PA3 Tutorial

CS 4740 Cloud Computing

Department of Computer Science,
University of Virginia, USA

- Video for computer program:

http://www.youtube.com/watch?v=bUB1RIpbFNs&mode=related&search=

Goal of this PA

- Gain hands-on experience with the MapReduce framework

- Understand the input and output of each phase
  - **<key, value> pair**

Python mrjob -- First Job in the Tutorial in "Fundamentals"

- A "step" consists of a mapper, a combiner (shuffle), and a reducer. All of those are optional, though you must have at least one.
- Map input
  - a key and a value (in this case, the key is ignored and a single line of text input is the value
- Map output
  - 3 keys for each line of text
  - Number of characters, number of words, and number of lines
- Reduce input
  - a key and an iterator of values
  - Why an iterator of values? Because all the values of the same key are passed to the same reducer.
- Reduce output
  - Sum the values for each key
  - Total number of characters, total number of words, total number of lines in the text

Required: import class MRJob from the library

```
from mrjob.job import MRJob

                                    Required: Inherit
class MRWordFrequencyCount(MRJob):
                                        Each line
    def mapper(self, _, line):
        yield "chars", len(line)
        yield "words", len(line.split())
        yield "lines", 1          An iterator of
                  key    value    values
    def reducer(self, key, values):
        yield key, sum(values)
                        Defined by yourself

                                    Required
if __name__ == '__main__':
    MRWordFrequencyCount.run()
```

Running the Job

The most basic way to run your job is on the command line:

```
$ python my_job.py input.txt
```

By default, output will be written to stdout.

If you like to save output to a file, then add “> file name”

The second job in the Tutorial in "Fundamentals"

- Most of the time, you'll need more than one step in your job. To define multiple steps, override steps() to return a list of MRSteps.

```
def steps(self):
    return [
        MRStep(mapper=self.mapper_get_words,
               combiner=self.combiner_count_words,
               reducer=self.reducer_count_words),
        MRStep(reducer=self.reducer_find_max_word)
    ]
```

The second job

Read this program!

Find the word with the highest occurrences.

```
from mrjob.job import MRJob
from mrjob.step import MRStep
import re

WORD_RE = re.compile(r"[\w']+")


class MRMostUsedWord(MRJob):

    def steps(self):
        return [
            MRStep(mapper=self.mapper_get_words,
                   combiner=self.combiner_count_words,
                   reducer=self.reducer_count_words),
            MRStep(reducer=self.reducer_find_max_word)
        ]

    def mapper_get_words(self, _, line):
        # yield each word in the line
        for word in WORD_RE.findall(line):
            yield (word.lower(), 1)

    def combiner_count_words(self, word, counts):
        # optimization: sum the words we've seen so far
        yield (word, sum(counts))

    def reducer_count_words(self, word, counts):
        # send all (num_occurrences, word) pairs to the same reducer.
        # num_occurrences is so we can easily use Python's max() function.
        yield None, (sum(counts), word)

    # discard the key; it is just None
    def reducer_find_max_word(self, _, word_count_pairs):
        # each item of word_count_pairs is (count, word),
        # so yielding one results in key=counts, value=word
        yield max(word_count_pairs)


if __name__ == '__main__':
    MRMostUsedWord.run()
```

```python
from mrjob.job import MRJob
from mrjob.step import MRStep
import re

WORD_RE = re.compile(r"[\w']+")


class MRMostUsedWord(MRJob):

    def steps(self):
        return [
            MRStep(mapper=self.mapper_get_words,
                   combiner=self.combiner_count_words,
                   reducer=self.reducer_count_words),
            MRStep(reducer=self.reducer_find_max_word)
        ]

    def mapper_get_words(self, _, line):
        # yield each word in the line
        for word in WORD_RE.findall(line):
            yield (word.lower(), 1)

    def combiner_count_words(self, word, counts):
        # optimization: sum the words we've seen so far
        yield (word, sum(counts))

    def reducer_count_words(self, word, counts):
        # send all (num_occurrences, word) pairs to the same reducer.
        # num_occurrences is so we can easily use Python's max() function.
        yield None, (sum(counts), word)

    # discard the key; it is just None
    def reducer_find_max_word(self, _, word_count_pairs):
        # each item of word_count_pairs is (count, word),
        # so yielding one results in key=counts, value=word
        yield max(word_count_pairs)


if __name__ == '__main__':
    MRMostUsedWord.run()
```

Remove punctuation

Define multiple steps

Count each word in one line

Occurrences of each word

Output just one key “_”, value is a tuple of (count, word)
So that you can use the max() easily

Output the word that has the highest occurrences

- line_split = line.split(',') # sep_length, sep_width, pet_length, pet_width, classification
- classification = line_split[-1] # last element
- sep_length = line_split[0] # first element

- yield classfication, float(sep_width)

- yield key, float(sum(sep_width)) / len(sep_width)

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
5.4,3.7,1.5,0.2,Iris-setosa
4.8,3.4,1.6,0.2,Iris-setosa
4.8,3.0,1.4,0.1,Iris-setosa
4.3,3.0,1.1,0.1,Iris-setosa
5.8,4.0,1.2,0.2,Iris-setosa
5.7,4.4,1.5,0.4,Iris-setosa
5.4,3.9,1.3,0.4,Iris-setosa
5.1,3.5,1.4,0.3,Iris-setosa
5.7,3.8,1.7,0.3,Iris-setosa
5.1,3.8,1.5,0.3,Iris-setosa
5.4,3.4,1.7,0.2,Iris-setosa
5.1,3.7,1.5,0.4,Iris-setosa
4.6,3.6,1.0,0.2,Iris-setosa
5.1,3.3,1.7,0.5,Iris-setosa
4.8,3.4,1.9,0.2,Iris-setosa
5.0,3.0,1.6,0.2,Iris-setosa
5.0,3.4,1.6,0.4,Iris-setosa
```

yield key, float(sum(sep_width)) / len(sep_width)

- This equation does not work for some computers. "float division by zero" ZeroDivisionError will be shown.

  Because the value input of reducer is a generator. Generator in python can be only used once. After sum(sep_width), the generator becomes empty, so len(sep_width)=0.

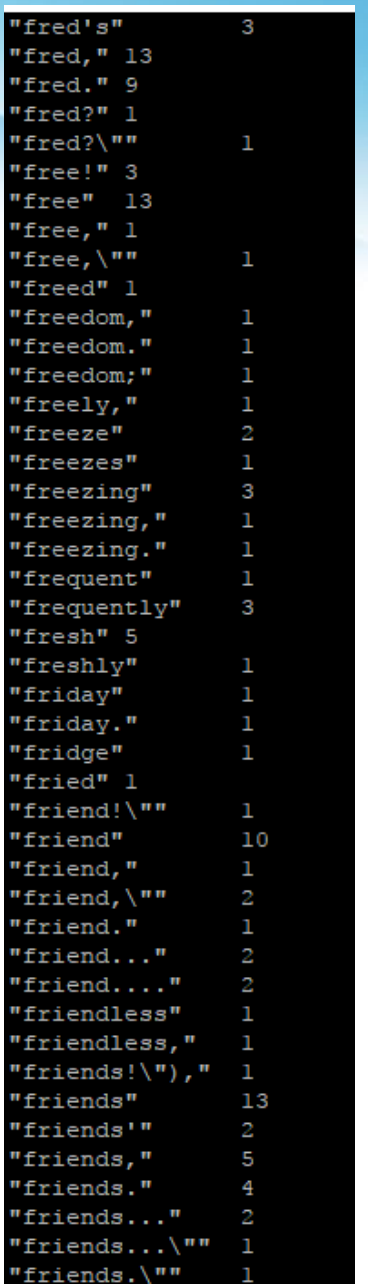
- You can use a "for" loop to calculate the average -- "for i in sep_width".

3) if key == 'Iris Setosa':

One tip for Step 4 of PA3 …

- The output of reducer is sorted by key.

- Use this feature to sort the occurrences of words.



Step 4

- Original order:
- 1
- 1111
- 2
- 2222

- What we want:
- 1
- 2
- 1111
- 2222

```
def map_sort(self, word, count):
    count = '%04d' % int(count) # change integer to string with 4 characters
    yield count, word
```

- Print out all data and just snapshot the result needed

Run on Amazon EMR

- The steps on PA3 document are already very clear.

Important Note

- A single-threaded implementation of MapReduce will usually not be faster than a traditional (non-MapReduce) implementation.
- MapReduce is good for multi-threaded implementations.
- So, to get full credit, do not use the traditional implementation to finish the PA. Please use mrjob.