

Paul 5 - faster do...

%pyspark

FINISHED

```
# Zeppelin notebook to create domain summaries based on the May/Jun/Jul 2017 CommonCrawl
# as per description here: http://commoncrawl.org/2017/08/webgraph-2017-may-june-july/
# PJ - 18 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", Strin
#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.a|
|  2|aaa.a|
+---+-----+
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)
nld_edges_df.cache()
```

```
+---+-----+
```

```
|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"
#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) #.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).repartition(64)
host_df.show(3)
host_df.cache()
#print(host_df.count()) # Should have 1.3B hosts
```

```
+-----+-----+
```

```
|hostid|      host|
```

```
+-----+-----+
```

```
| 17|  aaa.bzzzz|
```

```
| 81|  abc.2020|
```

```
|145|abc.tenmien|
```

```
+-----+-----+
```

```
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"
(), False), 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
```

```
#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/domaining
#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()
```

286

384

64

384

```
%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(fin_degrees_rdd.lookup(846558))
```

```
+-----+-----+
|      src|count|
```

```
+-----+-----+
|13208927|    3|
```

```
|13209857|    1|
```

```
|13210113|   12|
```

```
+-----+-----+
```

```
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 count)
# 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
#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.rdd.collect(): # limit(10000000) # TODO: Still bad (and exceeds spark limit)
    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 implementation - can't figure out how to properly use
#tmp=pld_df.rdd.map(lambda x: pld_bf.add(x['PLD'])) # Very bad - pld_bf gets copied to each node
#tmp=pld_df.rdd.map(lambda x: (pld_bf.add(x['PLD']), pld_bf)).reduce(lambda x,y: x[1].update(y[1]))
#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 to
# 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
90751305
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 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
print(convert_hostname("aaa.aaa"))
```

```
aaa.aaa
```

```
True
```

```
%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
# (An attempt to collectAsMap at the end results in java Integer.MAX_VALUE or memory error)
count_rdd=host_df.drop('hostid').rdd.map(lambda x: (convert_hostname(x['host']),1)).reduceByKey(lambda x,y: x+y)
bool_rdd=host_df.drop('hostid').rdd.map(lambda x: (x['host'], is_a_pld(x['host']))).filter(lambda x: x[1])
```

```
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[0]=='aaa.aaa').collect())
print(len(count_rdd.filter(lambda x: x[0]=='ERROR').collect())) # Should be zero once we
```

```
[(u'jp.kids-labo', 1), (u'org.g7fyp39crg0065nzu94cq1m4e35k793os', 1), (u'org.palletizer', 1)]
```

```
[(u'cn.bjbworld', True), (u'cn.bjcnw', True), (u'cn.bjcxbz36', True)]
```

```
90839924
```

```
89276336
```

```
[(u'aaa.aaa', 6)]
```

```
1
```

```
%pyspark
```

FINISHED

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

```
# The following code works well when the data is small enough to collect into a python dictionary
# 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)

# Convert the result RDDs to dataframes, ready for joining
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("PLDtest",
bool_df=bool_rdd.toDF(boolschema)
bool_df.show(3)

# Join these new dataframes with the original dataframe (using fast equi-joins)
pld_df2=pld_df.join(count_df, count_df.PLD2==pld_df.PLD, "leftOuter").drop("PLD2")
bool_test=when(col("PLDtest").isNull(), lit(False)).otherwise(lit(True))
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)
122090711|      com.rcz| 1360713|      true|
169702382|      net.eachtong| 1340854|      true|
17182925|      cn.fjmeilun| 1338282|      true|
17147207|      cn.dq2918| 1329916|      false|
17358421|      cn.jinwolf| 1329710|      false|
132134194|      com.mayienglish| 1325006|      true|
17332379|      cn.hzhemul| 1320528|      false|
16920418|      cn.cnzhizhen| 1317289|      false|
17201787|      cn.gdlswy| 1307459|      true|
16917185|      cn.cn-oy| 1306957|      true|
17130482|      cn.datongcompany| 1306765|      false|
17143733|      cn.dmmarket| 1304280|      false|
17288576|      cn.hailinfengde| 1300478|      false|
17148929|      cn.dsental| 1299703|      true|
17205524|      cn.giantcn| 1297858|      true|
+-----+-----+-----+-----+
only showing top 100 rows
DataFrame[ID: string, PLD: string, numHosts: bigint, pldIsHostFlag: 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, "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)
```

```

+---+-----+-----+-----+-----+-----+
| ID|          PLD|numHosts|pldIsHostFlag|pldLinksOut|pldLinksIn|
+---+-----+-----+-----+-----+-----+
| 26|      abb.nic|      3|      true|      2|      3|
| 29|abbott.corelabora...|      2|      true|     34|     40|
| 474|    ac.americancars|      1|      true|    null|      3|
| 964|          ac.cmt|      1|     false|      1|    null|
|1677|    ac.insight|      1|      true|      7|      1|
+---+-----+-----+-----+-----+-----+

```

only showing top 5 rows

```
DataFrame[ID: string, PLD: string, numHosts: bigint, pldIsHostFlag: boolean, pldLinksOut:
: bigint, pldLinksIn: bigint]
```

```
%pyspark
```

FINISHED

```

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

```

```

+---+-----+-----+-----+-----+-----+-----+
| ID|          PLD|numHosts|pldIsHostFlag|pldLinksOut|pldLinksIn|wasCrawledFlag|
+---+-----+-----+-----+-----+-----+-----+
| 26|      abb.nic|      3|      true|      2|      3|      true|
| 29|abbott.corelabora...|      2|      true|     34|     40|      true|
| 474|    ac.americancars|      1|      true|    null|      3|     false|
| 964|          ac.cmt|      1|     false|      1|    null|      true|
|1677|    ac.insight|      1|      true|      7|      1|      true|
+---+-----+-----+-----+-----+-----+-----+

```

only showing top 5 rows

```
DataFrame[ID: string, PLD: string, numHosts: bigint, pldIsHostFlag: boolean, pldLinksOut:
: bigint, pldLinksIn: bigint, wasCrawledFlag: 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 presorting
pld_df_joined5=pld_df_joined4.join(pr_df, pr_df.host_rev==pld_df_joined4.PLD, "leftOuter")
                                .withColumnRenamed("#hc_val", "HarmonicCentrality").withColumnRenamed("#hc_pos", "hc_pos").withColumnRenamed("pr_rev", "pr_rev")
pld_df_joined4.unpersist()
pld_df_joined5.show(5)
pld_df_joined5.cache()

```



```

+----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
| ID|          PLD|numHosts|pldIsHostFlag|pldLinksOut|pldLinksIn|wasCrawledFlag|
hc_pos|HarmonicCentrality| pr_pos|          PageRank|
+----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
| 120|          abc.web|          1|          false|          null|          1|          false|
38991028|          10015440| 1478887|3.78405976859536e-08|
| 311|          ac.8411|          1|          false|          null|          1|          false|
69935624|          9082498|36930613|4.76481484534919e-09|
| 713|          ac.bgc|          1|          false|          null|          1|          false|
63729192|          9237769|32796120|4.90517712841288e-09|
| 871|          ac.casinos|          1|          true|          2|          1|          true|
78150671|          7839579.5|12855010|7.68640254732439e-09|
|1014|ac.cosmopolitanun...|          1|          true|          null|          18|          false|
1636792|          12615973|20034471|5.85933334251156e-09|
+----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+

```

`%pyspark`

FINISHED

```

# Save final table to S3 in compressed CSV format, broken into smaller files
outputURI="s3://billsdata.net/CommonCrawl/domain_summaries5/"
codec="org.apache.hadoop.io.compress.GzipCodec"
#pld_df_joined5.coalesce(1).write.format('com.databricks.spark.csv').options(header='true')
pld_df_joined5.write.save(outputURI)

```

`%pyspark`

FINISHED

```

# Clean up some objects to free memory if needed!
count_rdd.unpersist()
count_df.unpersist()
bool_rdd.unpersist()
bool_df.unpersist()
in_degrees.unpersist()
out_degrees.unpersist()
pld_edges_df.unpersist()
pld_bf_distrib.unpersist()

# Encourage a garbage collection!
import gc
collected = gc.collect()
print "Garbage collector: collected %d objects." % collected

```

Garbage collector: collected 312 objects.

`%pyspark`

FINISHED

```

from pyspark.sql.functions import udf, col, when, lit
def reverse_domain(domain):

```

```

    return '.'.join(reversed(domain.split('.')))
print(reverse_domain("com.facebook.abc"))

udf_reverse_domain = udf(reverse_domain, StringType())
summary_df2=pld_df_joined5.drop("ID").withColumnRenamed("PLD","PLD_rev").withColumn("payLevel",
("PageRank"))
summary_df2.show(3)
summary_df2.cache()

```

```
abc.facebook.com
```

```

+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+
|numHosts|pldIsHostFlag|pldLinksOut|pldLinksIn|wasCrawledFlag|  hc_pos|  pr_pos|payLevel
Domain|
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+
|      1|      false|      null|      1|      false|38991028| 1478887|      w
eb.abcl
|      1|      false|      null|      1|      false|69935624|36930613|      8
411.accl
|      1|      false|      null|      1|      false|63729192|32796120|
bgc.accl
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+

```

```
only showing top 3 rows
```

```
DataFrame[numHosts: bigint, pldIsHostFlag: boolean, pldLinksOut: bigint, pldLinksIn: big
int, wasCrawledFlag: boolean, hc_pos: string, pr_pos: string, payLevelDomain: string]
```

```
%pyspark
```

FINISHED

```

# Convert HC and PR positions to a rank (with 3dp, out of 100, where 100 is the most pos)
COUNT=int(91034128)
def rank(pos):
    return "{0:.3f}".format(((COUNT-float(pos))/COUNT)*100)
print(rank(1478887))

udf_rank=udf(rank, StringType())

summary_df3=summary_df2.withColumn("hcRank", udf_rank("hc_pos")).drop("hc_pos").withColu
summary_df2.unpersist()
summary_df3.show(3)
summary_df3.cache()

```

98.375

```

+-----+-----+-----+-----+-----+-----+-----+-----+
--+
|numHosts|pldIsHostFlag|pldLinksOut|pldLinksIn|wasCrawledFlag|payLevelDomain|hcRank|prRa
nkl
+-----+-----+-----+-----+-----+-----+-----+-----+
--+
|      1|      false|      null|      1|      false|      web.abc|57.169|98.3
75|
|      1|      false|      null|      1|      false|      8411.ac|23.176|59.4
32|
|      1|      false|      null|      1|      false|      bgc.ac|29.994|63.9
74|
+-----+-----+-----+-----+-----+-----+-----+-----+
--+
only showing top 3 rows
DataFrame[numHosts: bigint, pldIsHostFlag: boolean, pldLinksOut: bigint, pldLinksIn: big
int, wasCrawledFlag: boolean, payLevelDomain: string, hcRank: string, prRank: string]

```

%pyspark

FINISHED

```

# Re-order the columns for final output
summary_df4=summary_df3.select("payLevelDomain","numHosts","pldIsHostFlag","pldLinksIn"
summary_df3.unpersist()
summary_df4.show(3)
summary_df4.cache()

```

```

+-----+-----+-----+-----+-----+-----+-----+-----+
--+
|payLevelDomain|numHosts|pldIsHostFlag|pldLinksIn|pldLinksOut|wasCrawledFlag|hcRank|prRa
nkl
+-----+-----+-----+-----+-----+-----+-----+-----+
--+
|      web.abc|      1|      false|      1|      null|      false|57.169|98.3
75|
|      8411.ac|      1|      false|      1|      null|      false|23.176|59.4
32|
|      bgc.ac|      1|      false|      1|      null|      false|29.994|63.9
74|
+-----+-----+-----+-----+-----+-----+-----+-----+
--+
only showing top 3 rows
DataFrame[payLevelDomain: string, numHosts: bigint, pldIsHostFlag: boolean, pldLinksIn:
bigint, pldLinksOut: bigint, wasCrawledFlag: boolean, hcRank: string, prRank: string]

```

%pyspark

FINISHED

```

# Let's take a look at the final table before saving it
summary_df4.sort("numHosts", ascending=False).show(100)

```

```

+-----+-----+-----+-----+-----+-----+-----+
--++-----+
|      payLevelDomain|numHosts|pldIsHostFlag|pldLinksIn|pldLinksOut|wasCrawledFlag| hcRa
nkl prRank|
+-----+-----+-----+-----+-----+-----+-----+
--++-----+
|      533b8.com|24285866|      true|      11|      25|      true| 91.0
60| 54.579|
|      composesite.com|15089842|      true|      35|      16|      true| 77.6
93| 97.294|
|getrichandgivebac...|14513586|      true|      4|      91|      true| 81.3
41| 50.058|
|      bumpbabyme.com|13727426|      true|      22|      27|      true| 94.1
14| 74.273|
|      ourssite.com|12917619|      true|      50|      15|      true| 76.0
82| 97.610|
|      blogspot.com|12717586|      true| 1588452| 14903607|      true|100.0
00|100 000|

```

```
%pyspark
```

FINISHED

```

# Save final table to S3 in compressed CSV format, broken into smaller files
outputURI="s3://billsdata.net/CommonCrawl/domain_summaries7/"
codec="org.apache.hadoop.io.compress.GzipCodec"
summary_df4.coalesce(10).write.format('com.databricks.spark.csv').options(header='true'

```

```
%pyspark
```

FINISHED