documentText.split()
    for w in words:
        Emit(w, docId)

Reduce(word, docIdList):
    unique = RemoveDuplicates(docIdList)
    Emit(word, unique)
```

2.

Problem 2 [20 points]

1. In regular DBs we have logs, checkpoints, write-ahead logging. We must restore the state exactly before crash, hence we have strict durability and consistency. So the pros is that we have strong guarantees and safe for transactions. The cons is that there is heavy overhead, slow in massive distributed joins. Hadoop does not have transaction logs, if a crash occurs, we just need to reexecute failed tasks, because tasks are deterministic and input stored in HDFS. So, the pros are simple, fault-tolerant and scalable. The cons we have no ACID and it is not suited for live transactional workloads.

2. In Hadoop, we have recovery by rerunning map/reduce tasks, there is no lineage tracking. For Spark, we have RDD lineage graph to recompute only lost partitions, not entire job. This is a faster recovery and avoids full recomputation. The cons are that the lineage graph must be stored.

Problem 3 [20 points]

1. We use HBase when data is sparse, wide, semi-structured or column oriented. Also, when we need masive scalability and random read/write. (DB row store becomes inefficient when most columns are empty)

   The advantages of using HBase are that column families are stored separately, which is more efficient for sparse storage. It is distributed and fault-tolerant via HDFS, and and we have high throughput and random access.

2. HDFS is optimized for sequential access to very large files and batch processing, which makes it good for log storage and offline analytics. However, it cannot support random reads well or small updates because files are written once and acessed by the bunch.
   HBase, built on top of HDFS, provides a schema layer that enables random cell access and updates but requires additional coordination. In short, HDFS is simpler and best for high-throughput scans, while HBase is more complex but enables faster data lookup and modification.

Problem 4 [20 points]

1. RDDs allow Spark to maintain fault tolerance through lineage rather than replication. If something like a partition is lost, Spark reconstructs only that partition using previous history, rather than restarting the job. This recovery model enables fast distributed computation and efficient in-memory processing.

2. Spark delays execution until an action occurs so that it can analyze the entire plan and optimize it before running. This avoids computing intermediate results that are never used and reduces unnecessary communication between nodes, improving performance and planning efficiency.

3. A distributed join is a wide dependency because keys must be shuffled across the cluster before merged. This causes each output partition to depend on multiple parent partitions, which requires full data redistribution.

Problem 5 [20 points]

1. The nice thing about BSP is that each superstep completes fully before the next begins, ensuring consistent views of graph state. However, the con is that strict synchronization means slow nodes delay the entire system and communication overhead increases.

2. Separating compute from storage allows independent scaling and lets systems maintain cheap persistent storage while compute resources scale only when needed. The drawback is latency, since data must be accessed remotely and consistency, it must be maintained across several distributed components.