CME 193: Introduction to Scientific Python
Lecture 7: MapReduce

Dan Frank

Institute for Computational and Mathematical Engineering (ICME)

January 30, 2014

Administrivia

- Final HW 5 due on Feb 4
- No class Feb 2

MapReduce

Hadoop

What is MapReduce

Me: A programming paradigm that fits many computational tasks and an infrastructure to execute code written in that framework. Enables massively parallel and fault tolerant code execution over arbitrarily large datasets. The most common implementation of such an infrastructure is open-source and called Hadoop.

Wikipedia: MapReduce is a programming model for processing large data sets with a parallel, distributed algorithm on a cluster.

What is MapReduce: classic map()

A map operation takes several pieces of data and computes with each. No need to have this data or computation on only one machine.

```
f = lambda x: x + 1

map(f, [1, 2, 3, 4])
```

What is MapReduce: classic reduce()

A reduce operation takes several pieces of data and combines them. Need to have all the data and computation done on one machine.

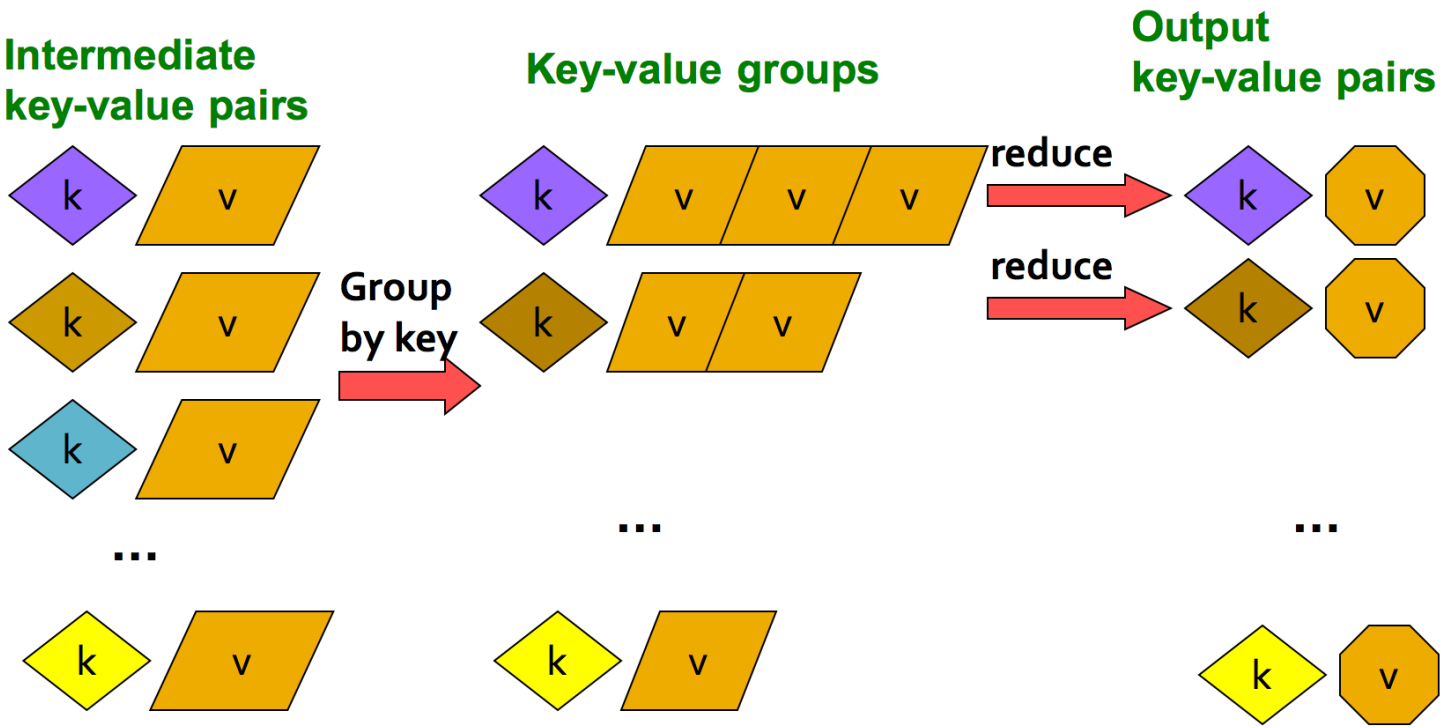
```
lst = [1, 2, 3, 4, 5]

reduce(lambda x, y: x + y, lst)

# computes ((((1+2)+3)+4)+5)
```

MapReduce: 3 steps

Computation structured with `(key, value)` pairs

1. `mapper(key, value)` take one `(key, value)` pair as in put and output 0 or more `(key, value)` pairs
2. `shuffle/sort/groupby`: Group all of the values corresponding to the same key together (to be sent to a reducer). Taken care of for you by MapReduce infrastructure.
3. `reducer(key, values)` take a key and a list of values as input and output 0 or more `(key, value)` pairs

MapReduce: 3 steps



shamelessly stolen from CS246

Aside: Python Generators & Lazy Evaluation

Suppose I want to represent the entire set of natural numbers in a python program. This set is infinite so we clearly can't store it in memory. But so long as we don't need to access every integer at once, we can still represent them using generators

```
def integer_generator(maxval=float('inf')):
    # generate integers up to maxval
    i = 0
    while i < maxval:
        yield i
        i += 1
```

Generator Example

```
from integer_generator import integer_generator

gen = integer_generator(2)
for i in gen:  # prints 0, 1
    print i

gen = integer_generator(2)
print gen.next()  # prints 0
print gen.next()  # prints 1
print gen.next()  # StopIteration error

gen = integer_generator()
for i in gen:
    # never terminates... prints all integers
    print i
```

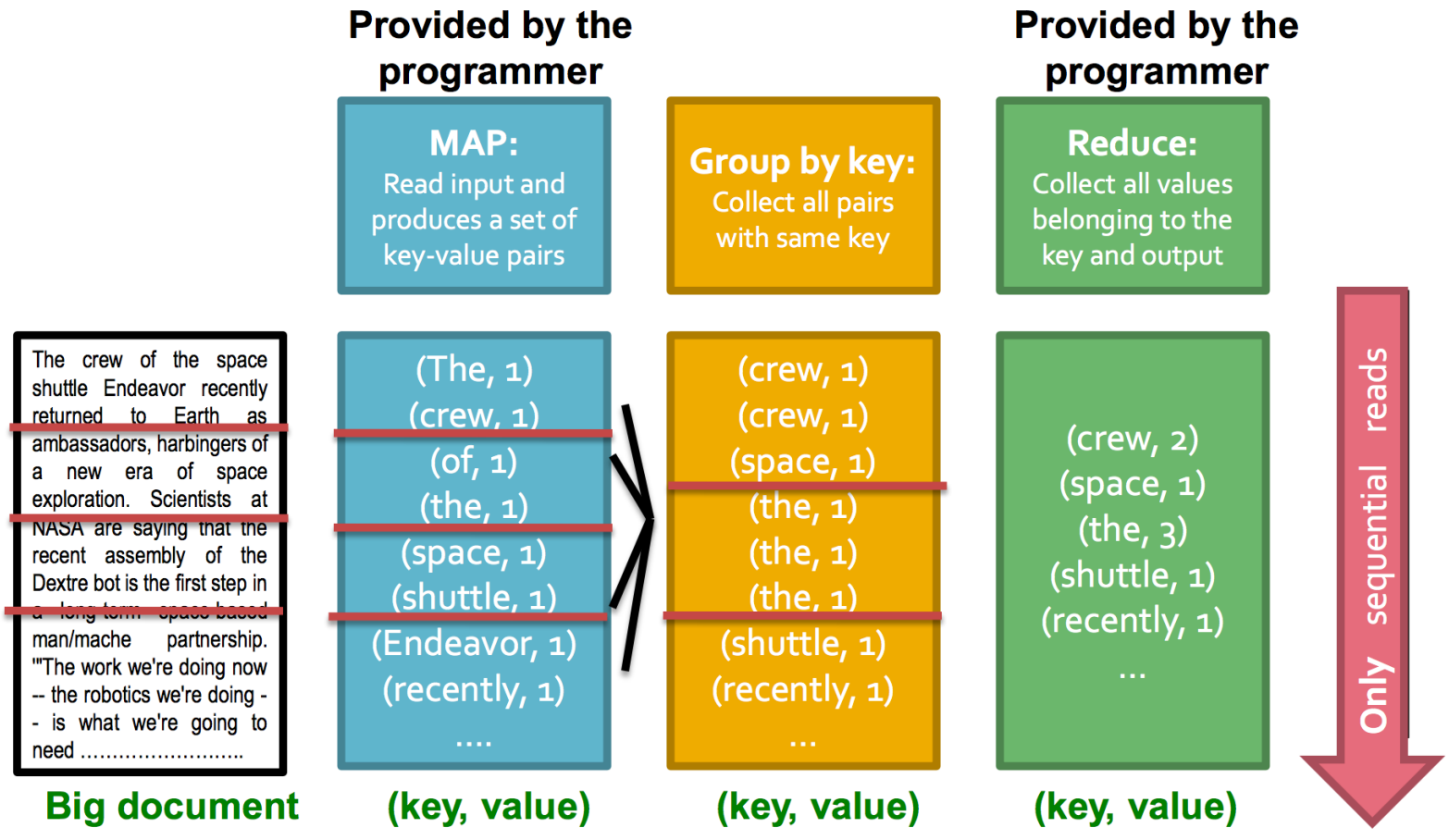
Classic example: Word Count

Take a large file and output the counts of each word in that file.

We'll input each line of the file into our mappers as values, ignoring the key. From each mapper we'll emit pairs consisting of (word, 1).

The reducer will take keys and get a list of 1's as values which we will sum to get each word count.

Classic example: Word Count



shamelessly stolen from CS246

Mapper for WordCount

Take in one `(_, line_of_text)` pair and output (word, 1) pairs

```
def mapper(key, value):
    # value is ignored, key is a line of text
    for k in key.split():
        yield k.strip(), 1
```

Reducer for WordCount

Take in one (`word, list_of_ones`) pair and output (word, sum) pair

```
def reducer(key, values):
    # key is a word string
    # values is a list of 1's corresponding
    #   to a mapper finding that word
    yield key, sum(values)
```

MapReduce

Hadoop

What is Hadoop

The standard open-source MapReduce infrastructure including...

1. job scheduler to execute map and reduce tasks with appropriate data
2. job tracking to re-execute failed jobs
3. storage infrastructure: distributed file system (HDFS)

Hadoop Setup

How do we hook up our code to a MapReduce infrastructure... especially since it's written in Java.

1. Running Hadoop On Ubuntu Linux (Single-Node Cluster) - How to set up a pseudo-distributed, single-node Hadoop cluster backed by the Hadoop Distributed File System (HDFS)
2. Running Hadoop On Ubuntu Linux (Multi-Node Cluster) - How to set up a distributed, multi-node Hadoop cluster backed by the Hadoop Distributed File System (HDFS)

Hadoop Streaming

Mostly Hadoop expects us to be writing in Java or at least Jython, but we'd like to keep writing pure python, so we can instead use Hadoop Streaming which passes key values to an executable. In our case this will be python.

The values are passed via standard input and output values are expected to be written to standard out. And key value pairs are separated by the first TAB.

Hadoop Streaming

Here's how we call hadoop streaming...

```
$HADOOP_HOME/bin/hadoop jar \
  $HADOOP_HOME/hadoop-streaming.jar \
    -input input_dir \
    -output output_dir \
    -mapper mapper.py \
    -reducer reducer.py\
    -file mapper.py  \
    -file reducer.py
```

Hadoop Streaming: Mapper

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

def mapper(key, value):
    # value is ignored, key is a line of text
    for word in key.split():
        yield word.strip(), 1

if __name__ == "__main__":
    for line in sys.stdin:
        keyval = line.split('\t')
        if len(keyval) > 1:
            key, val = (keyval[0], '\t'.join(keyval[1:]))
        else:
            key, val = (keyval[0], None)
        for outkey, outval in mapper(key, val):
            sys.stdout.write('%s\t%s\n' % (outkey, outval))
```

Hadoop Streaming: Reducer

```
#!/usr/bin/env python
import sys
from itertools import groupby, imap
from operator import itemgetter

def reducer(key, values):
    # key: word; values: list of 1's
    yield key, sum([int(v) for v in values])

if __name__ == "__main__":
    input = imap(lambda l: l.strip().split('\t'), sys.stdin)
    for key, val_gen in groupby(input, key=itemgetter(0)):
        vals = map(itemgetter(1), val_gen)
        for outkey, outval in reducer(key, vals):
            sys.stdout.write('%s\t%s\n' % (outkey, outval))
```

Hadoop Streaming: Testing

Since we're using `stdin` and `stdout`, we can easily test out code locally.

```
cat input | ./mapper.py | sort | ./reducer.py
```

Which does basically locally what is happening in the cluster.

Be sure to `chmod +x` your python mapper and reducer so they are executable

Hadoop Streaming: Testing

```
turncoat:code danfrankj$ cat course_description.txt | ./hs_mapper.py
CME     1
193     1
Introduction    1
to      1
Scientific      1
Python  1
This    1
short   1
course  1
```

Hadoop Streaming: Testing

```
turncoat:code danfrankj$ cat course_description.txt | ./hs_mapper.py | sort
01      1
1       1
1       1
193     1
193     1
200-034 1
60784   1
CME     1
CME     1
Class   1
Credit  1
Credit  1
```

Hadoop Streaming: Testing

```
turncoat:code danfrankj$ cat course_description.txt | ./hs_mapper.py | sort | ./hs_reducer.py
01      1
1       2
193     2
200-034 1
60784   1
CME     2
Class   1
Credit  2
File    1
Grading 2
I       1
```

Hadoop Streaming: On Your Cluster

1. copy local data to HDFS
2. launch job
3. monitor job status
4. collect output

Hadoop Streaming: Copying local data

First we need to copy local data into HDFS so that our jobs can find it

```
$HADOOP_HOME/bin/hadoop dfs -copyFromLocal \
  course_description.txt input_dir
$HADOOP_HOME/bin/hadoop dfs -ls input_dir
```

Hadoop Streaming: Launching the Job

Once our cluster is configured we can launch the job with...

```
$HADOOP_HOME/bin/hadoop jar \
  $HADOOP_HOME/hadoop-streaming.jar \
    -input input_dir \
    -output output_dir \
    -mapper python_mapper.py \
    -reducer /bin/cat # executable\
    -file python_mapper.py # make file available \
    -file auxiliary_data.txt
```

Hadoop Streaming: Monitoring Job Status

http://localhost:50030/ `http://localhost:50030/`

Hadoop Streaming: Collecting Output

```
$HADOOP_HOME/bin/hadoop dfs -ls output_dir
$HADOOP_HOME/bin/hadoop dfs -cat output_dir/part-* > \
  local_output
```