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 - 6 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", String
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 i
pld_df=temp_pld.toDF(pld_schema).limit(LIMIT).repartition(16)
pld_df.show(3)
pld_df.cache()
# Should have 91M domains
#print(pld_df.count())

+---+----------------+
| ID|            PLD|
+---+----------------+
|  4|      aaa.aaaaaa|
| 20|aarp.disruptaging|
| 36|       abbott.it|
+---+----------------+
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", LongType(), False), StructField("dst", L
pld_edges_txt=sc.textFile("s3://commoncrawl/projects/hyperlinkgraph/cc-main-2017-may-ju
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).repartition(16) # TODO
pld_edges_df.show(3)
pld_edges_df.cache()
```

```
+---+--------+
|src|     dst|
+---+--------+
| 24| 9258593|
| 27|  416120|
| 27|17255607|
+---+--------+
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",
host_txt=sc.textFile("s3://commoncrawl/projects/hyperlinkgraph/cc-main-2017-may-jun-jul/
temp_host = host_txt.map(lambda k: k.split()) # By default, splits on whitespace, which
host_df=temp_host.toDF(host_schema).limit(LIMIT).repartition(16) # TODO - remove tempor
host_df.show(3)
host_df.cache()
# Should have 1.3B hosts
#print(host_df.count())
```

```
+------+---------------+
|hostid|           host|
+------+---------------+
|     4| aaa.aaa.aaa.aaa|
|    20|aaa.ldap-primary|
|    36|       abb.brand|
+------+---------------+
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)])
pr_txt=sc.textFile("s3://commoncrawl/projects/hyperlinkgraph/cc-main-2017-may-jun-jul/d
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(LIMI
pr_df.show(3)
pr_df.cache()
#pr_df.count() # Should be 91M
```

```
+-------+--------+-------+--------------------+-------------+
|#hc_pos| #hc_val|#pr_pos|             #pr_val|     host_rev|
+-------+--------+-------+--------------------+-------------+
|      5|21234402|      5| 0.00706371663388533|   com.google|
|     23|17323658|     47|0.000801430611613196|    com.vimeo|
|     44|16646934|    185|0.000141243073946399|org.wikimedia|
+-------+--------+-------+--------------------+-------------+
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()
```

```
16
16
16
16
```

%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 nee

# 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()
out_degrees.show(3)
in_degrees.show(3)
#print(out_degrees.rdd.lookup(846558))
#print(in_degrees.rdd.lookup(846558))
```

```
+-----+-----+
|  src|count|
+-----+-----+
|   29|   34|
| 2214|  127|
|13518|   22|
+-----+-----+
only showing top 3 rows
+--------+-----+
|     dst|count|
+--------+-----+
|      29|    8|
|27481043|    2|
|66742183|    2|
+--------+-----+
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())
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 bre
#pld_map=sc.broadcast(pld_lookup_table)
```

2

%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:
        return True
    else:
        return False

# Define a function to do the hostname->pld conversion, if the pld exists in our diction
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"))
print(is_a_pld("aaa.aaa"))
```

```
aaa.aaa
True
```

%pyspark   FINISHED

```
# 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_df2.drop('hostid').rdd.map(lambda x: (convert_hostname(x['host']),1)).
bool_table=host_df2.drop('hostid').rdd.map(lambda x: (x['host'], is_a_pld(x['host']))).
print(count_table['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
```

```
6
True
7180422
```

%pyspark   FINISHED

```
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())
```

```python
# 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 whethe
#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)
pld_df_joined.sort("NumHosts", ascending=False).show(100)
pld_df_joined.cache()
```

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

%pyspark   FINISHED

```python
# 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()
```

```
+----+------------------+--------+---------+---------+--------+
|  ID|               PLD|NumHosts|PLDisHost?|OutDegree|InDegree|
+----+------------------+--------+---------+---------+--------+
|  26|           abb.nic|       3|     true|        2|    null|
|  29|abbott.corelabora...|     2|     true|       34|       8|
| 474|     ac.americancars|     1|     true|     null|    null|
| 964|            ac.cmt|       1|    false|        1|    null|
|1677|        ac.insight|       1|     true|        7|    null|
+----+------------------+--------+---------+---------+--------+
only showing top 5 rows
DataFrame[ID: string, PLD: string, NumHosts: int, PLDisHost?: boolean, OutDegree: bigint
, InDegree: bigint]
```

%pyspark   FINISHED

```
# 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|       3|     true|        2|    null|    true|
|  29|abbott.corelabora...|     2|     true|       34|       8|    true|
| 474|     ac.americancars|     1|     true|     null|    null|   false|
| 964|            ac.cmt|       1|    false|        1|    null|    true|
|1677|        ac.insight|       1|     true|        7|    null|    true|
+----+------------------+--------+---------+---------+--------+--------+
only showing top 5 rows
DataFrame[ID: string, PLD: string, NumHosts: int, PLDisHost?: boolean, OutDegree: bigint
, InDegree: bigint, Crawled?: boolean]
```

%pyspark   FINISHED

```
# 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, "leftOuter
                        .withColumnRenamed("#hc_val","HarmonicCentrality").withColu
pld_df_joined4.unpersist()
pld_df_joined5.show(5)
pld_df_joined5.cache()
```

```
+----+------------------+--------+---------+---------+--------+--------+-------------
-----+--------+
|  ID|               PLD|NumHosts|PLDisHost?|OutDegree|InDegree|Crawled?|HarmonicCentr
ality|PageRank|
+----+------------------+--------+---------+---------+--------+--------+-------------
-----+--------+
| 120|           abc.web|       1|    false|     null|    null|   false|
null|    null|
| 311|          ac.8411|       1|    false|     null|       1|   false|
null|    null|
| 713|            ac.bgc|       1|    false|     null|    null|   false|
null|    null|
| 871|         ac.casinos|       1|     true|        2|    null|    true|
null|    null|
|1014|ac.cosmopolitanun...|       1|     true|     null|    null|   false|
null|    null|
+----+------------------+--------+---------+---------+--------+--------+-------------
-----+--------+
```

%pyspark   FINISHED

```
# Save final table to S3 in compressed CSV format
outputURI="s3://billsdata.net/CommonCrawl/domain_summaries2/"
codec="org.apache.hadoop.io.compress.GzipCodec"
pld_df_joined5.coalesce(1).write.format('com.databricks.spark.csv').options(header='tru
```

%pyspark   READY