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 - 4 October 2017

import boto
from pyspark.sql.types import *

#LIMIT=1000000 # TODO - remove temporary limit to run full summaries!

# 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)
```

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

```
%pyspark                                                    FINISHED

# Next import the PLD edges as a DataFrame
pld_edges_schema=StructType([StructField("src", StringType(), False), StructField
    ("dst", StringType(), 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: k.split()) # By default, splits on
    whitespace, which is what we want
```

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

```
%pyspark

# 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) #.limit(LIMIT*10) # TODO - remove temporary limit
    to run full summaries!
```

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", 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
```

READY

```
+-------+--------+-------+-------------------+-------------+
|#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 #--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()
print(out_degrees.rdd.lookup("846558"))
```

```
[1]
[156]
```

```
%pyspark                                                                RUNNING 0%

# 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())
#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"))

# Next, broadcast this map so it's available on all the slave nodes - this seems to
```

Started 12 minutes ago.

```
%pyspark                                                                PENDING
```

```
# 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:
        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:
        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"))
```

```
%pyspark                                                                PENDING

# Now count the number of hosts per PLD in a scalable way, and create another
    dictionary
# Takes 5mins for first 10M rows -> approx 8 hours for all 1.3B rows?
count_table=host_df.drop('hostid').rdd.map(lambda x: (convert_hostname(x['host']),1
    )).reduceByKey(lambda x,y: x+y).collectAsMap()
bool_table=host_df.drop('hostid').rdd.map(lambda x: (x['host'], is_a_pld(x['host']
    ))).filter(lambda x: x[1]==True).collectAsMap()
print(count_table['aaa.aaa'])
print(bool_table['aaa.aaa'])
print(count_table['ERROR']) # Should be zero once we've loaded all the PLDs!
```

```
        at py4j.reflection.MethodInvoker.invoke(MethodInvoker.java:244)
        at py4j.reflection.ReflectionEngine.invoke(ReflectionEngine.java:357)
        at py4j.Gateway.invoke(Gateway.java:280)
        at py4j.commands.AbstractCommand.invokeMethod(AbstractCommand.java:132)
        at py4j.commands.CallCommand.execute(CallCommand.java:79)
        at py4j.GatewayConnection.run(GatewayConnection.java:214)
        at java.lang.Thread.run(Thread.java:748)
Caused by: java.lang.IllegalArgumentException: Size exceeds Integer.MAX_VALUE
        at sun.nio.ch.FileChannelImpl.map(FileChannelImpl.java:869)
        at org.apache.spark.storage.DiskStore$$anonfun$getBytes$4.apply(DiskStore.scala:
125)
        at org.apache.spark.storage.DiskStore$$anonfun$getBytes$4.apply(DiskStore.scala:
124)
        at org.apache.spark.util.Utils$.tryWithSafeFinally(Utils.scala:1337)
        at org.apache.spark.storage.DiskStore.getBytes(DiskStore.scala:126)
        at org.apache.spark.storage.BlockManager.getLocalValues(BlockManager.scala:520)
        at org.apache.spark.storage.BlockManager.get(BlockManager.scala:693)
        at org.apache.spark.storage.BlockManager.getOrElseUpdate(BlockManager.scala:753)
```

%pyspark READY

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

# 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)
```

```
+------+-------------------+--------+----------+--------+---------+--------+
|    ID|                PLD|NumHosts|PLDisHost?|InDegree|OutDegree|Crawled?|
+------+-------------------+--------+----------+--------+---------+--------+
|846558|             au.com|  813989|      true|      12|        1|    true|
|600347|          at.ibooked|  275515|      true|       1|       28|    true|
|948740|     au.com.blogspot|  126110|      true|    1299|  1124067|    true|
|114523|     ar.com.blogspot|   97052|      true|    4044|   462744|    true|
|482699|      at.co.blogspot|   29811|      true|    2306|   270569|    true|
|723578|            at.radio|   24659|      true|      49|     1188|    true|
|723512|            at.radio|   24659|      true|      19|       35|    true|
|739424|       at.safedomain|   15202|      true|      22|        3|    true|
| 69342|              am.do|   10416|      true|      60|    31745|    true|
| 15549|         ae.blogspot|    9702|      true|      61|   103731|    true|
| 50431|              ai.id|    7574|      true|       0|     1907|    true|
|739460|         at.safesite|    7565|      true|       5|        4|    true|
|865746|   au.com.adelaidebd|    4978|      true|       3|      154|    true|
|794268|   at.topdestination|    4935|      true|       0|       29|    true|
|  1426|              ac.ae|    4106|      true|       0|      564|    true|
```

```
%pyspark                                                                    READY

# 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_joined2=pld_df_joined.join(pr_df, pr_df.host_rev==pld_df_joined.PLD,
    "leftOuter").drop("#hc_pos").drop("#pr_pos").drop("host_rev").withColumnRenamed
    ("#hc_val","HarmonicCentrality").withColumnRenamed("#pr_val","PageRank")
```

```
+----+-------------------+--------+----------+--------+---------+--------+-------------
-----+--------------------+
|  ID|                PLD|NumHosts|PLDisHost?|InDegree|OutDegree|Crawled?|HarmonicCentr
ality|            PageRank|
+----+-------------------+--------+----------+--------+---------+--------+-------------
-----+--------------------+
| 120|             abc.web|       1|     false|       0|        0|   false|          100
15440|3.78405976859536e-08|
| 311|              ac.8411|       1|     false|       1|        0|   false|           90
82498|4.76481484534919e-09|
| 713|              ac.bgc|       1|     false|       0|        0|   false|           92
37769|4.90517712841288e-09|
| 871|          ac.casinos|       1|      true|       0|        2|    true|         7839
579.5|7.68640254732439e-09|
|1014|ac.cosmopolitanun...|       1|      true|       1|        0|   false|          126
15973|5.85933334251156e-09|
|1089|             ac.dibru|       1|      true|       0|        0|   false|          120
12666|4.49359706049864e-09|
```

```
%pyspark                                                                    READY

# Save final table to S3 in compressed CSV format
outputURI="s3://billsdata.net/CommonCrawl/domain_summaries/"
```

```
codec="org.apache.hadoop.io.compress.GzipCodec"
```

```
%pyspark
```

READY