CS 644: Homework 3

1. (6 points) **Latency and fault-tolerance**.

   (a) *Latency* is degradation in performance due to... (choose two)

   □ a small number of cores in the central processing unit
   □ slow data transfer across the network or cluster
   □ shuffling data between different nodes in a cluster
   □ failure of one or more nodes in the cluster
   □ stack overflow caused by recursion

   (b) Hadoop achieves fault-tolerance by... (choose one)

   □ using lazy evaluation and garbage collection.
   □ writing intermediate computations to disk.
   □ keeping all data immutable and in-memory.
   □ replaying functional transformations over the original (immutable) dataset.

   (c) Spark decreases latency while remaining fault-tolerant by... (choose three)

   □ using ideas from imperative programming.
   □ using ideas from functional programming.
   □ discarding data when it's no longer needed.
   □ keeping all data immutable and in-memory.
   □ replaying functional transformations over the original (immutable) dataset.

2. (6 points) **Transformations and actions**.

   (a) In Spark a **transformation** on an RDD... (choose three)

   □ is eagerly evaluated.
   □ is lazily evaluated.
   □ immediately computes and returns a result.
   □ does not immediately compute a result.
   □ always returns another RDD (once it's evaluated).

   (b) In Spark an **action** on an RDD... (choose three)

   □ is eagerly evaluated.
   □ is lazily evaluated.
   □ immediately computes and returns a result.
   □ does not immediately compute a result.
   □ always returns another RDD (once it's evaluated).

   (c) After performing a series of transformations on an `RDD`, which of the following methods could you use to make sure those transformations are not repeated (e.g., on each iteration of an algorithm)? (choose one)

   □ `save`
   □ `persist`
   □ `memoize`
   □ There is no such method because of the JVM's garbage collection mechanism.

(d) Why does Spark's RDD class not have a foldLeft method?

- □ foldLeft can only be performed on lists of Boolean values.
- □ foldLeft doesn't work on immutable collections.
- □ foldLeft is not stack-safe.
- □ foldLeft is not fault-tolerant.
- □ foldLeft is not parallelizable.

(e) Why is available in Spark's RDD class that overcomes the limitation of foldLeft mentioned in the previous part of this exercise?

□ aggregate □ fold □ foldLeft □ join □ leftOuterJoin

3. (4 points) **Read the docs**. Navigate to the Spark API documentation at

`https://spark.apache.org/docs/3.3.1/api/scala/org/apache/spark/index.html`

Enter "RDD" in the search box and select RDD from the results that appear on the left.

(a) Scroll down the resulting RDD API documentation page and find the cache() method. What does it say?

- □ Persist this RDD with the default storage level (MEMORY_ONLY).
- □ Mark the RDD as non-persistent, and remove all blocks for it from memory and disk.
- □ Set this RDD's storage level to persist its values across operations after the first time it is computed.
- □ Save this RDD as a SequenceFile of serialized objects.

(b) On the RDD API doc page, find the version of persist that takes an argument: def persist(newLevel: StorageLevel). What does it say?

- □ Persist this RDD with the default storage level (MEMORY_ONLY).
- □ Mark the RDD as non-persistent, and remove all blocks for it from memory and disk.
- □ Set this RDD's storage level to persist its values across operations after the first time it is computed.
- □ Save this RDD as a SequenceFile of serialized objects.

(c) On the RDD API doc page, find the unpersist method. What does it say?

- □ Persist this RDD with the default storage level (MEMORY_ONLY).
- □ Mark the RDD as non-persistent, and remove all blocks for it from memory and disk.
- □ Set this RDD's storage level to persist its values across operations after the first time it is computed.
- □ Save this RDD as a SequenceFile of serialized objects.

(d) What's the difference between the `sample` and `takeSample` methods of the `RDD` class? (choose two)

- □ `sample` always uses a with-replacement sampling method, while `takeSample` always samples without replacement.
- □ `sample` returns an `RDD`, while `takeSample` returns an `Array`.
- □ The second argument specifies the number of samples desired either as a fraction of the size of the RDD (`sample`) or as an absolute number (`takeSample`).
- □ There is no difference; they are just two different names one can use to invoke the same function.

4. (4 points) *Everyday I'm shufflin.*

(a) What is shuffling?

- □ a method for recovering data after hardware failure
- □ the method used to ensure a random number generator is unbiased
- □ any movement of data
- □ moving data from memory to disk, usually caused by insufficient fast memory
- □ transferring data between nodes in a cluster, usually in order to complete a computation

(b) How can shuffling sometimes be reduced or avoided using Spark?

- □ use higher quality, fault-tolerant hardware
- □ use a pre-shuffled random number generator
- □ avoid algorithms that process the entire data set in favor of algorithms that only need a small subset of it
- □ use only fast memory, eliminating all spinning disks from the network
- □ partition an `RDD` before applying transformations or actions that cause shuffling