Week 11
Some Hadoop MapReduce Examples

Mastering Cloud Computing

Coleman Kane
(based on material by Paul Talaga)

UNIVERSITY OF Cincinnati

Basic MapReduce Recipe (Python)

Most MapReduce applications operate in this manner:

- There's a "mapper" tool that receives the raw data set
- a "reducer" tool that receives the mapper's output as input
- Reducer's output produces the result set

Basic MapReduce Recipe (Python)

Mapper

- Maps/filters input data elements to intermediate representation
- Takes data on STDIN
- Yields intermediate “key / value” pairs on STDOUT

Reducer

- Reduces sets of intermediate data elements sharing a particular key into a single result
- Takes intermediates in STDIN
- Yields summary result on STDOUT

https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html#MapReduce_-_User_Interfaces

UNIVERSITY OF Cincinnati

Basic MapReduce Recipe (Python) cmdline

```
cd /usr/local/hadoop

hadoop jar \
share/hadoop/tools/lib/hadoop-streaming-2.7.1.jar \
-input  /users-cloud-16fs/kaneca/input/file.txt   \
-output /users-cloud-16fs/kaneca/output/job1-out  \
-mapper ~/mr_ex/mapper.py                        \
-reducer ~/mr_ex/reducer.py                      \
-file ~/mr_ex/{mapper,reducer}.py
```

UNIVERSITY OF Cincinnati

Simple Example - Word Frequency Counter (mapper)

Convert the input data into a “word N” format that encodes the word followed by the frequency count (N) observed over the interval.

In this case, interval is an individual word, and the
output lines > input lines

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

import sys

# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    # increase counters
    for word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        #
        # tab-delimited; the trivial word count is 1
        print '%s\t%s' % (word, 1)
```

UNIVERSITY OF Cincinnati

Simple Example - Word Frequency Counter (reducer)

"word N" rows accepted as input

MapReduce will sort based upon the first field in the row

Yields output rows that are the same format as the input rows, but are summed aggregates of the data set

Consider the "intermediate" data as a "partial computation", if you prefer

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

from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

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

    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)

    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        continue

    # this IF-switch only works because Hadoop sorts map output
    # by key (here: word) before it is passed to the reducer
    if current_word == word:
        current_count += count
    else:
        if current_word:
            # write result to STDOUT
            print '%s\t%s' % (current_word, current_count)
        current_count = count
        current_word = word

# do not forget to output the last word if needed!
if current_word == word:
    print '%s\t%s' % (current_word, current_count)
```

UNIVERSITY OF Cincinnati

Simple Example - Word Frequency Counter (results)

For simple input, works pretty well. Consider file containing the following text:
“one two three a b one three four”

Yields the following output (in output file):

```
a       1
b       1
four    1
one     2
three   2
two     1
```

Simple Example - Word Frequency Counter (results)

For complex input, falls down. Consider file containing the following text: “one two three. a b one, three four.”

Yields the following output (in output file):

```
a       1
b       1
four.   1
one     1
one,    1
three   1
three.  1
two     1
```

UNIVERSITY OF Cincinnati

Improved Ex. - Word Frequency Counter (mapper)

New logic introduced to clean up words that contain stray punctuation

Still continue to use the same reducer code

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

import sys

# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()

    # increase counters
    for word in words:
        # Remove excess punctuation, etc, leading/trailing
        # from each word
        word = word.strip('.,;:"\'')

        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        #
        # tab-delimited; the trivial word count is 1
        print '%s\t%s' % (word, 1)
```

UNIVERSITY OF Cincinnati

Improved Example - Word Frequency Counter (results)

Works better on file containing the following text:
“one two three. a b one, three four.”

Yields the following output (in output file):

```
a       1
b       1
four    1
one     2
three   2
two     1
```

Matches the output from run without punctuation

Local Testing without Hadoop

Helpful for debugging - mostly single-threaded execution

Mapper:

- python mapper.py < input.txt

Reducer:

- python mapper.py < input.txt | sort -k 1 | \
  python reducer.py > results.txt

UNIVERSITY OF Cincinnati