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
```

FINISHED

```
+-------+--------+-------+-------------------+--------------+
|#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
```

RUNNING 0%

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

# First, let's use the PLD edges dataframe to compute in and out-degrees for each PLD
    ID, and store as dictionaries.
# Note: we could use GraphFrames for this but it's a pain to get GraphFrames working
    in a Zeppelin notebook!
out_degrees=dict(pld_edges_df.groupBy("src").count().collect())
```

Started a minute ago.

```
%pyspark
```

PENDING

```
# 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
pld_lookup_table=dict(pld_df.rdd.map(lambda x: (x['PLD'], x['ID'])).collect())
print(pld_lookup_table["aaa.aaa"])

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

```
%pyspark
```

PENDING

```
# 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:
        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 '.'.join(parts[0:4]) in pld_lookup_table):
            return '.'.join(parts[0:4])
        if (len(parts)>3 and '.'.join(parts[0:3]) in pld_lookup_table):
            return '.'.join(parts[0:3])
        if (len(parts)>2 and '.'.join(parts[0:2]) in pld_lookup_table):
            return '.'.join(parts[0:2])
        if (len(parts)>1 and '.'.join(parts[0:1]) in pld_lookup_table):
            return '.'.join(parts[0:1])
        return "ERROR" # Couldn't find a corresponding PLD - this should never happen!
    except:
        return "ERROR"

# Returns a Boolean to say whether PLD is a hostname in itself
def is_hostname_a_pld(hostname):
    if hostname in pld_lookup_table:
        return True
    else:
        return False

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

```
aaa.aaa
True
```

```
%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_hostname_a_pld
    (x['host']))).filter(lambda x: x[1]==True).collectAsMap()
```

```
6
True
7180422
```

```
%pyspark                                                                    PENDING

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"))\
```

```
|293592|ar.com.publicargr...|    1377|      true|       1|      14|    true|
|308619|        ar.com.ruoff|    1370|     false|       0|       0|   false|
| 83250|        ar.com.a-e-a|    1363|     false|       0|       0|   false|
|244970|        ar.com.lyros|    1356|     false|       0|       0|   false|
|818378|          at.webnode|    1347|      true|     372|     988|    true|
|385848|           asia.6eha|    1344|     false|       0|       0|   false|
|324057|        ar.com.speis|    1337|     false|       0|       0|   false|
|293172|         ar.com.psdi|    1323|     false|       0|       0|   false|
|276548|        ar.com.ostiz|    1322|     false|       0|       0|   false|
|357313|         ar.com.zonq|    1294|     false|       0|       0|   false|
|349896|        ar.com.vmnet|    1292|     false|       0|       0|   false|
|192232|        ar.com.fsgsa|    1248|     false|       0|       0|   false|
| 75471|       am.schoolsite|    1243|      true|       5|      95|    true|
| 51928|               ai.nl|    1241|      true|       0|       0|   false|
|186154|        ar.com.fimct|    1237|     false|       0|       0|   false|
+------+--------------------+--------+----------+--------+---------+--------+
only showing top 100 rows
DataFrame[ID: string, PLD: string, NumHosts: int, PLDisHost?: boolean, InDegree: int, Ou
```

```
%pyspark
```

PENDING

```
# 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")
```

```
|1089|            ac.aibru|       1|      true|          12012666|4.49359706049864e-09|
|1435|          ac.gorilla|       1|      true|           9114256|4.50619088846452e-09|
|2476|          ac.philter|       1|      true|          10434785|4.44625601709852e-09|
|3138|              ac.ula|       1|     false|          10046531| 4.5521807953781e-09|
|3145|         ac.umedalen|       2|      true|          12093009|4.69402844088012e-09|
|3373|              ac.yui|       2|      true|          12105217|5.09160908953513e-09|
|3484|   academy.alphastar|       1|      true|           9816919|4.91209890117111e-09|
|3768|    academy.cirulnik|       1|      true|         7967335.5|6.12220241367047e-09|
|3787|       academy.cocoa|       1|      true|          10196119|6.02409896712952e-09|
|3882|academy.dental-coach|       1|      true|          10981495|4.43152551513855e-09|
|4157|         academy.ger|       1|      true|           8087047|1.37105166301547e-08|
|4410|academy.investmen...|       2|      true|          12299180|1.31584108265683e-08|
|4769|     academy.newtown|       1|      true|          10334425|8.54845369040491e-09|
|5224|        academy.talk|       2|      true|          12012667|5.30628234199031e-09|
|6064|accountant.buy-mo...|       1|      true|          10192661|5.29716249291459e-09|
+----+--------------------+--------+----------+------------------+--------------------+
only showing top 20 rows
DataFrame[ID: string, PLD: string, NumHosts: int, PLDisHost?: boolean, HarmonicCentralit
```

```
%pyspark                                                              PENDING

# Save final table to S3 in compressed CSV format
outputURI="s3://billsdata.net/CommonCrawl/domain_summaries/"
codec="org.apache.hadoop.io.compress.GzipCodec"
pld_df_joined2.coalesce(1).write.format('com.databricks.spark.csv').options(header
```

```
%pyspark                                                              FINISHED
```