Paul 5 - faster do…

%pyspark FINISHED

```
# Zeppelin notebook to create domain summaries based on the May/Jun/Jul 2017
    CommonCrawl graph
# 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",
    StringType(), False)])
#pld_txt=sc.textFile("s3://commoncrawl/projects/hyperlinkgraph/cc-main-2017-may-jun
    -jul/domaingraph/vertices.txt.gz")
#temp_pld = pld_txt.map(lambda k: k.split()) # By default, splits on whitespace, which
    is what we want
#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
    /domaingraph/vertices/"
```

```
+---+-------+
| 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"
    , LongType(), False)])
#pld_edges_txt=sc.textFile("s3://commoncrawl/projects/hyperlinkgraph/cc-main-2017-may
    -jun-jul/domaingraph/edges.txt.gz")
#temp_edges_pld = pld_edges_txt.map(lambda k: map(int, k.split())) # By default,
    splits on whitespace, which is what we want
#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
    /domaingraph/edges/"
```

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

```
%pyspark                                                    FINISHED

# Load the host-level graph vertices in the same way
#host_schema=StructType([StructField("hostid", StringType(), False), StructField
    ("host", StringType(), False)])
#host_txt=sc.textFile("s3://commoncrawl/projects/hyperlinkgraph/cc-main-2017-may-jun
    -jul/hostgraph/vertices.txt.gz")
#temp_host = host_txt.map(lambda k: k.split()) # By default, splits on whitespace,
    which is what we want
#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
    /hostgraph/vertices/"
#host_df.coalesce(128).write.save(saveURI) # Use all default options
```

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

```
%pyspark                                                    FINISHED

# 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", StringType(), False), StructField("pr_pos", StringType(), False),
    StructField("pr_val", StringType(), False), StructField("host_rev", StringType(),
    False)])
#pr_txt=sc.textFile("s3://commoncrawl/projects/hyperlinkgraph/cc-main-2017-may-jun-jul
    /domaingraph/ranks.txt.gz")
```

```
#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 what we want
#pr_df=temp_pr.toDF(header.split()).withColumnRenamed("#host_rev","host_rev") #.limit
    (LIMIT*10) #.repartition(8)
#pr_df.show(3)
```

```
+-------+--------+-------+-------------------+-------------+
|#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()
```

```
128
128
128
128
```

```
%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
    need.

# 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))
```

```
+---+-----+
|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
    is the ID)
# 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
    PLD
#pld_bf = pld_df.stat.bloomFilter("PLD", expectedNumItems, fpp) # Doesn't exist in
    pyspark API!
#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
    .driver.MaxResultSize with all rows)!
    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
    properly combine BFs in a reduce operation!
```

```
#tmp=pld_df.rdd.map(lambda x: pld_bf.add(x['PLD'])) # Very bad - pld_bf gets copied to
    each of the workers then discarded!
#tmp=pld_df.rdd.map(lambda x: (pld_bf.add(x['PLD']), pld_bf)).reduce(lambda x,y: x[1]
    .union(y[1])) # Should work but complains that BloomFilter isn't iterable!
#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
```

```
[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
False
64
9970668
False
False
False
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
    dictionary
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
```

```
ERROR
False
```

```
%pyspark
```

FINISHED

```
# Now count the number of hosts per PLD in a scalable way, and create another
    dictionary
# Still takes over an hour since host_df contains 1.3B rows but should complete
    without errors.
# (An attempt to collectAsMap at the end results in java Integer.MAX_VALUE or memory
    errors!)
count_rdd=host_df.drop('hostid').rdd.map(lambda x: (convert_hostname(x['host']),1
    )).reduceByKey(lambda x,y: x+y) #.collectAsMap() # Consider using a CountMin
    sketch here in future?
bool_rdd=host_df.drop('hostid').rdd.map(lambda x: (x['host'], is_a_pld(x['host']
    ))).filter(lambda x: x[1]==True) #.collectAsMap() # Only outputs PLD hosts (so
    <91M rows)

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

host_df.unpersist()
```

```
[(u'com.topratedmichaelkorsoutlet', 1), (u'com.shoppingviponline', 1), (u'com.xinjistone
', 1)]
[(u'com.sergioarboleda', True), (u'com.sergioarcaya', True), (u'com.sergioarevalo', True
)]
9970669
9835722
DataFrame[hostid: string, host: string]
```

```
%pyspark
```

FINISHED

```
#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
    dictionary but our data is too big.

# 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 say whether the PLD is a host in itself
# While we're at it, let's add in the in and out-degrees too, and an indicator of
    whether the site has been crawled.
#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", LongType(), False)])
count_df=count_rdd.toDF(countschema)
count_df.show(3)
boolschema=StructType([StructField("PLD2", StringType(), False), StructField
    ("PLDisHost?", BooleanType(), False)])
bool_df=bool_rdd.toDF(boolschema)
bool_df.show(3)
```

```
|46405401|          com.vronex|  376022|      true|
|44930614|           com.toobt|  374277|      true|
|43110227|            com.t790|  373985|      true|
|41076029|      com.shuxiangcn|  365465|      true|
|47950322|           com.wvamt|  360947|      true|
|48423989|      com.yanhanhome|  360394|      true|
|45670750|         com.tzjiuge|  359158|      true|
|48986909|    com.zhuangxiuapp|  356508|      true|
|49266024|           cz.booked|  330839|      true|
|49112755|          com.ztkyhl|  321460|      null|
|47794541|    com.worddetector|  305116|      true|
|48467181|          com.ycsthy|  303559|      true|
|42124715|     com.squarespace|  298593|      true|
|46940414|      com.wallinside|  297138|      true|
|47181275|            com.webs|  292218|      true|
+--------+--------------------+--------+----------+
only showing top 100 rows
DataFrame[ID: string, PLD: string, NumHosts: bigint, PLDisHost?: boolean]
```

```
%pyspark
```

FINISHED

```
# Join with in-degree and out-degree dataframes
pld_df_joined2=pld_df_joined.join(out_degrees, out_degrees.src==pld_df_joined.ID,
    "leftOuter").drop("src").withColumnRenamed("count","OutDegree")
pld_df_joined.unpersist()
pld_df_joined3=pld_df_joined2.join(in_degrees, in_degrees.dst==pld_df_joined2.ID,
    "leftOuter").drop("dst").withColumnRenamed("count","InDegree")
pld_df_joined3.unpersist()
```

```
+----+------------------+--------+----------+---------+--------+
|  ID|               PLD|NumHosts|PLDisHost?|OutDegree|InDegree|
+----+------------------+--------+----------+---------+--------+
|  26|            abb.nic|    null|      null|        2|       3|
|  29|abbott.corelabora...|    null|      null|       34|      40|
| 474|     ac.americancars|    null|      null|     null|       3|
| 964|             ac.cmt|    null|      null|        1|    null|
|1677|          ac.insight|    null|      null|        7|       1|
+----+------------------+--------+----------+---------+--------+
only showing top 5 rows
DataFrame[ID: string, PLD: string, NumHosts: bigint, PLDisHost?: boolean, OutDegree: big
int, InDegree: bigint]
```

FINISHED

```
%pyspark

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()
```

```
+----+------------------+--------+----------+---------+--------+--------+
|  ID|               PLD|NumHosts|PLDisHost?|OutDegree|InDegree|Crawled?|
+----+------------------+--------+----------+---------+--------+--------+
|  26|            abb.nic|    null|      null|        2|       3|    true|
|  29|abbott.corelabora...|    null|      null|       34|      40|    true|
| 474|     ac.americancars|    null|      null|     null|       3|   false|
| 964|             ac.cmt|    null|      null|        1|    null|    true|
|1677|          ac.insight|    null|      null|        7|       1|    true|
+----+------------------+--------+----------+---------+--------+--------+
only showing top 5 rows
DataFrame[ID: string, PLD: string, NumHosts: bigint, PLDisHost?: boolean, OutDegree: big
int, InDegree: bigint, Crawled?: boolean]
```

FINISHED

```
%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
    presorting (but we don't really need to since this is only 91Mx91M)
pld_df_joined5=pld_df_joined4.join(pr_df, pr_df.host_rev==pld_df_joined4.PLD,
    "leftOuter").drop("#hc_pos").drop("#pr_pos").drop("host_rev")\
```

```
                                .withColumnRenamed("#hc_val","HarmonicCentrality"
```

```
+----+--------------------+--------+----------+---------+--------+--------+-------------
-----+--------------------+
|  ID|                 PLD|NumHosts|PLDisHost?|OutDegree|InDegree|Crawled?|HarmonicCentr
ality|            PageRank|
+----+--------------------+--------+----------+---------+--------+--------+-------------
-----+--------------------+
| 120|             abc.web|    null|      null|     null|       1|   false|          100
15440|3.78405976859536e-08|
| 311|             ac.8411|    null|      null|     null|       1|   false|           90
82498|4.76481484534919e-09|
| 713|              ac.bgc|    null|      null|     null|       1|   false|           92
37769|4.90517712841288e-09|
| 871|          ac.casinos|    null|      null|        2|       1|    true|         7839
579.5|7.68640254732439e-09|
|1014|ac.cosmopolitanun...|    null|      null|     null|      18|   false|          126
15973|5.85933334251156e-09|
+----+--------------------+--------+----------+---------+--------+--------+-------------
-----+--------------------+
```

```
%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
```

RUNNING 29%

Started 39 minutes ago.

```
%pyspark
```

FINISHED