Midterm Exam

COEN 242 Spring 2021

Q1. (30 pts) Multi-select questions (6×5 pts). For each multi-select question, we will dock 3 pts for incomplete selections and 5 pts if there are any incorrect choices.

(1) [ ] Please select all the immutable data types (in Python or Spark) from the following.

- (A) String
- (B) RDD
- (C) Tuple
- (D) List

(2) [ ] Please select all the possible limitations of a MapReduce algorithm.

- (A) The MapReduce algorithm will cause high I/O operations on disks.
- (B) The MapReduce algorithm will need lots of memory space to save the intermediate results.
- (C) The MapReduce algorithm cannot work with $(key, value)$ input data.
- (D) The MapReduce algorithm is inflexible to work with one-pass jobs.

(3) [ ] Please select all the functions that could be used in a Spark reduce action.

- (A) addConstant = lambda x: x+2
- (B) matEleMul = lambda A,B: numpy.multiply(A, B) # element-wise matrix multiply
- (C) squareDiff = lambda x,y: x**2-y**2
- (D) addList = lambda x,y: [x] + [y]

(4) [ ] Given an input RDD inp=sc.parallelize(range(100)).map(lambda x: (x,1)), please select all the options that will return an RDD variable containing a partitioner (numPartitions=5).

- (A) reu = inp.partitionBy(5).collectAsMap()
- (B) reu = inp.partitionBy(5).filter(lambda x: x[0]%5 != 1)
- (C) reu = inp.map(lambda x: (x[0]%10, x[1])).reduceByKey(lambda x,y: x+y, 5)
- (D) reu = inp.partitionBy(5).map(lambda x: (x[0], x[0]%5))

(5) [ ] Given two paired RDDs and a pre-defined hashing function f (different from the Python default hash function), tabA=tabA.partitionBy(10) and tabB=tabB.partitionBy(8), please select all the options that will cause a *shuffle* operation.

- (A) tabA.join(tabB)
- (B) tabC = tabA.partitionBy(10, f)
- (C) tabC = tabA.reduceByKey(lambda x,y:x+y); tabC.join(tabB)
- (D) tabC = tabA.mapValues(lambda x: x+2); tabA.join(tabC)

(6) [ ] Please select all the facts about lazy evaluation from the following.

- (A) Lazy evaluation will not execute RDD transformations until required by an action function.
- (B) Lazy evaluation is useful and efficient when computing RDDs in an iterative algorithm.
- (C) Lazy evaluation saves all the RDD variables with DAG in memory to achieve resilience.
- (D) Lazy evaluation is one of the keys to achieve fault tolerance in Spark.

Q2. (10 pts) Select **T**-"true" or **F**-"false" for each of the following statements (5×2 pts).

(1) [   ] When running MapReduce programs on a distributed file system (DFS), it is common to assign multiple mapping tasks on the same DFS chunk.

(2) [   ] When running MapReduce in parallel, all the keys will be shuffled to different reduce tasks and will be sorted globally across different nodes.

(3) [   ] Spark outperforms MapReduce since it enables in-memory data processing and provides more flexible APIs.

(4) [   ] The aggregate function doesn't involve a communication cost among different partitions since it doesn't cause a shuffle to the keys.

(5) [   ] When a Spark driver program ships a mapping function to different partitions, the local variables contained by this function will also be shipped.

Q3. (15 pts) Given a pair RDD x=sc.parallelize([(1, 'a'), (3, 'a'), (2, 'a'), (1, 'b'), (4, 'c')]), please give the Spark solution for the following questions.

a) How to explicitly distribute x into two partitions? By using the default hash function, how the key-value pairs will be distributed across these two partitions? Give the partitions results by using two tables to show the pairs in each partition, respectively. (Hint: target_partition = hash(key) % num_partitions)

b) Based on the partitioned RDD in a), show the codes to invert the key and value for each pair.

c) Based on the new pair RDD given in b), compute the per-key average with combineByKey. Will this process involve a shuffle operation?

Consider these documents:

Doc 1 breakthrough drug for schizophrenia
Doc 2 new schizophrenia drug
Doc 3 new approach for treatment of schizophrenia
Doc 4 new hopes for schizophrenia patients

Figure 1: A small corpus with four documents.

Q4. (25 pts) Given a small corpus as shown in Fig. 1, please answer the following questions. **1)** Term-document frequency matrix (5 pts). Draw a term-document frequency matrix for the given corpus, where the rows are corresponding to all the unique words and each column is one document. Please sort the words in alphabetical order ($a \rightarrow z$ from top to bottom). The result should be a 10×4 table. **2)** TF-IDF (5 pts). Based on the term-document matrix given in **1)**, compute the TF-IDF values for each document, which should give another 10×4 table. The IDF value is computed by $\log \frac{|C|}{|\{F \in C : w \in F\}|+1}$, where C represents the given corpus in Fig. 1, F denotes one document in C, and w refers to a word. (Hint: You may use your HW1's code for computing the values). **3)** Spark Program Design (15 pts). Cosine similarity is a common way to compare the similarity between two documents upon their TF-IDF features (values). Given two documents F and F' in our corpus C, the cosine similarity is defined as

$$\cos(F, F') = \frac{\sum_{w \in F \cap F'} \mathtt{TFIDF}(w, F) \cdot \mathtt{TFIDF}(w, F')}{\sqrt{\sum_{w \in F} (\mathtt{TFIDF}(w, F))^2} \cdot \sqrt{\sum_{w \in F'} (\mathtt{TFIDF}(w, F'))^2}},$$

where $\mathtt{TFIDF}(w, F)$ denotes the TF-IDF value of a word w in the document F. Please implement a function with pyspark to compute the cosine similarity between two given documents' TF-IDF values. The function should be programmed with two input variables as

```
def SIM(sc, inpDoc1, inpDoc2):
    # inpDoc1 and inpDoc2 are the saved TF-IDF files of two documents
    return similarity
```

Similar to HW1, you should save the TF-IDF values of each document in advance, and load them for implementing the SIM function. **The provided implementation of SIM must be executable in a Spark session**. Incorrect codes may lose all the points. By using SIM, what is the cosine similarity between Doc1 and Doc3 in Fig. 1?

Hint: As a take-home exam, you could attach a screenshot of your implementation to your final submission.

Q5. (20 pts) Given a web graph $\mathcal{G}$ as shown in Fig. 2, please answer the following questions.

a) Compute the transition matrix M of the given web graph $\mathcal{G}$.

b) Let the stop criterion $\epsilon = 0.01$. By computing the convergence criterion with the ℓ_1 norm, what is the final PageRank score of each node in $\mathcal{G}$? Show the steps (included intermediate results in each iteration) using power iteration to compute PageRank scores with the transition matrix M given in a).

c) Let $\beta = 0.8$, compute the Google's matrix A for the given web graph $\mathcal{G}$.

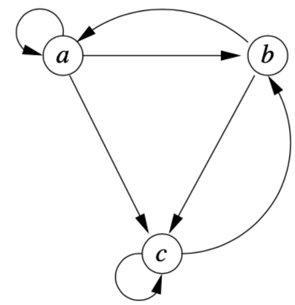

Figure 2: A web graph of three web pages.

d) Let the stop criterion $\epsilon = 0.01$. By computing the convergence criterion with the ℓ_∞ norm, what is the final PageRank score of each node in $\mathcal{G}$? Show the steps (included intermediate results in each iteration) using power iteration to compute PageRank scores with the Google's matrix A given in c).

Hint: Denote the PageRank score of each node in $\mathcal{G}$ as a vector $r \in \mathbb{R}^3$.
Hint: Given a vector $x \in \mathbb{R}^d$, the ℓ_1 norm is $\|x\|_1 = \sum_{i=1}^{d}(|x_i|)$ and the ℓ_∞ norm is $\|x\|_\infty = \max\{|x_1|, \ldots, |x_d|\}$.