Spark Lesson 3

1.

Check all true statements about the Directed Acyclic Graph Scheduler



The DAG is managed by the cluster manager



A DAG is used to track dependencies of each partition of each RDD



Each transformation is executed as soon as it is called on a RDD



If a partition is lost, the DAG is traversed forward to check what other steps are affected

2.

Why is building a DAG necessary in Spark but not in MapReduce?



For resiliency: it is necessary to make sure a partition can be recovered in case it is lost.



Because MapReduce always has the same type of workflow, Spark needs to accommodate diverse workflows.



In order to make a computation distributed at large scale

3.

What are the differences between an action and a transformation? Mark all that apply



An action always triggers a shuffle.



An action always writes the disk.



A transformation is from worker nodes to worker nodes, an action between worker nodes and the Driver (or a data source like HDFS)



A transformation is lazy, an action instead executes immediately.

4.

Generally, which are good stages to mark a RDD for caching in memory?



After data cleaning, parsing and validation.



At the start of an iterative algorithm.



Every 2 or 3 transformations, to keep a recent backup.



The first RDD, just after reading from disk, so we avoid reading from disk again.

5.

What are good cases for using a broadcast variable? Mark all that apply



Copy a large configuration dictionary to all worker nodes



Broadcast a Python module to all worker nodes



Copy a small/medium sized RDD for a join



Copy a large lookup table to all worker nodes

6.

We would like to count the number of invalid entries in this example dataset:

invalid = sc.accumulator(0)

d = sc.parallelize(["3", "23", "S", "99", "TT"]).foreach(count\_invalid)

What would be a good implementation of the count\_invalid function?



def count\_invalid(element):

try:

int(element)

except:

invalid = invalid.add(1)



def count\_invalid(element):

try:

int(element)

except:

invalid.add(1)



def count\_invalid(element):

try:

int(element)

except:

invalid.accumulate(1)



def count\_invalid(element):

try:

int(element)

except:

invalid = invalid + 1