MapReduce*

650.663: Cloud Computing Security

Fall 2017

In class, we've discussed MapReduce [1] as a paradigm for large-scale data processing. This assignment provides practical experience writing MapReduce jobs.

1 Overview

The National Do Not Call Registry[1] is a national database of telephone numbers of individuals who do not want to be contacted by telemarketers. Unfortunately, robocalls and spoofing are on the rise, leading to a record number of complaints in recent years.

In an effort to stop unwanted calls, law enforcement recently seized evidence from a cloud provider that assists businesses in contacting their customers.[2] In the excitement of the raid, however, service metadata about the records and cloud consumers was damaged, leaving only portions of logs recording phone calls. As part of the forensics team, you must uncover which cloud consumers are violating the law!

Fragments of the original call logs have been partially pieced together. Each log entry is a single line that provides the following information:

1. the date and time of a call,
2. the company responsible for the call,
3. the originating phone number for the call,
4. the recipient's phone number, and
5. the duration of the call (in seconds).

For example, the log entry

*This assignment was developed following a suggestion from course evaluations: "[…] we read MapReduce paper, why not give us some practical exercise? […] If instructor can give up some topics and prepare some homework for classical paper, it will be great!"

[1] https://www.donotcall.gov/

[2] The events that follow are fictitious. Any similarity to real life is purely incidental.

```
2017-04-09 17:48:04;Acme Corporation;4297854094;4298261640;9
```

indicates that the Acme Corporation used (429) 785-4094 to place a 9-second call to (429) 826-1640 slightly before 6 p.m. on 9 April 2017. Of course, this information alone is insufficient to determine if this phone call is legitimate: one must know that 1) the first telephone number has been reported for spam calls or 2) the second telephone number is part of the Do Not Call Registry and reported this specific call as unwanted.

Law enforcement has identified that the following numbers reported unwanted calls during the time frame captured in the call log: (216) 684-9356, (404) 934-5110, (589) 371-5037, and (945) 792-0329. You may assume that *all* calls that appear in the log and were placed to these numbers are violations of the Do Not Call Registry. That is, no business contacted these numbers for legitimate reasons such as an order confirmation.

As part of your forensics investigation, you must answer the following questions:

1. On how many occasions did companies violate the Do Not Call Registry?
2. How many numbers should be blocked / marked as spam to reduce the number of unwanted calls?
3. Which telephone numbers received the most spam calls?
4. Which telephone numbers are responsible for the most spam calls?
5. Which hours of the day are spam calls most likely?

Although you technically need not use MapReduce to answer these questions, large-scale data analysis practically necessitates a parallel processing framework.

2 What to turn in

You may implement the MapReduce jobs using any programming language. For simplicity, consider using a scripting language (e.g., Python) and Hadoop's streaming application programming interface (API) to facilitate testing your jobs.

Your submission for this assignment should comprise two parts: 1) a document that answers the aforementioned questions and uses the guidance that follows for forming your responses to them and 2) an archive that contains your source code. The source code archive should include all code used to answer each question, with the source code for each question in a separate directory named (`01`, `02`, …). That is, the root directory of the archive should contain a subdirectory for each question and each subdirectory could include all source code (i.e., implementation of the MapReduce job) used to answer that question.

Please note that answering some questions may require post-processing of the MapReduce results (e.g., extracting only the top-3 hours that had the most spam calls). You are not required to submit any code used for such post-processing, as it is assumed that you can perform this step manually.

Specific guidance for answering each question follows.

On how many occasions did companies violate the Do Not Call Registry? For each company that violated the Do Not Call Registry list the number of known unwanted calls placed by that company. That is, how many times did each company contact one of the numbers that reported unwanted calls? Order your results lexicographically by company (i.e., alphabetically by company name).

Your answer should resemble the following:

Acme Corporation 4

...

How many numbers should be blocked / marked as spam to reduce the number of unwanted calls? What is the total number of telephone numbers that should be blocked for each company? These telephone numbers should be *all* numbers used by the companies guilty of violating the Do Not Call Registry. That is, if the company violated the Do Not Call Registry once, then assume that all its calls should be marked as spam. Order your results lexicographically by company (i.e., alphabetically by company name).

Your answer should resemble the following:

Acme Corporation 6411

...

Which telephone numbers received the most spam calls? List the top-3 telephone numbers receiving spam calls and how many spam calls each received. Order your results in decreasing order of the telephone number receiving the most calls.

Your answer should resemble the following:

(847) 580-3060 18

...

Which telephone numbers are responsible for the most spam calls? List the top-3 telephone numbers placing spam calls and how many calls originated from each. Order your results in decreasing order of the telephone number responsible for the most spam calls.

Your answer should resemble the following:

(202) 221-4130 77
...

Which hours of the day are spam calls most likely? List the top-3 hours that had the most spam calls and how many spam calls were placed in each hour. Order the results in decreasing order of the number of calls.

Your answer should resemble the following:

11 a.m. 3157
...

3 Grading

Each question is worth 2 points. You will receive 1 point for each correct answer and 1 point for your source code.

References

[1] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation*, volume 6 of *OSDI '04*, pages 10–10, 2004.

[2] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google File System. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (SOSP '03)*, pages 29–43, 2003.

A Hadoop

Apache Hadoop[3] is an open source software framework for big data processing. It has several components, but the two most critical to this assignment are its

[3] https://hadoop.apache.org/

implementation of MapReduce and the Hadoop Distributed File System (HDFS), which are based on MapReduce [1] and the Google File System (GFS) [2] respectively.

For simplicity in this assignment, we'll use Hadoop's streaming API which allows the use of any executable script to define the `map` and `reduce` operations. The streaming API uses the standard input and output streams to pass information among jobs. More specifically, the `map` operation coverts lines of input (text-based and terminated by a line break) into a series of key-value pairs, one per line of output. After these key-value pairs are sorted (automatically by Hadoop), the `reduce` operation aggregates them to produce a final value for each unique key. By convention, the streaming API uses the first tab character on a line to delimit the key and value.

An advantage of the streaming API is that you can use command line utilities to test your `map` and `reduce` operations. For example, the following shell command executes two Python scripts using a (small) local data file:

```
cat /path/to/data | python map.py | sort | python reduce.py
```

where `cat` prints the specified data files on standard out, `map.py` defines the `map` operation, `sort` sorts the script's output in ascending order, and `reduce.py` defines the `reduce` operation. Of course, none of these steps are parallelized in this case, but Hadoop will perform the various operations in parallel when processing multiple data files.

B MapReduce

Writing a MapReduce job using the streaming API is straightforward. The canonical MapReduce example is counting words so we'll use it to illustrate the process.

As previously mentioned, the `map` operation reads input from standard in and outputs a series of key-value pairs, one per line. The following Python code implements this operation for counting words:

```
1  #!/usr/bin/env python
2
3  import sys
4
5
6  for line in sys.stdin:
7      line = line.strip()  # remove leading and trailing whitespace
8
9      # split line using whitespace as delimiters
10     tokens = line.split()
11     # iterate over tokens
```

```
    for token in tokens:
        print("{token}\t{count}".format(token=token, count=1))
```

That's it! This Python code outputs a stream of tokens with a '1' to indicate that each token was encountered once in the line of text. (If the same token appears multiple times, then it will be listed multiple times.) For example, the input

```
a man a plan a canal panama
```

becomes

```
a       →1
man       →1
a       →1
plan        →1
a       →1
canal         →1
panama          →1
```

where → indicates a tab character (i.e., `\t`).

The `reduce` operation reads the key-value pairs and aggregates them to produce a final value for each key. Of course, its input must be sorted to produce the correct results. The following Python code implements this operation for counting words:

```
#!/usr/bin/env python

import sys


def emit(token, count):
    print('{token}\t{count}'.format(token=token, count=count))


previous = None

for line in sys.stdin:
    line = line.strip()  # remove leading and trailing whitespace

    token, count = line.split('\t', 1)  # split key-value pair
    try:
        count = int(count)
    except ValueError:
        continue

```

```
21      if previous == token:
22          total = total + count
23      else:
24          if previous:
25              emit(previous, total)
26
27          previous = token
28          total = count
29
30  emit(token, total)
```

In essence, this script simply checks to see if the prior token is the same as the current token, incrementing the total count when they match and outputting the total when they differ.

Try writing these scripts and testing them as follows:

```
echo "a man a plan a canal panama" | map.py | sort | reduce.py
```

You should see the following result:

```
a       3
canal       1
man       1
panama       1
plan       1
```

(Note: Both scripts must be executable to invoke them in this fashion.)