CS 8001 Big Data

HW #3: Web Server Log Analysis (10 points + 2 bonus points)

Jing Su

js929@mail.missouri.eud

In this exercise, you will write Python code on Spark that analyzes a data set from NASA Kennedy Space Center WWW server in Florida, which is freely available at http: //ita.ee.lbl.gov/html/contrib/NASA-HTTP.html. Use the first log of the data set in this homework, which was collected from 00:00:00 July 1, 1995 through 23:59:59 July 31, 1995, a total of 31 days.

This exercise consists of 4 steps:

1. Apache Web Server Log file format

2. Sample Analyses on the Web Server Log File

3. Analyzing Web Server Log File

4. Exploring 404 Response Codes See reference here.

Part I (10 points): Run Spark locally with one or more worker threads.

1) A brief description of your Spark environment, your code structure, and the execution process of the code.

Spark Environment: AWS EMR(1 node), Windows 10

Code Structure:

Parsing each log line

Creating RDDs and cleaning data

Analyses log data

Execution Process:

1. Put the log file to HDFS.
2. Use command: spark-submit logmining.py access_log_Jul95

2) Output of your code for the following analysis:

a. Top Ten Error Endpoints. Create a sorted list containing top ten endpoints and the number of times that they were accessed with non-200 return code.

```
Top Ten failed URLs: [(u'/images/NASA-logosmall.gif HTTP/1.0', 21010), (u'/images/KSC-logosmall.gif HTTP/1.0', 12435), (u'/images/MOSAIC-logosmall.gif HTTP/1.0', 6628), (u'/images/USA-logosmall.gif HTTP/1.0', 6577), (u'/images/WORLD-logosmall.gif HTTP/1.0', 6413), (u'/images/ksclogo-medium.gif HTTP/1.0', 5837), (u'/images/launch-logo.gif HTTP/1.0', 4628), (u'/shuttle/countdown/liftoff.html HTTP/1.0', 3509), (u'/shuttle/countdown/ HTTP/1.0', 3345), (u'/shuttle/countdown/images/cdtclock.gif HTTP/1.0', 3251)]
```

b. Number of Unique Hosts.

```
Unique hosts: 81982
```

c. Number of Unique Daily Hosts

```
Unique hosts per day: [(1, 5192), (2, 4859), (3, 7336), (4, 5524), (5, 7383), (6, 7820), (7, 6474), (8, 2898), (9, 2554), (10, 4464), (11, 4927), (12, 5345), (13, 6951), (14, 5297), (15, 3116), (16, 3013), (17, 4943), (18, 4523), (19, 4919), (20, 4729), (21, 4339), (22, 2575), (23, 2635), (24, 4298), (25, 4376), (26, 4296), (27, 4368), (28, 2175)]
```

d. Counting 404 Response Codes. How many 404 records are in the log?

```
Found 10845 404 URLs
```

e. Listing the Top Twenty 404 Response Code Endpoints

```
Top Twenty 404 URLs: [(u'/pub/winvn/readme.txt HTTP/1.0', 667), (u'/pub/winvn/release.txt HTTP/1.0', 547), (u'/history/apollo/apollo-13.html HTTP/1.0', 286), (u'/history/apollo/a-001/a-001-patch-small.gif HTTP/1.0', 230), (u'/shuttle/resources/orbiters/atlantis.gif HTTP/1.0', 230), (u'/://spacelink.msfc.nasa.gov HTTP/1.0', 215), (u'/history/apollo/pad-abort-test-1/pad-abort-test-1-patch-small.gif HTTP/1.0', 215), (u'/images/crawlerway-logo.gif HTTP/1.0', 214), (u'/history/apollo/sa-1/sa-1-patch-small.gif HTTP/1.0', 183), (u'/shuttle/resources/orbiters/discovery.gif HTTP/1.0', 180), (u'/shuttle/missions/sts-68/ksc-upclose.gif HTTP/1.0', 175), (u'/shuttle/missions/sts-71/images/KSC-95EC-0916.txt HTTP/1.0', 168), (u'/elv/DELTA/uncons.htm HTTP/1.0', 163), (u'/history/apollo/publications/sp-350/sp-350.txt~ HTTP/1.0', 140), (u'/shuttle/missions/technology/sts-newsref/stsref-toc.html HTTP/1.0', 107), (u'/shuttle/resources/orbiters/challenger.gif HTTP/1.0', 92), (u'/procurement/procurement.htm HTTP/1.0', 86), (u'/history/apollo-13/apollo-13.html HTTP/1.0', 73), (u'/history/apollo/pad-abort-test-2/pad-abort-test-2-patch-small.gif HTTP/1.0', 71), (u'/shuttle/countdown/video/livevideo.jpeg HTTP/1.0', 68)]
```

3) Execution time of your code in seconds.

```
Execution time: 216.564713001
```

4) Your Python code with appropriate comments.

```python
from __future__ import print_function
import sys
reload(sys)
sys.setdefaultencoding('utf-8')
from operator import add
from pyspark import SparkContext
import re
import time
import datetime
from pyspark.sql import Row

month_map = {'Jan': 1, 'Feb': 2, 'Mar':3, 'Apr':4, 'May':5, 'Jun':6, 'Jul':7,
    'Aug':8,  'Sep': 9, 'Oct':10, 'Nov': 11, 'Dec': 12}

def parse_apache_time(s):
    """ Convert Apache time format into a Python datetime object
    Args:
        s (str): date and time in Apache time format
    Returns:
        datetime: datetime object (ignore timezone for now)
    """
```

```
    return datetime.datetime(int(s[7:11]),
                             month_map[s[3:6]],
                             int(s[0:2]),
                             int(s[12:14]),
                             int(s[15:17]),
                             int(s[18:20]))

def parseApacheLogLine(logline):
    """ Parse a line in the Apache Common Log format
    Args:
        logline (str): a line of text in the Apache Common Log format
    Returns:
        tuple: either a dictionary containing the parts of the Apache Access Log and 1,
               or the original invalid log line and 0
    """
    match = re.search(
    '^(\S+) (\S+) (\S+) \[([\w:/]+\s[+\-]\d{4})\] "(\S*)\s*(.*)\s*(\S*)" (\d{3}) (\S+)',
    logline)
    if match is None:
        return (logline, 0)
    size_field = match.group(9)
    if size_field == '-':
        size = long(0)
    else:
        size = long(match.group(9))
    return (Row(
        host          = match.group(1),
        client_identd = match.group(2),
        user_id       = match.group(3),
        date_time     = parse_apache_time(match.group(4)),
        method        = match.group(5),
        endpoint      = match.group(6),
        protocol      = match.group(7),
        response_code = int(match.group(8)),
        content_size  = size
    ), 1)
```

```
if __name__ == "__main__":
    start = time.time()
    #Print error message when the command is wrong.
    if len(sys.argv) != 2:
        print("Usage: wordcount <file>", file=sys.stderr)
        exit(-1)
    #Create a RDD to store parsed logs.
    sc = SparkContext(appName="PythonLog")
    parsed_logs = sc.textFile(sys.argv[1],8) \
                    .map(parseApacheLogLine) \
                    .cache()
    #Create a RDD to store accessed logs.
    access_logs = parsed_logs.filter(lambda s: s[1] == 1) \
                    .map(lambda s: s[0]) \
                    .cache()
    #Create a RDD to store failed logs.
    failed_logs = parsed_logs.filter(lambda s: s[1] == 0) \
                    .map(lambda s: s[0])
    #Count the number of failed logs.
    failed_logs_count = failed_logs.count()
    #Print failed logs and the number of them.
    if failed_logs_count > 0:
        print ('Number of invalid logline: %d' % failed_logs.count())
        for line in failed_logs.take(20):
            print ('Invalid logline: %s' % line)

    print ('Read %d lines, successfully parsed %d lines, failed to parse %d lines'
    % (parsed_logs.count(), access_logs.count(), failed_logs.count()))

    #Top Ten Error Endpoints.
    #Filter accessed logs with non-200 return code.
    not200 = access_logs.filter(lambda log: log.response_code != 200)
    endpointCountPairTuple = not200.map(lambda log: (log.endpoint, 1))
    endpointSum = endpointCountPairTuple.reduceByKey(lambda a, b: a + b)
    #Show the top 10 error urls in descending order.
    topTenErrURLs = endpointSum.takeOrdered(10, lambda s: -1 * s[1])

    print ('Top Ten failed URLs: %s' % topTenErrURLs)

    #Number of Unique Hosts.
    hosts = access_logs.map(lambda log: log.host)
    #filter same hosts to make unique hosts
    uniqueHosts = hosts.distinct()
    #count the number of unique hosts.
    uniqueHostCount = uniqueHosts.count()
    print ('Unique hosts: %d' % uniqueHostCount)
```

```
#Number of Unique Daily Hosts
#get unique logs for format day-host
dayToHostPairTuple = access_logs.map(lambda log: (log.date_time.day, log.host)).distinct()
#group hosts by day
dayGroupedHosts = dayToHostPairTuple.groupBy(lambda (x,y): x)
#count the number of different hosts that make requests each day
dayHostCount = dayGroupedHosts.map(lambda (x, hosts): (x, len(hosts)))
#sort day-count pairs by day in ascending order
dailyHosts = (dayHostCount.sortByKey() \
                          .cache())
# get the top 30 unique daily hosts
dailyHostsList = dailyHosts.take(30)
print ('Unique hosts per day: %s' % dailyHostsList)

#Counting 404 Response Codes. How many 404 records are in the log?
#filter the accessed logs whose response code is 404.
badRecords = (access_logs.filter(lambda log: log.response_code == 404) \
                         .cache())
#count the number of 404 logs
print ('Found %d 404 URLs' % badRecords.count())

#Listing the Top Twenty 404 Response Code Endpoints
badEndpointsCountPairTuple = badRecords.map(lambda log: (log.endpoint, 1))
#get endpoints with 404 response code and count number of each end ponits
badEndpointsSum = badEndpointsCountPairTuple.groupByKey().map(lambda (endpoint, counts): (endpoint, sum(counts)))
#sort the endpoint-count pairs by count in descending order and get the top 20 pairs.
badEndpointsTop20 = badEndpointsSum.sortBy(lambda (endpoint,count): -count).take(20)
print ('Top Twenty 404 URLs: %s' % badEndpointsTop20)

end = time.time()
print("Execution time:",end-start)
```

Part II (2 bonus points): Run Spark on a cluster in Amazon AWS.

1) A brief description of your cluster environment and the execution process of the code.

Spark Environment: AWS EMR(4 node), Windows 10

Code Structure:

Parsing each log line

Creating RDDs and cleaning data

Analyses log data

Execution Process:

1. Put the log file to HDFS.
2. Use command: spark-submit logmining.py access_log_Jul95

2) Execution time of your code in seconds.

```
Execution time: 138.759047031
```

3) Comparison of the execution results with those in Part I

Compared with Part I, Part II almost saved half of time spent on Part I.