Paul 5 - faster do...

```
%pyspark                                                                     FINISHED

# Zeppelin notebook to create domain summaries based on the May/Jun/Jul 2017 CommonCraw
# as per description here: http://commoncrawl.org/2017/08/webgraph-2017-may-june-july/
# PJ - 11 October 2017

import boto
from pyspark.sql.types import *

#LIMIT=10000000 # Temporary limit while developing code.

# Import the PLD vertices list as a DataFrame
#pld_schema=StructType([StructField("ID", StringType(), False), StructField("PLD", Stri
#pld_txt=sc.textFile("s3://commoncrawl/projects/hyperlinkgraph/cc-main-2017-may-jun-jul
#temp_pld = pld_txt.map(lambda k: k.split()) # By default, splits on whitespace, which
#pld_df=temp_pld.toDF(pld_schema) #.limit(LIMIT) #.repartition(4)
#pld_df.show(3)

# Load in an uncompressed, partitioned format, for fast reading in the future
saveURI="s3://billsdata.net/CommonCrawl/hyperlinkgraph/cc-main-2017-may-jun-jul/domaing
#pld_df.coalesce(64).write.save(saveURI) # Use all default options
pld_df=spark.read.load(saveURI)
pld_df.show(3)
pld_df.cache()
print(pld_df.count()) # Should have 91M domains
```

```
+---+-------+
| ID|    PLD|
+---+-------+
|  0|  aaa.a|
|  1| aaa.aa|
|  2|aaa.aaa|
+---+-------+
only showing top 3 rows
91034128
```

```
%pyspark                                                                     FINISHED

# Next import the PLD edges as a DataFrame
#pld_edges_schema=StructType([StructField("src", LongType(), False), StructField("dst",
#pld_edges_txt=sc.textFile("s3://commoncrawl/projects/hyperlinkgraph/cc-main-2017-may-j
#temp_edges_pld = pld_edges_txt.map(lambda k: map(int, k.split())) # By default, splits
#pld_edges_df=temp_edges_pld.toDF(pld_edges_schema) #.limit(LIMIT*10) #.repartition(8)
#pld_edges_df.show(3)

# Load in an uncompressed, partitioned format, for fast reading in the future
saveURI="s3://billsdata.net/CommonCrawl/hyperlinkgraph/cc-main-2017-may-jun-jul/domaing
#pld_edges_df.coalesce(64).write.save(saveURI) # Use all default options
```

```
pld_edges_df=spark.read.load(saveURI)
pld_edges_df.show(3)
pld_edges_df.cache()
```

```
+---+--------+
|src|     dst|
+---+--------+
|  2| 9193244|
| 20|75600973|
| 21|46356172|
+---+--------+
only showing top 3 rows
DataFrame[src: bigint, dst: bigint]
```

```
%pyspark

# Load the host-level graph vertices in the same way
#host_schema=StructType([StructField("hostid", StringType(), False), StructField("host"
#host_txt=sc.textFile("s3://commoncrawl/projects/hyperlinkgraph/cc-main-2017-may-jun-ju
#temp_host = host_txt.map(lambda k: k.split()) # By default, splits on whitespace, whic
#host_df=temp_host.toDF(host_schema) #.repartition(4)
#host_df.show(3)

# Save in an uncompressed, partitioned format, for fast reading in the future
saveURI="s3://billsdata.net/CommonCrawl/hyperlinkgraph/cc-main-2017-may-jun-jul/hostgra
#host_df.coalesce(128).write.save(saveURI) # Use all default options
host_df=spark.read.load(saveURI)
host_df.show(3)
host_df.cache()
#print(host_df.count()) # Should have 1.3B hosts
```

FINISHED

```
+------+-------+
|hostid|   host|
+------+-------+
|     0|  aaa.a|
|     1| aaa.aa|
|     2|aaa.aaa|
+------+-------+
only showing top 3 rows
DataFrame[hostid: string, host: string]
```

```
%pyspark

# Load in all harmonic centrality and page-ranks, and join based on reverse domain name
# Format: #hc_pos #hc_val #pr_pos #pr_val #host_rev
#pr_schema=StructType([StructField("hc_pos", StringType(), False), StructField("hc_val"
    StructField("host_rev", StringType(), False)])
#pr_txt=sc.textFile("s3://commoncrawl/projects/hyperlinkgraph/cc-main-2017-may-jun-jul/
#header=pr_txt.first()
#pr_txt=pr_txt.filter(lambda x: x!=header)
#temp_pr = pr_txt.map(lambda k: k.split()) # By default, splits on whitespace, which is
#pr_df=temp_pr.toDF(header.split()).withColumnRenamed("#host_rev","host_rev") #.limit(L
```

FINISHED

```
#pr_df.show(3)

# Save in an uncompressed, partitioned format, for fast reading in the future
saveURI="s3://billsdata.net/CommonCrawl/hyperlinkgraph/cc-main-2017-may-jun-jul/domaing
#pr_df.coalesce(64).write.save(saveURI) # Use all default options
pr_df=spark.read.load(saveURI)
pr_df.show(3)
pr_df.cache()
```

```
+-------+--------+-------+------------------+--------------+
|#hc_pos| #hc_val|#pr_pos|           #pr_val|      host_rev|
+-------+--------+-------+------------------+--------------+
|      1|24989952|      1| 0.0155264576161686|  com.facebook|
|      2|22460880|      3|0.00866038900847366|   com.twitter|
|      3|22097514|      2| 0.0128827315785546|com.googleapis|
+-------+--------+-------+------------------+--------------+
only showing top 3 rows
DataFrame[#hc_pos: string, #hc_val: string, #pr_pos: string, #pr_val: string, host_rev:
string]
```

```
%pyspark                                                                    FINISHED

# Debug partitioning of our 4 big dataframes
sc.getConf().getAll() #.mkString("\n")
print(pld_df.rdd.getNumPartitions())
print(pld_edges_df.rdd.getNumPartitions())
print(host_df.rdd.getNumPartitions())
pr_df.rdd.getNumPartitions()
```

```
64
64
117
64
```

```
%pyspark #--packages graphframes:graphframes:0.5.0-spark2.1-s_2.11          FINISHED

# We now have everything we need in these four dataframes to create the summaries we ne

# This code can't handle the complete edge lists, and produces this exception:
# java.lang.IllegalArgumentException: Size exceeds Integer.MAX_VALUE
#out_degrees_=dict(pld_edges_df.groupBy("src").count().collect())
#in_degrees=dict(pld_edges_df.groupBy("dst").count().collect())
#print(out_degrees['846558'])
#print(in_degrees['846558'])

# Instead, just create RDDs and use lookup()
out_degrees=pld_edges_df.groupBy("src").count()
in_degrees=pld_edges_df.groupBy("dst").count()
pld_edges_df.unpersist()
out_degrees.show(3)
in_degrees.show(3)
#print(out_degrees.rdd.lookup(846558))
```

```
#print(in_degrees_rdd.lookup(846558))
```

```
+---+-----+
|src|count|
+---+-----+
| 26|    2|
| 29|   34|
|964|    1|
+---+-----+
only showing top 3 rows
+--------+-----+
|     dst|count|
+--------+-----+
|      29|   40|
|36750820|    5|
|61427989| 3242|
+--------+-----+
only showing top 3 rows
```

```
%pyspark                                                                    FINISHED

# Next, we'll construct a local dictionary from of all the PLDS (key is the PLD, value
# This is our truth-table of known PLDs that we'll use when counting hosts
# This code can't handle the full PLD list and produces this exception:
# Stack trace: ExitCodeException exitCode=52
#pld_lookup_table=dict(pld_df.rdd.map(lambda x: (x['PLD'], x['ID'])).collect()) # Bad!
#print(pld_lookup_table["aaa.aaa"])

# Instead, just create an RDD and use lookup()
#pld_lookup_table=pld_df.rdd.map(lambda x: (x['PLD'], x['ID']))
#print(pld_lookup_table.lookup("aaa.aaa")) # Very bad!

# Or let's try creating as a BloomFilter, since we only want to record presence of a PL
#pld_bf = pld_df.stat.bloomFilter("PLD", expectedNumItems, fpp) # Doesn't exist in pysp
#pld_bf.mightContain("aaa.aaa")

# Create a bloom filter using a pure python package (might be a little slow)
from pybloom import BloomFilter
pld_bf = BloomFilter(capacity=91000000, error_rate=0.005)

for row in pld_df.limit(10000000).rdd.collect(): # TODO: Still bad (and exceeds spark.d
    pld_bf.add(row['PLD'])

print(pld_df.rdd.take(3))
print(pld_df.rdd.take(3)[2]['PLD'])
#pld_bf.add(pld_df.rdd.take(3)[2]['PLD'])
print("aaa.aaa" in pld_bf) # Should be True

# TODO: Fix this distributed BloomFilter iplementation - can't figure out how to proper
#tmp=pld_df.rdd.map(lambda x: pld_bf.add(x['PLD'])) # Very bad - pld_bf gets copied to
#tmp=pld_df.rdd.map(lambda x: (pld_bf.add(x['PLD']), pld_bf)).reduce(lambda x,y: x[1].u
#print(tmp.take(3))
#print(tmp.count()) # Ensure it runs the map across the entire dataframe
```

```
import sys
print(sys.getsizeof(pld_bf))
print(len(pld_bf)) # Should match number of items entered

# Broadcast the bloom filter so it's available on all the slave nodes - we don't need t
# it any more so it's fine being immutable.
pld_bf_distrib=sc.broadcast(pld_bf)

print("aaa.aaa" in pld_bf) # Should be true
print("aaa.aaa.bla" in pld_bf) # Should be false
print("aaa.aaa" in pld_bf_distrib.value) # Should be true
```

```
[Row(ID=u'0', PLD=u'aaa.a'), Row(ID=u'1', PLD=u'aaa.aa'), Row(ID=u'2', PLD=u'aaa.aaa')]
aaa.aaa
True
64
9978658
True
False
True
False
```

```
%pyspark                                                                    FINISHED

# Returns a Boolean to say whether PLD is a hostname in itself
def is_a_pld(hostname):
    #if hostname in pld_lookup_table:
    #if pld_lookup_table.filter(lambda a: a == hostname).count()>0:
    if hostname in pld_bf_distrib.value:
        return True
    else:
        return False

# Define a function to do the hostname->pld conversion, if the pld exists in our dictio
def convert_hostname(hostname):
    # Return hostname as-is, if this is already a PLD
    #if hostname in pld_lookup_table:
    #if pld_lookup_table.filter(lambda a: a == hostname).count()>0:
    if hostname in pld_bf_distrib.value:
        return hostname
    # Otherwise we're going to have to split it up and test the parts
    try:
        parts=hostname.split('.')
        if (len(parts)>4 and is_a_pld('.'.join(parts[0:4]))):
            return '.'.join(parts[0:4])
        if (len(parts)>3 and is_a_pld('.'.join(parts[0:3]))):
            return '.'.join(parts[0:3])
        if (len(parts)>2 and is_a_pld('.'.join(parts[0:2]))):
            return '.'.join(parts[0:2])
        if (len(parts)>1 and is_a_pld('.'.join(parts[0:1]))):
            return '.'.join(parts[0:1])
        return "ERROR" # Couldn't find a corresponding PLD - this should never happen!
    except:
        return "ERROR"
```

```
# Test
print(convert_hostname("aaa.aaa"))
```

```
aaa.aaa
True
```

```
%pyspark                                                                    ERROR

# Now count the number of hosts per PLD in a scalable way, and create another dictionar
# Still takes over an hour since host_df contains 1.3B rows but should complete without
# (An attempt to collectAsMap at the end results in java Integer.MAX_VALUE or memory er
count_rdd=host_df.drop('hostid').rdd.map(lambda x: (convert_hostname(x['host']),1)).red
bool_rdd=host_df.drop('hostid').rdd.map(lambda x: (x['host'], is_a_pld(x['host']))).fil

print(count_rdd.take(3))
print(bool_rdd.take(3))
print(count_rdd.count())
print(bool_rdd.count())

host_df.unpersist()

# Debugging
#print(count_rdd.filter(lambda x: x['['aaa.aaa'])
#print(bool_table['aaa.aaa'])
#print(count_table['ERROR']) # Should be zero once we've loaded all the PLDs!

# TODO: Fix error in collect()
# java.lang.IllegalArgumentException: Size exceeds Integer.MAX_VALUE
```

```
)
        at scala.collection.mutable.ArrayBuffer.foreach(ArrayBuffer.scala:48)
        at org.apache.spark.scheduler.DAGScheduler.abortStage(DAGScheduler.scala:1677)
        at org.apache.spark.scheduler.DAGScheduler$$anonfun$handleTaskSetFailed$1.apply(
DAGScheduler.scala:855)
        at org.apache.spark.scheduler.DAGScheduler$$anonfun$handleTaskSetFailed$1.apply(
DAGScheduler.scala:855)
        at scala.Option.foreach(Option.scala:257)
        at org.apache.spark.scheduler.DAGScheduler.handleTaskSetFailed(DAGScheduler.scal
a:855)
        at org.apache.spark.scheduler.DAGSchedulerEventProcessLoop.doOnReceive(DAGSchedu
ler.scala:1905)
        at org.apache.spark.scheduler.DAGSchedulerEventProcessLoop.onReceive(DAGSchedule
r.scala:1860)
        at org.apache.spark.scheduler.DAGSchedulerEventProcessLoop.onReceive(DAGSchedule
r.scala:1849)
        at org.apache.spark.util.EventLoop$$anon$1.run(EventLoop.scala:48)
        at org.apache.spark.scheduler.DAGScheduler.runJob(DAGScheduler.scala:671)
```

```
%pyspark                                                                    ERROR

#from pyspark.sql.types import IntegerType
```

```
#from pyspark.sql.functions import udf, col, when, lit

# This code works well when the data is small enough to collect into a python dictionar

# Define a UDF to perform column-based lookup
#def translate(mapping):
#    def translate_(col):
#        if not mapping.get(col):
#            return 0
#        else:
#            return mapping.get(col)
#    return udf(translate_, IntegerType())

# And a similar function for the Boolean map
#def translate_bool(mapping):
#    def translate_bool_(col):
#        if not mapping.get(col):
#            return False
#        else:
#            return mapping.get(col)
#    return udf(translate_bool_, BooleanType())

# Insert our count column back into the host summary dataframe, along with a boolean to
# While we're at it, let's add in the in and out-degrees too, and an indicator of wheth
#crawled_test=when(col("OutDegree")==0, lit(False)).otherwise(lit(True))
#pld_df_joined=pld_df.withColumn('NumHosts', translate(count_table)("PLD"))\
                    #.withColumn('PLDisHost?', translate_bool(bool_table)("PLD"))
                    #.withColumn('InDegree', translate(in_degrees)("ID"))\
                    #.withColumn('OutDegree', translate(out_degrees)("ID"))\
                    #.withColumn('Crawled?', crawled_test)

# Instead, just join our NumHosts and IsAPLD RDDs with the original dataframe
countschema=StructType([StructField("PLD2", StringType(), False), StructField("NumHosts
count_df=count_rdd.toDF(countschema)
count_df.show(3)
boolschema=StructType([StructField("PLD2", StringType(), False), StructField("PLDisHost
bool_df=bool_rdd.toDF(boolschema)
bool_df.show(3)
pld_df2=pld_df.join(count_df, count_df.PLD2==pld_df.PLD, "leftOuter").drop("PLD2")
pld_df_joined=pld_df2.join(bool_df, bool_df.PLD2==pld_df2.PLD, "leftOuter").drop("PLD2"

pld_df.unpersist()
pld_df_joined.sort("NumHosts", ascending=False).show(100)
pld_df_joined.cache()
Traceback (most recent call last):
  File "/tmp/zeppelin_pyspark-7300676184712394379.py", line 367, in <module>
    raise Exception(traceback.format_exc())
Exception: Traceback (most recent call last):
  File "/tmp/zeppelin_pyspark-7300676184712394379.py", line 355, in <module>
    exec(code, _zcUserQueryNameSpace)
  File "<stdin>", line 3, in <module>
  File "/usr/lib/spark/python/pyspark/sql/dataframe.py", line 336, in show
    print(self._jdf.showString(n, 20))
  File "/usr/lib/spark/python/lib/py4j-0.10.4-src.zip/py4j/java_gateway.py", line 1133,
in __call__
    answer, self.gateway_client, self.target_id, self.name)
  File "/usr/lib/spark/python/pyspark/sql/utils.py", line 63, in deco
    return f(*a, **kw)
  File "/usr/lib/spark/python/lib/py4j-0.10.4-src.zip/py4j/protocol.py", line 319, in ge
t_return_value
    format(target_id, ".", name), value)
Py4JJavaError: An error occurred while calling o397.showString
```

```
%pyspark                                                                    ERROR

# Join with in-degree and out-degree dataframes
pld_df_joined2=pld_df_joined.join(out_degrees, out_degrees.src==pld_df_joined.ID, "left(
pld_df_joined.unpersist()
pld_df_joined3=pld_df_joined2.join(in_degrees, in_degrees.dst==pld_df_joined2.ID, "left(
pld_df_joined2.unpersist()
pld_df_joined3.show(5)
pld_df_joined3.cache()
```

```
Traceback (most recent call last):
  File "/tmp/zeppelin_pyspark-7300676184712394379.py", line 367, in <module>
    raise Exception(traceback.format_exc())
Exception: Traceback (most recent call last):
  File "/tmp/zeppelin_pyspark-7300676184712394379.py", line 355, in <module>
    exec(code, _zcUserQueryNameSpace)
  File "<stdin>", line 1, in <module>
NameError: name 'pld_df_joined' is not defined
```

```
%pyspark                                                                    ERROR

from pyspark.sql.functions import col, when, lit

# Insert a flag to indicate whether the PLD has been crawled
crawled_test=when(col("OutDegree").isNull(), lit(False)).otherwise(lit(True))
pld_df_joined4=pld_df_joined3.withColumn('Crawled?', crawled_test)
pld_df_joined3.unpersist()
pld_df_joined4.show(5)
pld_df_joined4.cache()
```

```
Traceback (most recent call last):
  File "/tmp/zeppelin_pyspark-7300676184712394379.py", line 367, in <module>
    raise Exception(traceback.format_exc())
Exception: Traceback (most recent call last):
  File "/tmp/zeppelin_pyspark-7300676184712394379.py", line 355, in <module>
    exec(code, _zcUserQueryNameSpace)
  File "<stdin>", line 1, in <module>
NameError: name 'when' is not defined
```

```
%pyspark

# Finally, join with the harmonic centrality and page-rank for each domain
# Note: could probably speed this up using something like above techniques, or by preso
pld_df_joined5=pld_df_joined4.join(pr_df, pr_df.host_rev==pld_df_joined4.PLD, "leftOute
                             .withColumnRenamed("#hc_val","HarmonicCentrality").withCol
pld_df_joined4.unpersist()
pld_df_joined5.show(5)
pld_df_joined5.cache()
```

ERROR

```
Traceback (most recent call last):
  File "/tmp/zeppelin_pyspark-7300676184712394379.py", line 367, in <module>
    raise Exception(traceback.format_exc())
Exception: Traceback (most recent call last):
  File "/tmp/zeppelin_pyspark-7300676184712394379.py", line 355, in <module>
    exec(code, _zcUserQueryNameSpace)
  File "<stdin>", line 1, in <module>
NameError: name 'pld_df_joined4' is not defined
```

```
%pyspark

# Save final table to S3 in compressed CSV format, broken into smaller files
outputURI="s3://billsdata.net/CommonCrawl/domain_summaries2/"
codec="org.apache.hadoop.io.compress.GzipCodec"
pld_df_joined5.coalesce(16).write.format('com.databricks.spark.csv').options(header='tr
```

ERROR

```
Traceback (most recent call last):
  File "/tmp/zeppelin_pyspark-7300676184712394379.py", line 367, in <module>
    raise Exception(traceback.format_exc())
Exception: Traceback (most recent call last):
  File "/tmp/zeppelin_pyspark-7300676184712394379.py", line 360, in <module>
    exec(code, _zcUserQueryNameSpace)
  File "<stdin>", line 3, in <module>
NameError: name 'pld_df_joined5' is not defined
```

```
%pyspark
```

FINISHED