201708 common c...

FINISHED

Playing with WAT files

See Analysing Petabytes of Websites (http: //tech.marksblogg.com/petabytes-of-website-data-spark-emr.html)…

What this notebook does:

A first exploration of WAT files, and reproduces some exercises from the blog above.

Took 0 sec. Last updated by anonymous at October 14 2017, 10:42:02 AM.

READY

```
%pyspark

import boto
from boto.s3.key import Key
from gzipstream import GzipStreamFile
from pyspark.sql.types import *
import warc

import json

watlist = sc.textFile("s3://commoncrawl/crawl-data/CC-MAIN-2017-04/wat.paths.gz")
watlist.cache()
watlist.count()
```

```
57800
```

READY

```
%pyspark

for f in watlist.take(10):
    print(f)

conn = boto.connect_s3(anon=True, host='s3.amazonaws.com')
bucket = conn.get_bucket('commoncrawl')

def unpack(uri):
    key_ = Key(bucket, uri)
    file_ = warc.WARCFile(fileobj=GzipStreamFile(key_))
    return file_
```

```
crawl-data/CC-MAIN-2017-04/segments/1484560279169.4/wat/CC-MAIN-20170116095119-00000-ip-10-
171-10-70.ec2.internal.warc.wat.gz
crawl-data/CC-MAIN-2017-04/segments/1484560279169.4/wat/CC-MAIN-20170116095119-00001-ip-10-
171-10-70.ec2.internal.warc.wat.gz
crawl-data/CC-MAIN-2017-04/segments/1484560279169.4/wat/CC-MAIN-20170116095119-00002-ip-10-
171-10-70.ec2.internal.warc.wat.gz
crawl-data/CC-MAIN-2017-04/segments/1484560279169.4/wat/CC-MAIN-20170116095119-00003-ip-10-
171-10-70.ec2.internal.warc.wat.gz
crawl-data/CC-MAIN-2017-04/segments/1484560279169.4/wat/CC-MAIN-20170116095119-00004-ip-10-
171-10-70.ec2.internal.warc.wat.gz
crawl-data/CC-MAIN-2017-04/segments/1484560279169.4/wat/CC-MAIN-20170116095119-00005-ip-10-
171-10-70.ec2.internal.warc.wat.gz
```

```
crawl-data/CC-MAIN-2017-04/segments/1484560279169.4/wat/CC-MAIN-20170116095119-00006-ip-10-
171-10-70.ec2.internal.warc.wat.gz
crawl-data/CC-MAIN-2017-04/segments/1484560279169.4/wat/CC-MAIN-20170116095119-00007-ip-10-
171-10-70.ec2.internal.warc.wat.gz
crawl-data/CC-MAIN-2017-04/segments/1484560279169.4/wat/CC-MAIN-20170116095119-00008-ip-10-
```

READY

```
%pyspark

def mapper(uri):
    file = unpack(uri)
    return [record['Content-Type'] for record in file]

nfiles = 1
files = sc.parallelize(watlist.take(nfiles))

ct = files.flatMap(mapper)
ct.cache()
print(ct.count())
print(ct.countByValue())
ct.unpersist()
```

```
162875
defaultdict(<type 'int'>, {'application/warc-fields': 1, 'application/json': 162874})
PythonRDD[15] at RDD at PythonRDD.scala:48
```

READY

This took 6 mins on 6 nodes for 100 files – so ~20 seconds/node/file. So for 57800 files, about 300 hours/node.
This corresponds to the fact that WAT files are about a third the size of WARC files.

We can do this a little more efficiently using `mapPartionsWithIndex()`:

READY

```
%pyspark

def mapper(id_, iterator):
    for uri in iterator:
        file = unpack(uri)
        for record in file:
            try:
                yield record['Content-Type']
            except:
                yield None

nfiles = 1
files = sc.parallelize(watlist.take(nfiles))

ct = files.mapPartitionsWithIndex(mapper)
ct.cache()
print(ct.count())
print(ct.countByValue())
ct.unpersist()
```

```
162875
defaultdict(<type 'int'>, {'application/warc-fields': 1, 'application/json': 162874})
PythonRDD[20] at RDD at PythonRDD.scala:48
```

Here's a toy example with `mapPartionsWithIndex()`: READY

```
%pyspark                                                                 READY

from __future__ import print_function

print(nfiles, "files")

def f(splitIndex, iterator): yield splitIndex
print(files.mapPartitionsWithIndex(f).collect())
```

```
1 files
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24,
 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 4
7, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69,
 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 9
2, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111,
 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129,
 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147,
 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165,
 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183,
 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201,
 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219,
 220, 221, 222, 223]
```

Let's inspect the WARC-fields first (one per file): READY

```
%pyspark                                                                 READY

def mapper(id_, iterator):
    for uri in iterator:
        file = unpack(uri)
        for record in file:
            try:
                if record['Content-Type'] == 'application/warc-fields':
                    record = json.loads(record.payload.read())
                    yield record
            except:
                yield None

files.mapPartitionsWithIndex(mapper).collect()
```

```
[None]
```

And the json records: first, let's reproduce Litwintschik's server count: READY

```
%pyspark                                                                 READY

from __future__ import print_function

def get_servers(id_, iterator):
    for uri in iterator:

        file = unpack(uri)
        for record in file:
            if record['Content-Type'] == 'application/json':
                record = json.loads(record.payload.read())

                try:
                    yield record['Envelope']\
                                ['Payload-Metadata']\
                                ['HTTP-Response-Metadata']\
                                ['Headers']\
                                ['Server'].strip().lower()
                except KeyError:
                    yield None

servers = files.mapPartitionsWithIndex(get_servers)\
               .map(lambda x: (x,1))\
               .reduceByKey(lambda x,y: x+y)
servers.cache()

"""
... or use
servers = files.mapPartitionsWithSplit(get_servers).countByValue()
... which will return a lust
"""

print("Servers:", servers.count())
```

```
Servers: 1840
```

```
%pyspark

from __future__ import print_function

n = 20

top_n = servers.sortBy(lambda x: -x[1]).take(n)
for x in top_n:
    print(x[1], x[0])
```

READY

```
112661 None
10166 apache
8536 nginx
4674 cloudflare-nginx
2330 microsoft-iis/7.5
1750 microsoft-iis/8.5
1258 gse
1113 nginx/1.10.2
886 apache/2.2.15 (centos)
818 nginx/1.6.2
786 nginx/1.10.1
716 apache-coyote/1.1
644 apache/2.4.7 (ubuntu)
574 apache/2
573 microsoft-iis/8.0
537 nginx/1.8.0
514 litespeed
464 microsoft-iis/6.0
```

Let's aggregate out the version numbers:

READY

```
%pyspark                                                                READY

from __future__ import print_function

def stem(str):
    if str==None: return "None"
    try:
        return str.split('/')[0]
    except KeyError:
        return str

n = 20
top_n = servers.map(lambda p: [stem(p[0]), p[1]]).reduceByKey(lambda x, y: x + y).sortBy(l
for x in top_n:
    print(x[1], x[0])
```

```
112661 None
18054 apache
15409 nginx
5508 microsoft-iis
4674 cloudflare-nginx
1258 gse
716 apache-coyote
514 litespeed
197 openresty
165 cowboy
148 ats
119 kayak
113 tengine
110 lighttpd
97 sucuri
91 pws
85 nginx admin
83 arator
```

```
%pyspark                                                                READY
```