MrJob, Unsupervised Learning at Scale: Clustering, Expectation Maximization

James G. Shanahan [1,2]

[1]Church and Duncan Group, *[2]iSchool UC Berkeley, CA*

EMAIL: James_DOT_Shanahan_AT_gmail_DOT_com

Lecture #7 and #8

June 14, 2016

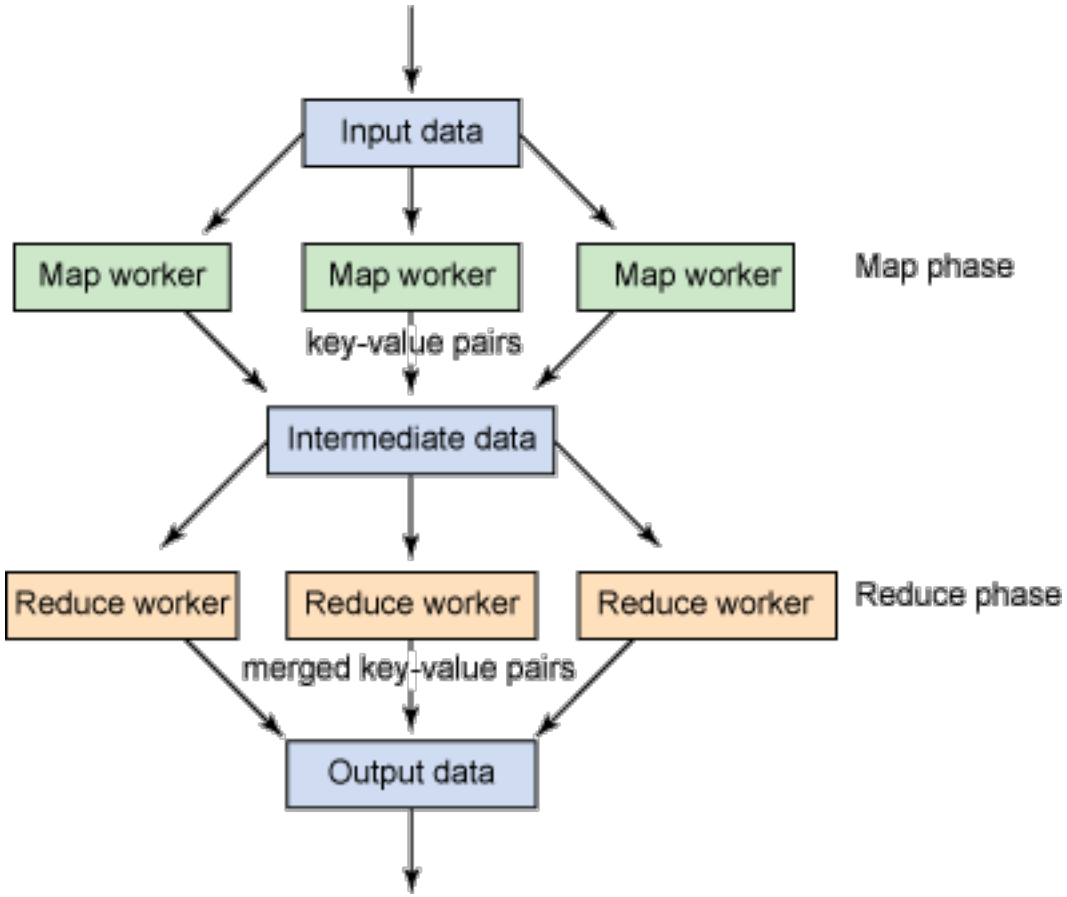
Map Reduce Algorithm Design

- 7.1 Background and Motivation (5)
- 7.2 MrJob
  - Installation (5)
  - MrJob Fundamentals and Concepts (5)
  - Writing Mrjob code (10)
  - BLT Word frequency challenge (15)
  - Log file processing (10)
  - BLT log file processing challenge (15)
  - Serializable, JSON and other MrJob info (10)
- END of Lecture
- Clustering Algorithms
  - Clustering overview (10 mins)
  - Kmeans algorithm (5)
  - Distributed Kmeans in MrJob [ScreenFlow] (10 minutes)
    - Initialization (Canopy Clustering)
- Sync time
  - MrJob : cluster tweets
  - Model-based clustering [Sync time or some in lecture 5]
    - Intro [5]

V4

Map Reduce

- ..

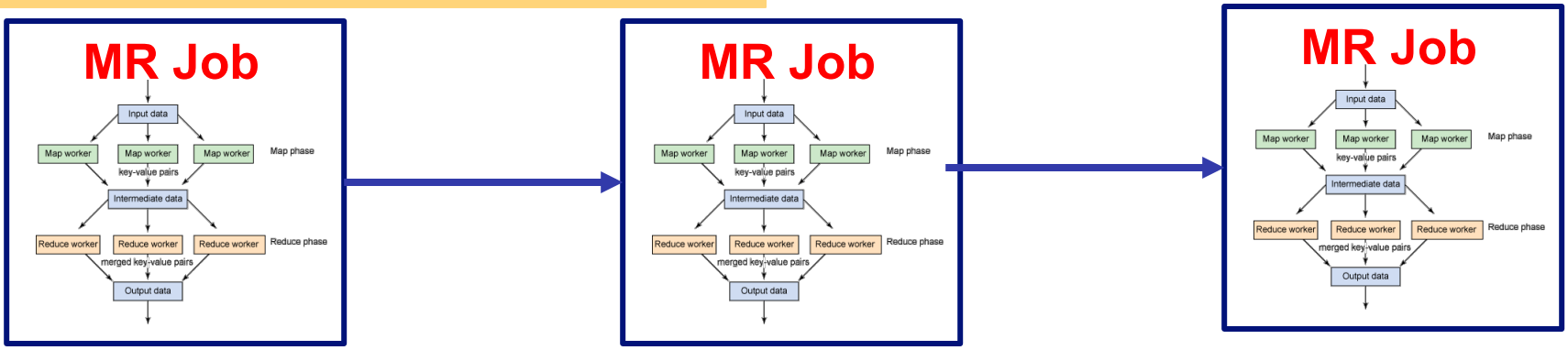

Input data
Map worker
Map worker
Map worker
Map phase
key-value pairs
Intermediate data
Reduce worker
Reduce worker
Reduce worker
Reduce phase
merged key-value pairs
Output data

Building a Multistep Pipeline

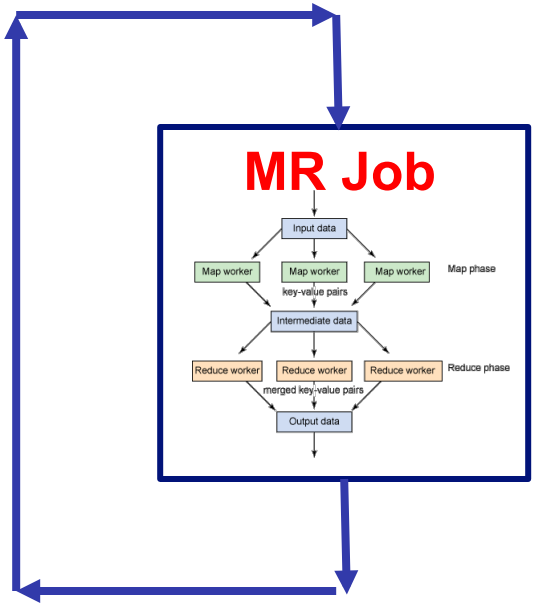
- **Although some large data processing tasks can be accomplished using a single MapReduce step, a common pattern is to run a job that takes multiple steps.**
- **In other cases, two separate sources will use two mapper functions to emit data containing the same key.**
  - Then a single reducer phase will combine the data based on the matching key into a combined output
- **E.g., Trending queries**
  - Get the counts
  - Sort in decreasing order

Pipeline of jobs

Pipeline of jobs in sequence



Iterative Pipeline



Background and Motivation

- **Hadoop**
  - Hadoop framework is written in Java

- **How to write code a hadoop mapreduce job in Python?**
  - Translate Python code using Jython (Inconvenient, problematic if you depend on Python features not provided by Jython)
  - Hadoop Streaming API
  - Mrjob

Background and Motivation

- **WordCount Example**
  - Java Code
- **Hadoop Streaming is also limited and complex**

```
1.  package org.myorg;
2.
3.  import java.io.IOException;
4.  import java.util.*;
5.
6.  import org.apache.hadoop.fs.Path;
7.  import org.apache.hadoop.conf.*;
8.  import org.apache.hadoop.io.*;
9.  import org.apache.hadoop.mapred.*;
10. import org.apache.hadoop.util.*;
11.
12. public class WordCount {
13.
14.   public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {
15.     private final static IntWritable one = new IntWritable(1);
16.     private Text word = new Text();
17.
18.     public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
19.       String line = value.toString();
20.       StringTokenizer tokenizer = new StringTokenizer(line);
21.       while (tokenizer.hasMoreTokens()) {
22.         word.set(tokenizer.nextToken());
23.         output.collect(word, one);
24.       }
25.     }
26.   }
27.
28.   public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {
29.     public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOExcepti
30.       int sum = 0;
31.       while (values.hasNext()) {
32.         sum += values.next().get();
33.       }
34.       output.collect(key, new IntWritable(sum));
35.     }
36.   }
37.
38.   public static void main(String[] args) throws Exception {
39.     JobConf conf = new JobConf(WordCount.class);
40.     conf.setJobName("wordcount");
41.
42.     conf.setOutputKeyClass(Text.class);
43.     conf.setOutputValueClass(IntWritable.class);
44.
45.     conf.setMapperClass(Map.class);
46.     conf.setCombinerClass(Reduce.class);
47.     conf.setReducerClass(Reduce.class);
48.
49.     conf.setInputFormat(TextInputFormat.class);
50.     conf.setOutputFormat(TextOutputFormat.class);
51.
52.     FileInputFormat.setInputPaths(conf, new Path(args[0]));
53.     FileOutputFormat.setOutputPath(conf, new Path(args[1]));
54.
55.     JobClient.runJob(conf);
56.   }
57. }
58.
59.
```


n_AT_gmail.com

In Hadoop

In [48]:
```
#usr/local/Cellar/hadoop/2.6.0/libexec/share/hadoop/tools/lib
dataDir = "/Users/jshanahan/Dropbox/lectures-uc-berkeley-ml-class-2015/Notebooks/WordCount"

!hadoop jar /usr/local/Cellar/hadoop/2.6.0/libexec/share/hadoop/tools/lib/hadoop-streaming*.jar \
   -mapper WordCount/mapper.py \
   -reducer WordCount/reducer.py \
   -input historical_tours.txt \     #file on Hadoop
   -output gutenberg-output          #output directory on Hadoop
```

```
ses where applicable
15/02/26 21:03:32 INFO Configuration.deprecation: session.id is deprecated. Instead, use dfs.metrics.session-id
15/02/26 21:03:32 INFO jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker, sessionId=
15/02/26 21:03:32 INFO jvm.JvmMetrics: Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initiali
zed
15/02/26 21:03:32 INFO mapred.FileInputFormat: Total input paths to process : 1
15/02/26 21:03:32 INFO mapreduce.JobSubmitter: number of splits:1
15/02/26 21:03:32 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_local2006957706_0001
15/02/26 21:03:33 INFO mapreduce.Job: The url to track the job: http://localhost:8080/
15/02/26 21:03:33 INFO mapred.LocalJobRunner: OutputCommitter set in config null
15/02/26 21:03:33 INFO mapreduce.Job: Running job: job_local2006957706_0001
15/02/26 21:03:33 INFO mapred.LocalJobRunner: OutputCommitter is org.apache.hadoop.mapred.FileOutputCommitter
15/02/26 21:03:33 INFO mapred.LocalJobRunner: Waiting for map tasks
15/02/26 21:03:33 INFO mapred.LocalJobRunner: Starting task: attempt_local2006957706_0001_m_000000_0
15/02/26 21:03:33 INFO util.ProcfsBasedProcessTree: ProcfsBasedProcessTree currently is supported only on Linux.
15/02/26 21:03:33 INFO mapred.Task:  Using ResourceCalculatorProcessTree : null
15/02/26 21:03:33 INFO mapred.MapTask: Processing split: hdfs://localhost:9000/user/jshanahan/historical_tours.txt:0+87483
15/02/26 21:03:33 INFO mapred.MapTask: numReduceTasks: 1
15/02/26 21:03:33 INFO mapred.MapTask: (EQUATOR) 0 kvi 26214396(104857584)
```

Python frameworks for Hadoop

- Hadoop Streaming
- mrjob (Yelp)
- dumbo
- Luigi (Spotify)
- hadoopy
- pydoop
- PySpark
- happy
- Disco
- octopy
- Mortar Data
- Pig UDF/Jython
- hipy
- Impala + Numba

cloudera

Goals for Python framework

1. "Pseudocodiness"/simplicity
2. Flexibility/generality
3. Ease of use/installation
4. Performance

cloudera

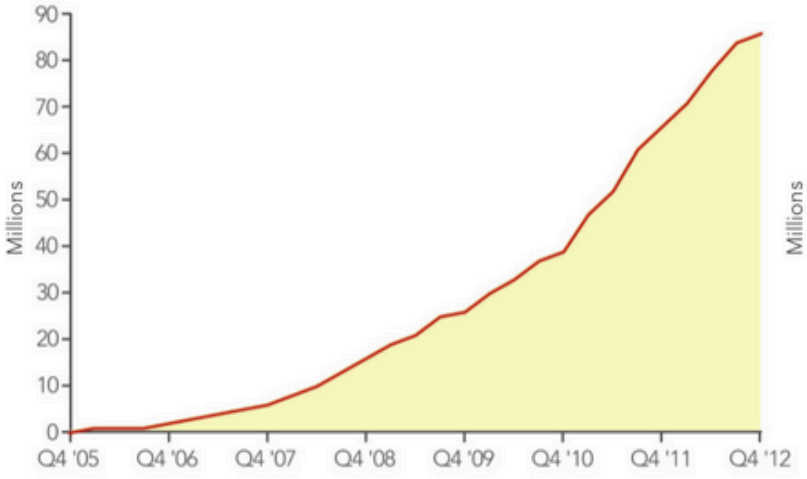
Background and Motivation

- **Mrjob is a Python package for running Hadoop streaming jobs.**
- **Mrjob is a python-based framework that assists you in submitting your job to the Hadoop job tracker and in running each individual step under Hadoop Streaming.**
- **Developed at Yelp.com**
- **It's offered under the Apache 2.0 license.**
- **The software was written to power a feature called "People Who Viewed this Also Viewed."**
  - (Regular users will see it in the lower right-hand corner of the screen when they look at popular content.);
  - used for advertising etc..

10 minutes to process a GB on one machine
250+ GB of logs per day

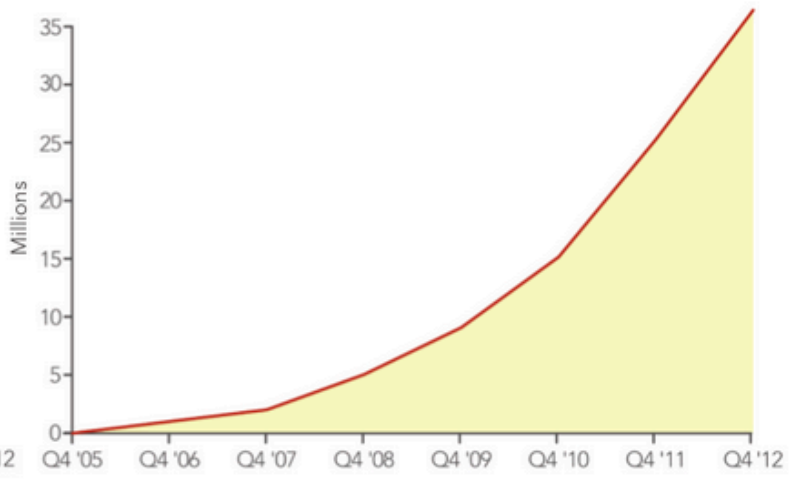
- 250+ GB of logs per day
- Each GB takes 10 minutes to process
- How long to handle a day's logs?

86 Million Monthly Visitors



Average monthly unique visitors for the quarter, as measured by Google Analytics

36 Million Reviews



Cumulative reviews contributed since inception

Too long

- On a single machine 40+ hours!
- If we really had only a single machine, we wouldn't be able to keep up!
- Mistake can't be fixed in a day (billing especially important)

Why MrJob

- **mrjob lets you write MapReduce jobs in Python 2.5+ and run them on several platforms.**
- **You can:**
  - Write multi-step MapReduce jobs in pure Python
  - Test on your local machine
  - Run on a Hadoop cluster
  - Run in the cloud using Amazon Elastic MapReduce (EMR)
  - Keep all MapReduce code for one job in a single class
  - Switch input and output formats with a single line of code
  - Simple and easy to learn!!!

- **mrjob is licensed under the Apache License, Version 2.0.**

mrjob: features

- Abstracted MapReduce interface
- Handles complex Python objects
- Multi-step MapReduce workflows
- Extremely tight AWS integration
- Easily choose to run locally, on Hadoop cluster, or on EMR
- Actively developed; great documentation

Background and Motivation

- **WordCount Example**
  - Mrjob Python Code

```
from mrjob.job import MRJob

class MRWordCounter(MRJob):
    def mapper(self, key, line):
        for word in line.split():
            yield word, 1

    def reducer(self, word, occurrences):
        yield word, sum(occurrences)

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

Solution: develop MrJob

- **A Divide and conquer system using Map/Reduce**

- ***Map***
  - Extract a property to summarize over
- ***Reduce***
  - Summarize all items with a particular property

- **Simple: Each operation stateless**
  - Stateless constraint means it can be used across thousands of computers
- **MapReduce's main benefits are for running over many machines, fault tolerance**

MrJob, Dumbo: wrappers for Hadoop Streaming API

Alternative Python-Based MapReduce Frameworks

mrjob is not the only Python-based MapReduce framework for Hadoop streaming. Another popular framework is **Dumbo**. Dumbo is similar to mrjob in many ways, but even simple MapReduce tasks, the structure of a Dumbo script is fairly similar to one using mrjob. In my opinion, one of the strongest differentiators is in mrjob's EMR integration. Because Yelp runs a lot of data processing tasks in the Amazon cloud, kicking off a job on Elastic MapReduce is a bit easier using mrjob than it is when using Dumbo. To its credit, Dumbo is probably a bit easier to use if you need to work with some type of custom data-input format.

Both mrjob and Dumbo are wrappers for the Hadoop streaming API, but they don't attempt to provide direct access to all the classes and methods found in the lower-level Hadoop Java API. For even more fine-grained access to these functions, it might be worth taking a look at Pydoop. Pydoop's goal is to be a Python interface for Hadoop and HDFS itself. By sidestepping Hadoop's streaming API, it *may* be more performant to use Pydoop over frameworks such as mrjob or Dumbo. If you are analyzing terabyte datasets, a 2X boost in performance might mean huge savings in overall processing time.

MrJob, Dumbo: wrappers for Hadoop Streaming API
PyDoop: a python interdace to Hadoop and HDFS itself

[Data Just Right: Introduction to Large-scale Data & Analytics
By Michael Manoochehri]

8. Putting It Together: MapReduce Data Pipelines

It's kind of fun to do the impossible.

—Walt Disney

Human brains aren't very good at keeping track of millions of separate data points, but we know that there is lots of data out there, just waiting to be collected, analyzed, and visualized. To cope with the complexity, we create metaphors to wrap our heads around the problem. Do we need to store millions of records until we figure out what to do with them? Let's file them away in a *data warehouse*. Do we need to analyze a billion data points? Let's *crunch* it down into something more manageable.

No longer should we be satisfied with just storing data and chipping away little bits of it to study. Now that distributed computational tools are becoming more accessible and cheaper to use, it's more and more common to think about data as a dynamic entity, flowing from a source to a destination. In order to really gain value from our data, it needs to be transformed from one state to another and sometimes literally moved from one physical location to another. It's often useful to think about looking at the state of data while it is moving from one state to another. In order to get data from here to there, just like transporting water, we need to build pipelines.

What Is a Data Pipeline?

At my local corner store, there is usually only one person working at any given time. This person runs the register, stocks items, and helps people find things. Overall, the number of people who come into this clerk's store is fairly manageable, and the clerk doesn't usually get overwhelmed (unless there is a run on beer at 1:55 a.m.). If I were to ask the shopkeeper to keep track of how many customers came in that day, I am pretty sure that not only would this task would be manageable, but I could even get an accurate answer.

The corner store is convenient, but sometimes I want to buy my shampoo and potato chips in gigantic sizes that will last me half the year. Have you ever shopped in one of those massive warehouse clubs, the ones that sell wholesale-sized pallets of toilet paper and ketchup by the kilogram? One of the things that has always amazed me about these huge stores is the number of checkout lines available to handle the flow of customers. Thousands and thousands of shoppers might be purchasing items each hour, all day long. On any given weekend, there might be twenty or more checkout lines open, each with dozens of customers waiting in line.

The checkout lines in a warehouse club are built to handle volume; the staff running the registers are not there to help you find items. Unlike the corner store clerk, the job of the register staff is specialized; it's to help the huge number of customers check out quickly.

There are other specialized tasks to be performed in our warehouse club. In order to be able to move pallets of two-liter maple syrup bottles to the sales floor, some employees must specialize in driving forklifts. Other employees are there simply to provide information to shoppers.

Now imagine that, as you pay for your extra-large pallet of liquid detergent, you ask a person at the checkout counter about the total number of customers that pass through all checkout lines for the entire day. It would be difficult for them to give you a real answer. Although the person at the register might easily be able to keep a running tally of the customers making their way through a single line, it would be difficult for them to know what is going on in the other checkout lines. The individuals at each register don't normally communicate with each other very much, as they are too busy with their own customers. Instead, we would need to deploy a different member of the staff whose job it is to go from register to register and aggregate the individual customer counts.

The Right Tool for the Job

The point here is that as customer volume grows to orders of magnitude beyond what a small convenience store is used to, it becomes necessary to build specialized solutions. Massive-scale data problems work like this too. We can solve data challenges by distributing problems across many machines and using specialized software to solve discreet problems along the way. A data pipeline is what facilitates the movement of our data from one state of utility to another.

Traditionally, developers depended on single-node databases to do everything. A single machine would be used to collect data, store it permanently, and run queries when we needed to ask a question. As data sizes grow, it becomes impossible to economically scale a single machine to meet the demand. The only practical solution is to distribute our needs across a collection of machines networked together in a cluster.

Collecting, processing, and analyzing large amounts of data sometimes requires using a variety of disparate technologies. For example, software specialized for efficient data collection may not be optimized for data analysis. This is a lot like the story of the warehouse club versus the tiny convenience store. The optimal technology necessary to ask huge, aggregate questions about massive datasets may be different from the software used to ensure that data can be collected rapidly from thousands of users. Furthermore, the data itself may need to be structured differently for different applications. A data-pipeline strategy is necessary to convert data from one format to another, depending on the need.

One of my favorite things about emerging open-source technologies in data is the potential to create systems that combine different tools to deal with massive amounts of data. Much of the value of large Internet companies is based on their ability to build efficient systems for large-scale data processing pipelines, and this technology is very quickly becoming more accessible.

Data Pipelines with Hadoop Streaming

One of the hallmarks of large-scale data processing technologies is the use of multiple machines in parallel to tackle data challenges. Using a cluster of inexpensive machines (or even a collection of virtual machines in the cloud) allows us to solve data problems that would be thought of as impossible just a few years ago. Not all problems can be easily solved by distributing them across a collection of machines, but many data collection and transformation tasks are well suited to this approach. Taking large amounts of unstructured data collected from a variety of sources, transforming it into something more structured, and then analyzing it is a very common use case. While there are many strategies for how to distribute a computational problem, a good general purpose approach is known as *MapReduce*. MapReduce can be a great way to facilitate large-scale data transformations in a reasonable amount of time, and many people are using it as the fundamental part of their data-pipeline work.

Listing 8.9 **A simple one-step MapReduce counter using mrjob**

```
from mrjob.job import MRJob

class MRBirthCounter(MRJob):
  # The mapper will read records from stdin
  def mapper(self, key, record):
    yield record[14:20], 1

  def reducer(self, month, births):
    # The reducer function yields a sum of the
    # counts of each month's births.
    yield month, sum(births)

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

Underneath the hood, the *mrjob_simple_example.py* script uses the same Hadoop streaming API that we used earlier, meaning that it is still possible to test our code by piping from stdin.

After we've tested our mrjob script on a local machine, let's run it on a Hadoop cluster (Listing 8.10). This step is as easy as running the script like a normal Python application, as long as we've set the HADOOP_HOME environment variable on the machine in which our script lives. In order to do this, we will need to use the -r flag to specify that we want to run the script on our Hadoop cluster, as opposed to running locally.

Listing 8.10 **Testing and running mrjob_simple_example.py**

```
# Run the mrjob script on our small sample data
> python mrjob_simple_example.py < birth_data_sample.txt

"201001"    8701
"201002"    8155
"201003"    8976
# etc...

# Run the mrjob script on an existing Hadoop cluster
# Ensure HADOOP_HOME environment variable is set
# (This setting may differ from your implementation)
> export HADOOP_HOME=/usr/local/hadoop-0.20.2/

# Specify that mrjob use Hadoop on the command line
> python mrjob_simple_example.py \
    -r hadoop hdfs:///user/hduser/data/VS2010NATL.DETAILUS.PUB
```

Building a Multistep Pipeline

Although some large data processing tasks can be accomplished using a single MapReduce step, a common pattern is to run a job that takes multiple steps. In other cases, two separate sources will use two mapper functions to emit data containing the same key. Then a single reducer phase will combine the data based on the matching key into a combined output.

To illustrate this type of pipeline, let's extend our original birth count example by adding another step. Let's count only the female births that happened per month in 2010. Before determining the number of births per month, let's filter each record by gender. To do this, let's add a new mapper function called `filter_births_by_gender`. This mapper will emit a value only if the birth record is female. The key emitted will simply be an F, and the value will be the month and year of the birth. The output of this mapper will be fed into another mapper function called `counter_mapper`, which will assign a 1 to the month–year key (just as we did in the single-step example). Finally, the output from this mapper function will be fed into our original reducer, `sum_births`, which will emit the total number of female births per month.

To specify the order in which we want the mappers to run, we will overload the `MrJob.steps` method to return an array containing each individual map and reduce phase in order. See Listing 8.11 for the code for each of these steps.

Listing 8.11 **A two-step MapReduce mrjob script**

```
mrjob_multistep_example.py

from mrjob.job import MRJob

class MRFemaleBirthCounter(MRJob):

  def filter_births_by_gender(self, key, record):
    if record[435] == 'F':
      year = record[14:18]
      month = record[18:20]
      birth_month = '%s-%s' % (month, year)
      yield 'Female', birth_month

  def counter_mapper(self, gender, month):
    yield '%s %s' % (gender,month), 1

  def sum_births(self, month, births):
        yield month, sum(births)

  def steps(self):
    return [self.mr(mapper=self.filter_births_by_gender),
            self.mr(mapper=self.counter_mapper,
                    reducer=self.sum_births)]
```

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

It's time to test our script using the slice of test data we created earlier. Multistep MapReduce jobs introduce more complexity than single-step jobs, with more opportunity for something to go wrong. Fortunately, a cool feature of mrjob is the ability to specify a particular step to run using the `step-num` flag on the command line. This is a useful way to perform a quick sanity check on a section of our pipeline. As shown in Listing 8.12, we can specify that we want the script to send the output of the first mapper step (zero-based) to stdout. Just as before, we can run the entire pipeline on our test data.

Listing 8.12 **Testing the mrjob_multistep_example.py script locally**

```
# Test the output of the mapper from the first step
> python mrjob_multistep_example.py --mapper \
    --step-num=0 < birth_data_big.txt

"F"     "10-2010"
"F"     "11-2010"
"F"     "09-2010"
"F"     "10-2010"
# etc...

# Test the output of the entire pipeline
> python mrjob_multistep_example.py < birth_data_sample.txt

"Female 01-2010"     4285
"Female 02-2010"     4002
"Female 03-2010"     4365
"Female 04-2010"     4144
```

Running mrjob Scripts on Elastic MapReduce

One of the core principles of this book is that data processing solutions should avoid managing hardware and infrastructure whenever it is practical and affordable. A great feature of mrjob is the ability to easily run MapReduce jobs using Amazon's Elastic MapReduce (EMR) service.

In order to take advantage of EMR integration, first create an Amazon Web Services account and sign up for the Elastic MapReduce service. Once these are created, you will also note (but don't share!) the Access Key ID and the corresponding Secret Access Key (under the Security Credentials section of the AWS accounts page).

With these Access- and Secret-Key values, set the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables on the machine on which your mrjob script is hosted. It is also possible to add many of these values to an *mrjobs.conf* configuration file stored in the user's home directory. Once this is set up, kick off your MapReduce job by running your mrjob script with the `-r` flag set to `emr`. It is also possible to specify the number and type of EC2 instances to use for your MapReduce job. See Listing 8.13 for an example.

Listing 8.13 **Using mrjobs with Elastic MapReduce**

```
# Set the Access Key ID and Secret Access Key Environment Variables
> export AWS_ACCESS_KEY_ID=XXXACCESSKEYHEREXXX
> export AWS_SECRET_ACCESS_KEY=XXXSECRETKEYHEREXXX

# Start an Elastic MapReduce job with 4 small instances
> python your_mr_job_sub_class.py -r emr \
  --ec2_instance_type c1.small --num-ec2-instances 4
```

Alternative Python-Based MapReduce Frameworks

mrjob is not the only Python-based MapReduce framework for Hadoop streaming. Another popular framework is **Dumbo**. Dumbo is similar to mrjob in many ways, and for simple MapReduce tasks, the structure of a Dumbo script is fairly similar to one using mrjob. In my opinion, one of the strongest differentiators is in mrjob's EMR integration. Because Yelp runs a lot of data processing tasks in the Amazon cloud, kicking off a job on Elastic MapReduce is a bit easier using mrjob than it is when using Dumbo. To its credit, Dumbo is probably a bit easier to use if you need to work with some type of custom data-input format.

Both mrjob and Dumbo are wrappers for the Hadoop streaming API, but they don't attempt to provide direct access to all the classes and methods found in the lower-level Hadoop Java API. For even more fine-grained access to these functions, it might be worth taking a look at Pydoop. Pydoop's goal is to be a Python interface for Hadoop and HDFS itself. By sidestepping Hadoop's streaming API, it *may* be more performant to use Pydoop over frameworks such as mrjob or Dumbo. If you are analyzing terabyte datasets, a 2X boost in performance might mean huge savings in overall processing time.

Summary

Dealing with the collection, processing, and analysis of large amounts of data requires specialized tools for each step. To solve this problem, it is necessary to build data processing pipelines to transform data from one state to another. Hadoop helps to make distribution of our pipeline task across a large number of machines accessible—but writing custom MapReduce jobs using the standard Hadoop API can be challenging. Many common, large-scale data processing tasks can be addressed using the Hadoop streaming API. Hadoop streaming scripts are a great solution for single-step jobs

- **Hadoop streaming does take binary or emit binary outputs (as it uses STDIN and STDOUT)**

- **MrJob can take text input; JSON and Pickle (??)REPR**
- **MrJob/Hadoop streaming is very inefficient**

- **Show graphs**
  - Text versus binary (byte encoded via serializable

Map Reduce Algorithm Design

- 7.1 Background and Motivation (5)
- 7.2 MrJob
  - Installation (5)
  - MrJob Fundamentals and Concepts (5)
  - Writing Mrjob code (10)
  - BLT Word frequency challenge (15)
  - Log file processing (10)
  - BLT log file processing challenge (15)
  - Serializable, JSON and other MrJob info (10)
- END of Lecture
- Clustering Algorithms
  - Clustering overview (10 mins)
  - Kmeans algorithm (5)
  - Distributed Kmeans in MrJob [ScreenFlow] (10 minutes)
    - Initialization (Canopy Clustering)
- Sync time
  - MrJob : cluster tweets
  - Model-based clustering [Sync time or some in lecture 5]
    - Intro [5]

V4

Installation of MrJob Windows/Mac/Linux

1. https://github.com/Yelp/mrjob/blob/master/mrjob/examples/mr_wc.py

- **Installation**
  - If you have Anaconda installed (Windows/Mac/Linux) then:
    - > *conda install mrjob*
  - Otherwise:
    - > *pip install mrjob*

Step thru Installer wizard

- Download a small script (wc word count) to test to verify the installation:
  1. https://github.com/Yelp/mrjob/blob/master/mrjob/examples/mr_wc.py
  2. On the command line run the test script **python** mr_wc.py mr_wc.py
  3. The last three lines of the output should be:

*"chars" ****
*"lines" ****
*"words" ****

Nice utility program: wc for characters,words, lines

MrJob Notebook for WordCount

- **MRWordCountUtility is a subclass of MrJob Class**

MrJob class code

In [ ]:

```
%%writefile mr_wc.py
"""An implementation of wc as an MRJob.
This is meant as an example of why mapper_final is useful."""
from mrjob.job import MRJob


class MRWordCountUtility(MRJob):

    def __init__(self, *args, **kwargs):
        super(MRWordCountUtility, self).__init__(*args, **kwargs)
        self.chars = 0
        self.words = 0
        self.lines = 0

    def mapper(self, _, line):
        # Don't actually yield anything for each line. Instead, collect them
        # and yield the sums when all lines have been processed. The results
        # will be collected by the reducer.
        self.chars += len(line) + 1  # +1 for newline
        self.words += sum(1 for word in line.split() if word.strip())
        self.lines += 1

    def mapper_final(self):
        yield('chars', self.chars)
        yield('words', self.words)
        yield('lines', self.lines)

    def reducer(self, key, values):
        yield(key, sum(values))


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

1. https://github.com/Yelp/mrjob/blob/master/mrjob/examples/mr_wc.py

MrJob WordCount test

Run the code in command line

```
1 !python mr_wc.py mr_wc.py
```

```
"chars" 981
"lines" 32
"words" 115
```

Pattern: reports back to the driver program

Run the code through python driver

```
1 from mr_wc import MRWordCountUtility
2 mr_job = MRWordCountUtility(args=['mr_wc.py'])
3 with mr_job.make_runner() as runner:
4     runner.run()
5     # stream_output: get access of the output
6     for line in runner.stream_output():
7         print mr_job.parse_output_line(line)
```

```
(u'chars', 981)
(u'lines', 32)
(u'words', 115)
```

MrJob WordCount Notebook

```
%%writefile mr_wc.py
"""An implementation of wc as an MRJob.
This is meant as an example of why mapper_final is useful."""
from mrjob.job import MRJob


class MRWordCountUtility(MRJob):

    def __init__(self, *args, **kwargs):
        super(MRWordCountUtility, self).__init__(*args, **kwargs)
        self.chars = 0
        self.words = 0
        self.lines = 0

    def mapper(self, _, line):
        # Don't actually yield anything for each line. Instead, collect them
        # and yield the sums when all lines have been processed. The results
        # will be collected by the reducer.
        self.chars += len(line) + 1  # +1 for newline
        self.words += sum(1 for word in line.split() if word.strip())
        self.lines += 1

    def mapper_final(self):
        yield('chars', self.chars)
        yield('words', self.words)
        yield('lines', self.lines)

    def reducer(self, key, values):
        yield(key, sum(values))


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

https://www.dropbox.com/s/lfhg2q3nj6kgk5n/MrjobMostUsedWord.ipynb?dl=0

Run the code in command line

```
!python mr_wc.py mr_wc.py
```

```
"chars" 981
"lines" 32
"words" 115
```

BLT: Install MRJob and verify

- **Run the MrJob WordCount Notebook available here as follows**
  1. **python** mr_wc.py mr_wc.py
  2. The last three lines of the output should be:

     *"chars" ****
     *"lines" ****
     *"words" ****
- **Note the file mr_wc.py file has been modified since we did this slide so please rerun to get the correct most uptodate output.**
- **How many words are in the file? Select the closest number.**
  - A: 183
  - B: 185
  - C: 209
  - D: 199

C is the correct answer

Map Reduce Algorithm Design

- 7.1 Background and Motivation (5)
- 7.2 MrJob
  - Installation (5)
  - MrJob Fundamentals and Concepts (5)
  - Writing Mrjob code (10)
  - BLT Word frequency challenge (15)
  - Log file processing (10)
  - BLT log file processing challenge (15)
  - Serializable, JSON and other MrJob info (10)
- END of Lecture
- Clustering Algorithms
  - Clustering overview (10 mins)
  - Kmeans algorithm (5)
  - Distributed Kmeans in MrJob [ScreenFlow] (10 minutes)
    - Initialization (Canopy Clustering)
- Sync time
  - MrJob : cluster tweets
  - Model-based clustering [Sync time or some in lecture 5]
    - Intro [5]

V4

Fundamentals and Concept

- **Mapreduce**
  - Mapper(): take a single record with key and value as input, and returns zero or more (key, value) pairs
  - Reducer(): take a key and a subset of the values for that key as input and returns zero or more (key, value) pairs.
  - Combiner(): take a key and the complete set of values for that key in the current step, and returns zero or more arbitrary (key, value) pairs as output.
    - This is a Mapper-side Reducer to alleviate network burden

- **mrjob is a Python 2.6+ package that helps you write and run Hadoop Streaming jobs.**

- Run jobs on EMR, your own Hadoop cluster, or locally (for testing).
- Write multi-step jobs (one map-reduce step feeds into the next)
- ***Duplicate your production environment inside Hadoop***
  - Upload your source tree and put it in your job's `$PYTHONPATH`
  - Run make and other setup scripts
  - Set environment variables (e.g. `$TZ`)
  - Easily install python packages from tarballs (EMR only)
  - Setup handled transparently by `mrjob.conf` config file
- Automatically interpret error logs from EMR
- SSH tunnel to hadoop job tracker on EMR
- ***Minimal setup***
  - To run on EMR, set `$AWS_ACCESS_KEY_ID` and `$AWS_SECRET_ACCESS_KEY`
  - To run on your Hadoop cluster, install `simplejson` and make sure `$HADOOP_HOME` is set.

https://pythonhosted.org/mrjob/

Setup and teardown of tasks

- **Remember from that your script is invoked once per task by Hadoop Streaming.**
- **It starts your script, feeds it stdin, reads its stdout, and closes it. mrjob lets you write methods to run at the beginning and end of this process: the *_init() and *_final() methods:**
  - mapper_init()
  - combiner_init()
  - reducer_init()
  - mapper_final()
  - combiner_final()
  - reducer_final()

Yield as a lazy function: allocate memory

Yield

`Yield` is a keyword that is used like `return`, except the function will return a generator.

```
>>> def createGenerator():
...     mylist = range(3)
...     for i in mylist:
...         yield i*i
...
>>> mygenerator = createGenerator() # create a generator
>>> print(mygenerator) # mygenerator is an object!
<generator object createGenerator at 0xb7555c34>
>>> for i in mygenerator:
...     print(i)
0
1
4
```

Here it's a useless example, but it's handy when you know your function will return a huge set of values that you will only need to read once.

To master `yield`, you must understand that **when you call the function, the code you have written in the function body does not run.** The function only returns the generator object, this is a bit tricky :-)

Then, your code will be run each time the `for` uses the generator.

WordCount with an in-memory Mapper

-

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

class MRWordFreqCount(MRJob):

    def init_get_words(self):
        self.words = {}

    def get_words(self, _, line):
        for word in WORD_RE.findall(line):
            word = word.lower()
            self.words.setdefault(word, 0)
            self.words[word] = self.words[word] + 1

    def final_get_words(self):
        for word, val in self.words.iteritems():
            yield word, val

    def sum_words(self, word, counts):
        yield word, sum(counts)

    def steps(self):
        return [MRStep(mapper_init=self.init_get_words,
                       mapper=self.get_words,
                       mapper_final=self.final_get_words,
                       combiner=self.sum_words,
                       reducer=self.sum_words)]
```

WordCount with an in-memory Mapper

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

class MRWordFreqCount(MRJob):

    def init_get_words(self):
        self.words = {}

    def get_words(self, _, line):
        for word in WORD_RE.findall(line):
            word = word.lower()
            self.words.setdefault(word, 0)
            self.words[word] = self.words[word] + 1

    def final_get_words(self):
        for word, val in self.words.iteritems():
            yield word, val

    def sum_words(self, word, counts):
        yield word, sum(counts)

    def steps(self):
        return [MRStep(mapper_init=self.init_get_words,
                       mapper=self.get_words,
                       mapper_final=self.final_get_words,
                       combiner=self.sum_words,
                       reducer=self.sum_words)]
```

Mr Job Parent class

Mapper Init

Mapper

Mapper final

Reducer

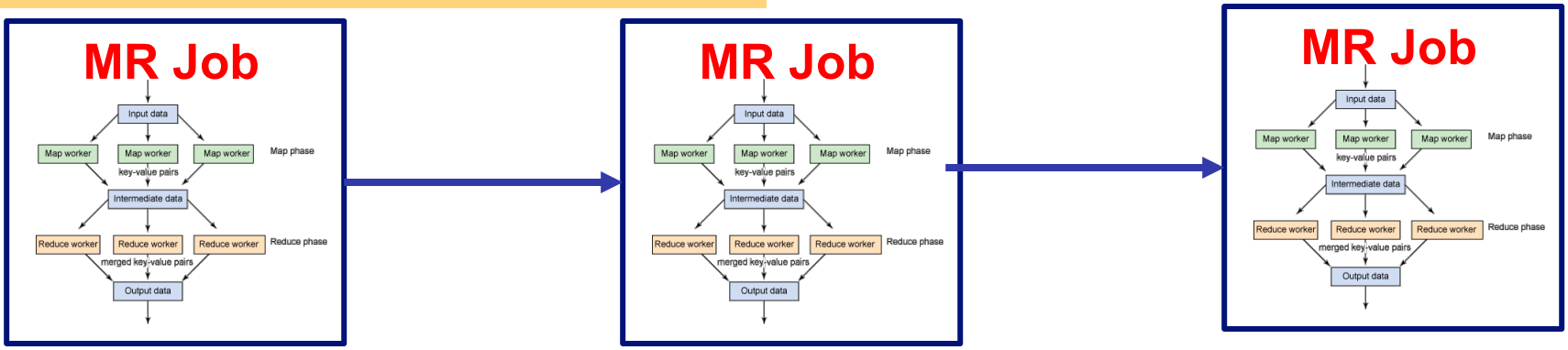
Steps

Fundamentals and Concept

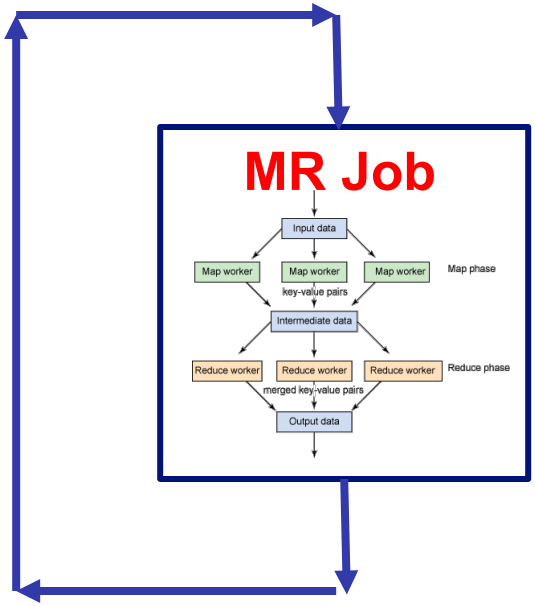
- **Mrjob**
  - Mrjob class
  - steps(): Configure Mrjob steps
  - mapper(), combiner(), reducer(),
  - mapper_init(), combiner_init(), reducer_init(),
  - mapper_final(), combiner_final(), reducer_final(), mapper_cmd(), combiner_cmd(), reducer_cmd(), mapper_pre_filter(), combiner_pre_filter(), reducer_pre_filter()
  - Write your own steps()
  - Object Oriented Programing: Inheritance and override (See Mrjob examples below)

Pipeline of jobs

Pipeline of jobs in sequence



Iterative Pipeline



Fundamentals and Concept

- **Algorithm with Loops in Mrjob**
  - Loop through step function

```
def steps(self):
    string1 = "a = [MRJobStep(mapper=self.mapper_FirstKM,reducer=self.reducer_FirstKM)"
    string2 = ",MRJobStep(mapper=self.mapper_KM,reducer=self.reducer_KM),MRJobStep(reducer=self.reducer_KM2)"*int(self.options.Iterations)
    stringall = string1+string2+"]"
    exec stringall
    return a
```

  - Call Mrjob from outside, More flexible

```
from time import strftime, time, localtime
def runJob(MRJobClass, argsArr, loc='local'):
    if loc == 'emr':
        argsArr.extend(['-r', 'emr'])

    startTime=time()
    print "%s starting %s job on %s" % (strftime("%a, %d %b %Y %H:%M:::%S", localtime()), MRJobClass.__name__, loc)
    mrJob = MRJobClass(args=argsArr)
    runner = mrJob.make_runner()
    runner.run()

    endTime=time()
    jobLength=endTime-startTime
    print "%s finished %s job in %d seconds" % (strftime("%a, %d %b %Y %H:%M:::%S", localtime()), MRJobClass.__name__, jobLength)
    return mrJob, runner

def runParallelJob(MRJobClass, argsArr):            #TO DO: add threading to allow jobs to run in
    pass                                                #parallel
    #launch a new thread
    #call runJob(MRJobClass, argsArr) on the new thread

for i in range (10):
    runJob(MrKmeansCanopyIteration, ['%sCanopiedTrainingData' % cwd, '--centroidsFile=%skMeansCentroids0.txt'%cwd])
    if clustersDelta() < tol:
        return
```

mrjob is a Python 2.6+ package that helps you write and run Hadoop Streaming jobs

MrJob: local/cloud; Text or Pickle objects

- ## *Hadoop with Python - mrjob*

  - Mapper, reducer written as functions
  - Can serialize (Pickle) objects to use as values
  - Presents a single key + all values at once
  - Extracts map/reduce errors from Hadoop for you
  - Hadoop runs entirely through Python:

```
$ ./wordcount-mrjob.py \
      --jobconf mapred.reduce.tasks=2 \
      -r hadoop \
      hdfs:///user/glock/mobydick.txt \
      --output-dir hdfs:///user/glock/output
```

Fundamentals and Concept

- **Running Job from command line**
  - Passing multiple input files

    $python my_job.py input1.txt input2.txt
  - Passing multiple input files with stand imput

    $python my_job.py input1.txt input2.txt - < input3.txt
  - Output file

    $python my_job.py input1.txt > output.txt
  - Running Mode

    -r inline(default), -r local, -r hadoop, -r emr

    $python my_job.py -r emr s3://my-inputs/input.txt

    $python my_job.py -r hadoop hdfs://my_home/input.txt

Running MrJobs on the cloud (e.g., EMR)

mrjob v0.4.2 documentation

Table Of Contents

Need help?

Join the mailing list by visiting the Google group page or sending an email to mrjob+subscribe@googlegroups.com.

type to search

Elastic MapReduce Quickstart

Configuring AWS credentials

Configuring your AWS credentials allows mrjob to run your jobs on Elastic MapReduce and use S3.

- Create an Amazon Web Services account
- Sign up for Elastic MapReduce
- Get your access and secret keys (click "Security Credentials" on your account page)

Now you can either set the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`, or set **aws_access_key_id** and **aws_secret_access_key** in your `mrjob.conf` file like this:

```
runners:
  emr:
    aws_access_key_id: <your key ID>
    aws_secret_access_key: <your secret>
```

Configuring SSH credentials

Configuring your SSH credentials lets mrjob open an SSH tunnel to your jobs' master nodes to view live progress, see the job tracker in your browser, and fetch error logs quickly.

- Go to https://console.aws.amazon.com/ec2/home
- Make sure the **Region** dropdown (upper left) matches the region you want to run jobs in (usually "US East").
- Click on **Key Pairs** (lower left)
- Click on **Create Key Pair** (center).
- Name your key pair `EMR` (any name will work but that's what we're using in this example)
- Save `EMR.pem` wherever you like (`~/.ssh` is a good place)
- Run `chmod og-rwx /path/to/EMR.pem` so that `ssh` will be happy
- Add the following entries to your `mrjob.conf`:

```
runners:
  emr:
```

EMR and MrJob: Homework Challenge

How To

```
python code/unique_review.py -r emr -c mrjob.conf s3://i290-jblomo/data/selected_reviews.json.gz
```

- Simply specify `-r emr`

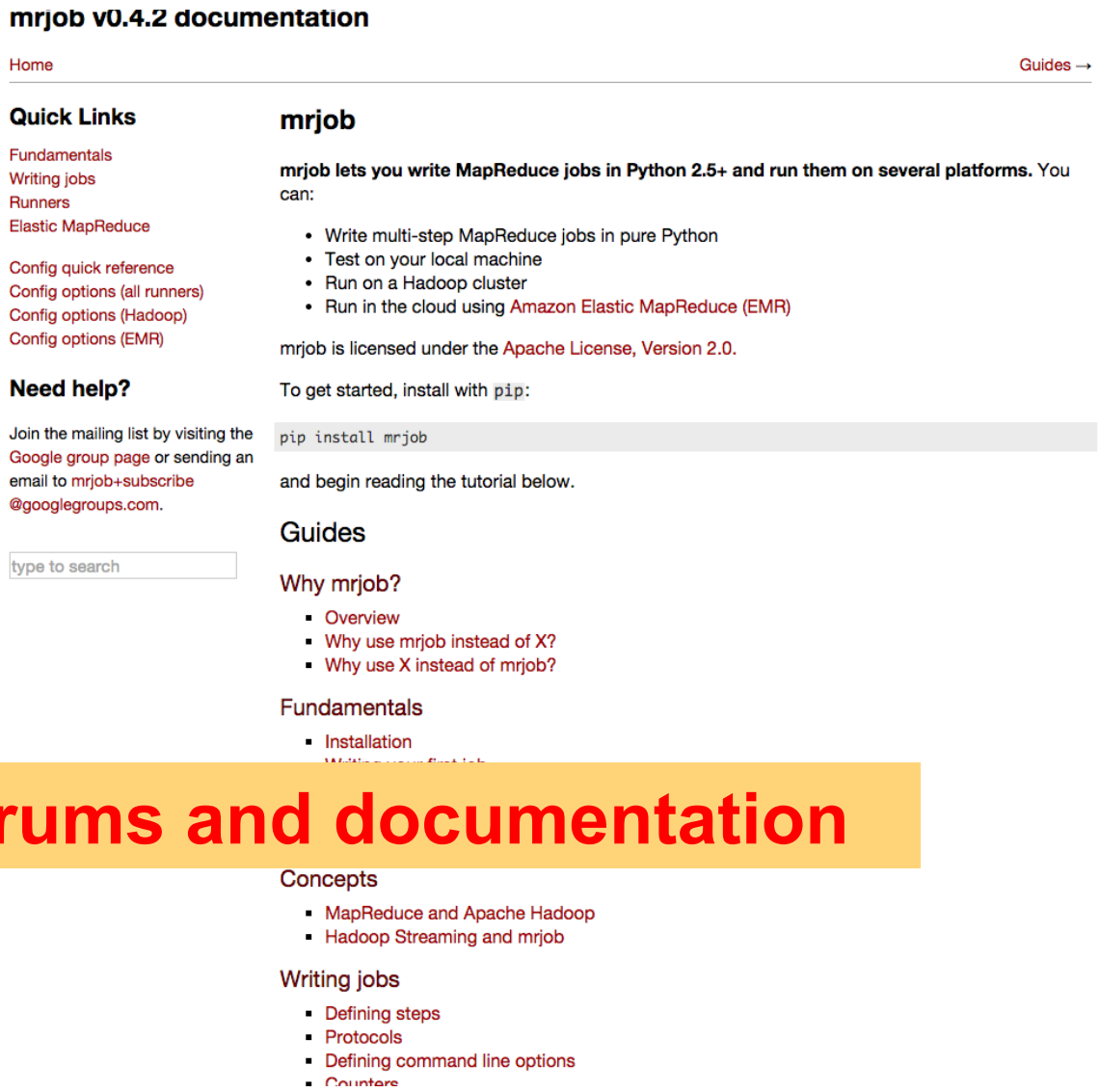
mrjob Workflow

- Develop locally on subset of data
- Run on EMR on all data located in S3 bucket
- Either save output in S3, or stream locally

ToDo get code and yelp/selected_r

MrJob Online forums and documentations

- https://pythonhosted.org/mrjob/



mrjob v0.4.2 documentation

Home Guides →

Quick Links

Fundamentals
Writing jobs
Runners
Elastic MapReduce

Config quick reference
Config options (all runners)
Config options (Hadoop)
Config options (EMR)

Need help?

Join the mailing list by visiting the Google group page or sending an email to mrjob+subscribe @googlegroups.com.

type to search

mrjob

mrjob lets you write MapReduce jobs in Python 2.5+ and run them on several platforms. You can:

- Write multi-step MapReduce jobs in pure Python
- Test on your local machine
- Run on a Hadoop cluster
- Run in the cloud using Amazon Elastic MapReduce (EMR)

mrjob is licensed under the Apache License, Version 2.0.

To get started, install with `pip`:

```
pip install mrjob
```

and begin reading the tutorial below.

Guides

Why mrjob?

- Overview
- Why use mrjob instead of X?
- Why use X instead of mrjob?

Fundamentals

- Installation

Concepts

- MapReduce and Apache Hadoop
- Hadoop Streaming and mrjob

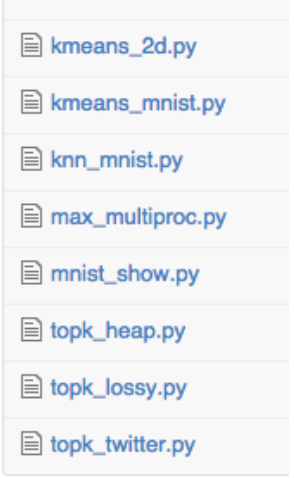
Writing jobs

- Defining steps
- Protocols
- Defining command line options

Online forums and documentation

MrJob Help Pages

- **MrJob Manual**
  - https://pythonhosted.org/mrjob/
- **The mrjob documentation has many examples**
  - Writing jobs in Python)
- **Other cool examples**
  - https://github.com/uchicago-cs/cmsc12300/tree/master/examples/data_analysis/bin



https://pythonhosted.org/mrjob/

- Why mrjob?
  - Overview
  - Why use mrjob instead of X?
  - Why use X instead of mrjob?
- Fundamentals
  - Installation
  - Writing your first job
  - Running your job different ways
  - Writing your second job
  - Configuration
- Concepts
  - MapReduce and Apache Hadoop
  - Hadoop Streaming and mrjob
- Writing jobs
  - Defining steps
  - Protocols
  - Defining command line options
  - Counters
- Runners
  - Testing locally
  - Running on your own Hadoop cluster
  - Running on EMR
  - Configuration
  - Running your job programmatically
- Config file format and location
  - Precedence and combining options
  - Option data types
  - Using multiple config files
- Options available to all runners
  - Making files available to tasks
  - Temp files and cleanup
  - Job execution context
  - Other
  - Options ignored by the local and inline runners
  - Options ignored by the inline runner

- https://groups.google.com/forum/#!topic/mrjob/rmUzy-SerQ4

```
#-----------------------------------------
# simple_mrjob.py
import json

from mrjob.job import MRJob

class PlatformCounter(MRJob):
    def mapper(self, key, line):
        line = json.loads(line.strip())
        yield line['platform'], 1

    def reducer(self, word, occurrences):
        print 'in reducer {0}  occurrences'.format(occurrences)
        yield word, sum(occurrences)
#-----------------------------------------
# simple_runner.py
from simple_mrjob import PlatformCounter
from runJob import runJob

runJob(PlatformCounter, ['hdfs:///user/hadoop/logs', '--output-dir=result-mrjob-args'], 'hadoop')
#-----------------------------------------
```

And next is the output of executing simple_runner.py:

```
hadoop@ip-10-34-139-215:~/vlad/pythonmr/test_mrjob$ python simple_runner.py
starting PlatformCounter job on hadoop
Traceback (most recent call last):
  File "simple_runner.py", line 10, in <module>
```

Map Reduce Algorithm Design

- 7.1 Background and Motivation (5)
- 7.2 MrJob
  - Installation (5)
  - MrJob Fundamentals and Concepts (5)
  - Writing Mrjob code (10)
  - BLT Word frequency challenge (15)
  - Log file processing (10)
  - BLT log file processing challenge (15)
  - Serializable, JSON and other MrJob info (10)
- END of Lecture
- Clustering Algorithms
  - Clustering overview (10 mins)
  - Kmeans algorithm (5)
  - Distributed Kmeans in MrJob [ScreenFlow] (10 minutes)
    - Initialization (Canopy Clustering)
- Sync time
  - MrJob : cluster tweets
  - Model-based clustering [Sync time or some in lecture 5]
    - Intro [5]

V4

Writing Mrjob code

- **Example 1: WordCount**
  - Count how often words occur
  - "Helloworld" of MapReduce Coding
  - Mapper
    - Read data from InputFile
    - Split it into words
    - Output <word, 1 > tuples
  - Combiner
    - Sum the occurrences of each word to a count
    - Output its results <word, LocalSum > tuples
  - Reducer
    - Read the results of Mapper
    - Sum the occurrences of each word to a final count
    - Output its results <word, Sum > tuples

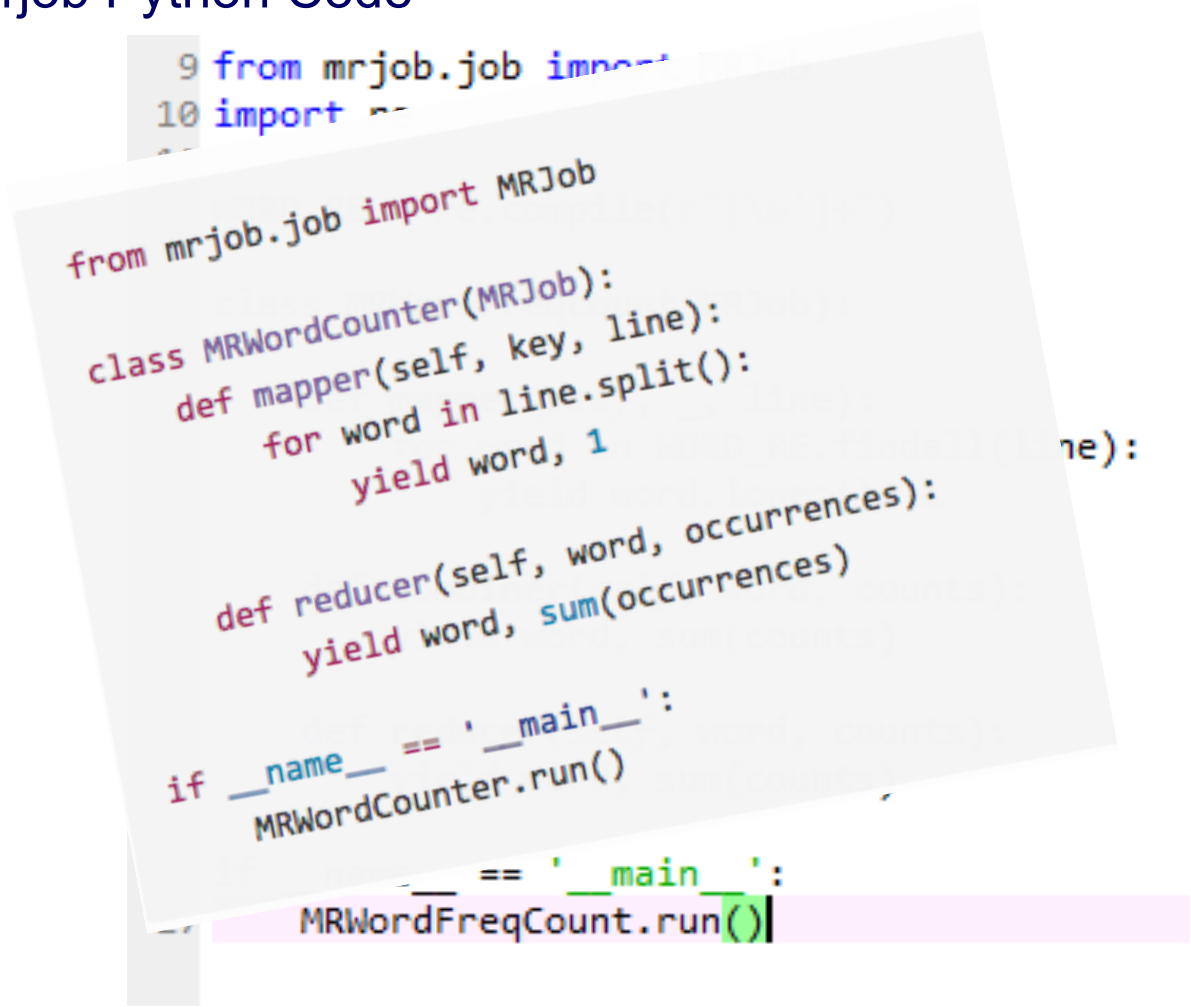
Hello MrJob

- **WordCount Example**
  - Mrjob Python Code

```
 9 from mrjob.job import MRJob
10 import re
11
12 WORD_RE = re.compile(r"[\w']+")
13
14 class MRWordFreqCount(MRJob):
15
16     def mapper(self, _, line):
17         for word in WORD_RE.findall(line):
18             yield word.lower(), 1
19
20     def combiner(self, word, counts):
21         yield word, sum(counts)
22
23     def reducer(self, word, counts):
24         yield word, sum(counts)
25
26 if __name__ == '__main__':
27     MRWordFreqCount.run()
```

Hello MrJob

- **WordCount Example**
  - Mrjob Python Code



WordCount Mapper: MrJob vs. Hadoop Streaming

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

from mrjob.job import MRJob

class MRwordcount(MRJob):

    def mapper(self, _, line):
        line = line.strip()
        keys = line.split()
        for key in keys:
            value = 1
            yield key, value

    def reducer(self, key, values):
        yield key, sum(values)

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

Hadoop Streaming requires managing I/O etc.

```python
for line in sys.stdin:
    line = line.strip()
    keys = line.split()
    for key in keys:
        value = 1
        print('%s\t%d' % (key, value))
```

WordCount Reducer: MrJob vs. Hadoop Streaming

mrjob - Reducer

```
    def mapper(self, _, line):
        line = line.strip()
        keys = line.split()
        for key in keys:
            value = 1
            yield key, value

    def reducer(self, key, values):
        yield key, sum(values)

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

- Reducer gets one key and ALL values
- No need to loop through key/value pairs
- Use list methods/ iterators to deal with keys

Writing Mrjob code

- **Example 1: WordCount (Cont.)**

  Brief Introduction of Yield:

Generally speaking, iterators and generators (functions that create iterators, for example with Python's `yield` statement) have the advantage that an element of a sequence is not produced until you actually need it. This can help a lot in terms of computational expensiveness or memory consumption depending on the task at hand.

```
8  def TestGenerator(l):
9    for e in l:
10    print ("before yield:" + str(e))
11    yield e
12    print ("after yield:" + str(e))
13
14 for el in TestGenerator([6,7,8,9]):
15   print (el)
```

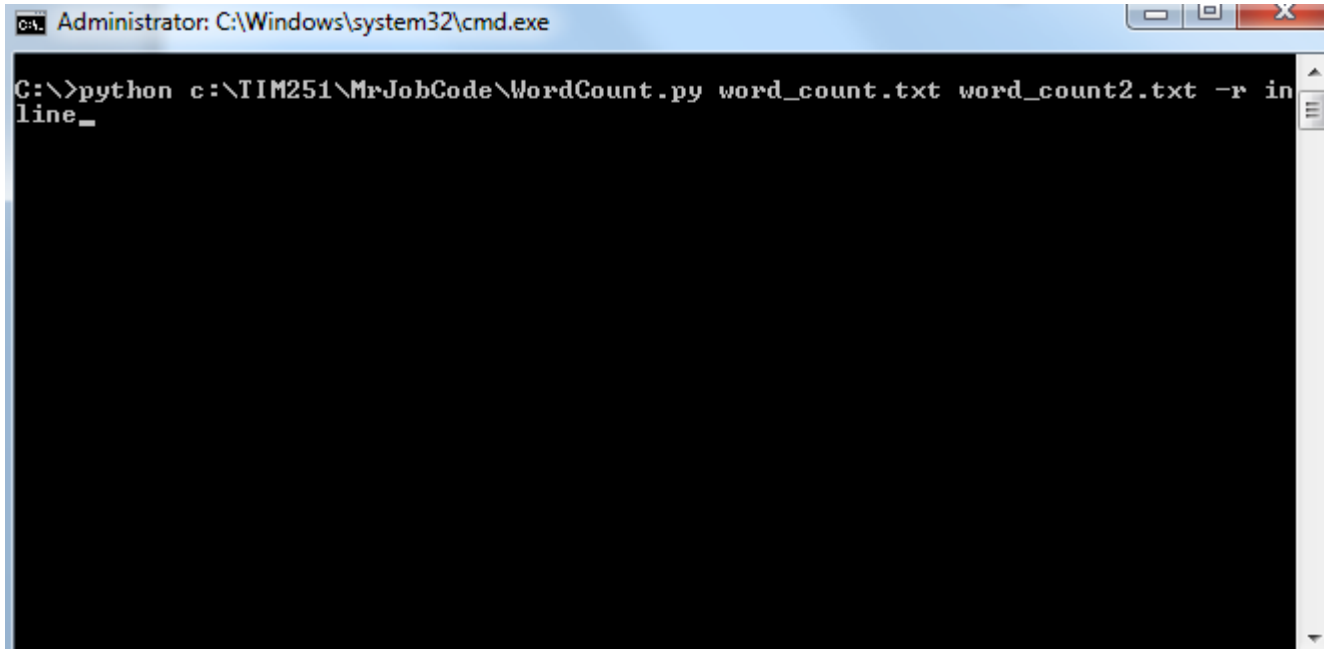
```
before yield:6
6
after yield:6
before yield:7
7
after yield:7
before yield:8
8
after yield:8
before yield:9
9
after yield:9
```

Running Mrjob code on command line

- **Example 1: WordCount (Cont.)**

  Command Line:

  $python MRWordFreqCount.py inputfile.txt

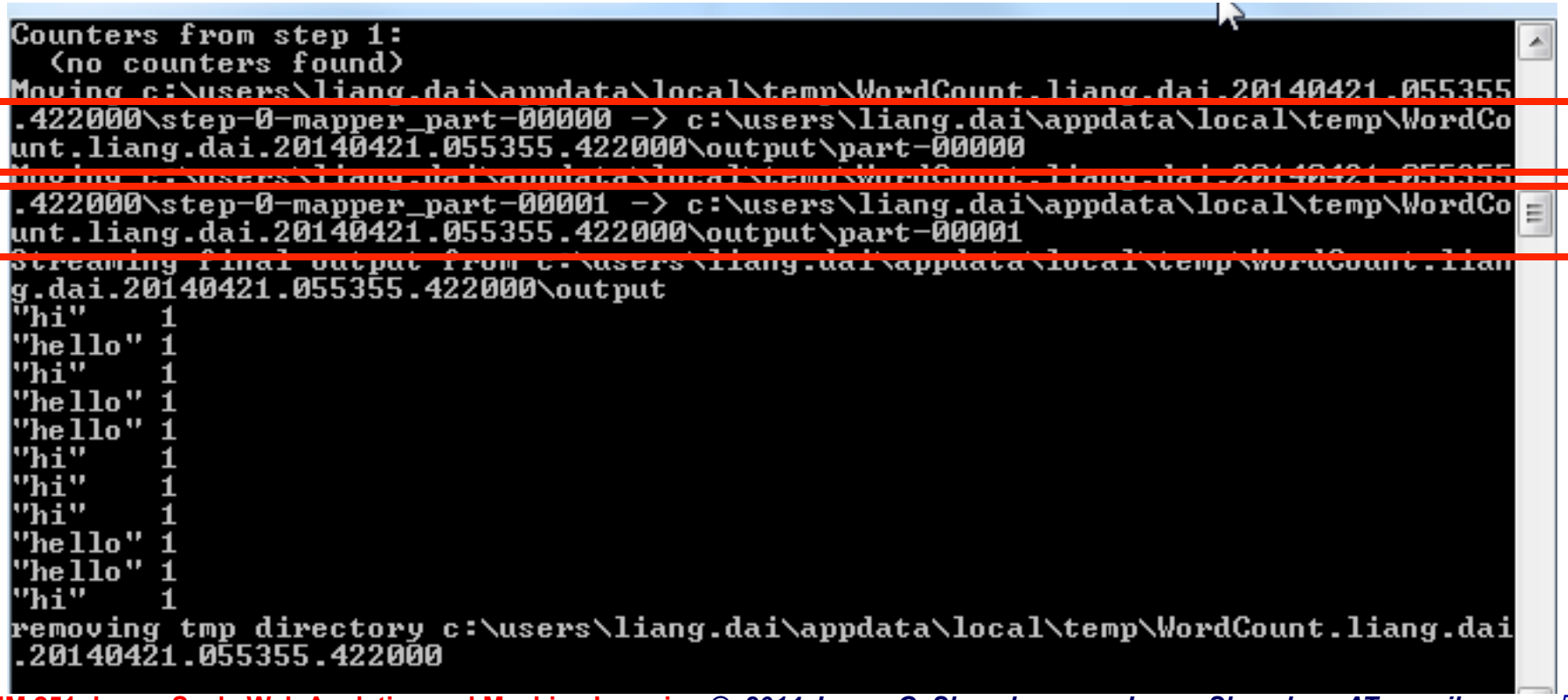


Runing Mrjob code: 2 output PART files

- **Example 1: WordCount (Cont.)**

  Result: partial results

  Notice two output partitions (two reducers are running)



Writing code in Mrjob: key development steps

- **key development steps using WordCount Example**
  - Write mapper and reducer functions for word count
  - What does each step (mapper, reducer, combine) do?
  - Verify mapper, reducer, combine steps
    - Let's check it step by step with two input files.
    - By Mrjob default setting, two mappers will be assigned. It is also a good way for debugging.

Writing Mrjob code with no Combiner: test mapper only

- **Example 1: WordCount (Cont.)**

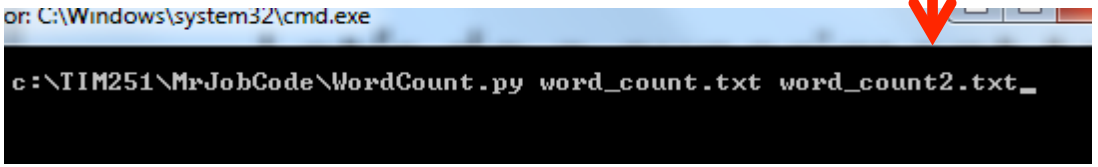
  Just use a Mapper

  Debug tactic!

```
9  from mrjob.job import MRJob
10 import re
11
12 WORD_RE = re.compile(r"[\w']+")
13
14 class MRWordFreqCount(MRJob):
15
16     def mapper(self, _, line):
17         for word in WORD_RE.findall(line):
18             yield word.lower(), 1
19 '''
20     def combiner(self, word, counts):
21         yield word, sum(counts)
22
23     def reducer(self, word, counts):
24         yield word, sum(counts)
25 '''
26 if __name__ == '__main__':
27     MRWordFreqCount.run()
```

Each mapper output word, 1

Two Inputs




```
Counters from step 1:
  (no counters found)
Moving c:\users\liang.dai\appdata\local\temp\WordCount.liang.d
.422000\step-0-mapper_part-00000 -> c:\users\liang.dai\appdata
unt.liang.dai.20140421.055355.422000\output\part-00000
Moving c:\users\liang.dai\appdata\local\temp\WordCount.liang.d
.422000\step-0-mapper_part-00001 -> c:\users\liang.dai\appdata
unt.liang.dai.20140421.055355.422000\output\part-00001
Streaming final output from c:\users\liang.dai\appdata\local\t
g.dai.20140421.055355.422000\output
"hi"    1
"hello" 1
"hi"    1
"hello" 1
"hello" 1
"hi"    1
"hi"    1
"hi"    1
"hello" 1
"hello" 1
"hi"    1
removing tmp directory c:\users\liang.dai\appdata\local\temp\W
.20140421.055355.422000
```

Writing Mrjob code with Combiner: Test Combiner output

- **Example 1: WordCount (Cont.)**

  Let's check combiner (and no reducer!)

  Good debug tactic!!

```
 9 from mrjob.job import MRJob
10 import re
11
12 WORD_RE = re.compile(r"[\w']+")
13
14 class MRWordFreqCount(MRJob):
15
16     def mapper(self, _, line):
17         for word in WORD_RE.findall(line):
18             yield word.lower(), 1
19
20     def combiner(self, word, counts):
21         yield word, sum(counts)
22
23    # def reducer(self, word, counts):
24    #     yield word, sum(counts)
25
26 if __name__ == '__main__':
27     MRWordFreqCount.run()
```

```
Administrator: C:\Windows\system32\cmd.exe

C:\>python c:\TIM251\MrJobCode\WordCount.py word_count.txt word_count2.txt_
```

```
Administrator: C:\Windows\system32\cmd.exe

C:\>python c:\TIM251\MrJobCode\WordCount.p
no configs found; falling back on auto-con
no configs found; falling back on auto-con
creating tmp directory c:\users\liang.dai\
.20140421.053209.981000
writing to step-0-mapper_part-00000
writing to step-0-mapper_part-00001
Counters from step 1:
  (no counters found)
Moving c:\users\liang.dai\appdata\local\te
.981000\step-0-mapper_part-00000 -> c:\use
unt.liang.dai.20140421.053209.981000\outpu
Moving c:\users\liang.dai\appdata\local\te
.981000\step-0-mapper_part-00001 -> c:\use
unt.liang.dai.20140421.053209.981000\outpu
Streaming final output from c:\users\liang
g.dai.20140421.053209.981000\output
"hello" 3
"hi"    4
"hello" 2
"hi"    2
removing tmp directory c:\users\liang.dai\
.20140421.053209.981000
```

```
Counters from step 1:
Streaming final output
"hi"    1
"hello" 1
"hi"    1
"hello" 1
"hello" 1
"hi"    1
"hi"    1
"hi"    1
"hello" 1
"hello" 1
"hi"    1
removing tmp directory
.20140421.055355.422000
```

Mapper only output

Combiner is a mapper-side reducer (two mappers as the word "hello" occurs twice in the output stream)

Writing Mrjob code: test reducer

- **Example 1: WordCount (Cont.)**

  Let's check reducer:

2 Reducers

```
9  from mrjob.job import MRJob
10 import re
11
12 WORD_RE = re.compile(r"[\w']+")
13
14 class MRWordFreqCount(MRJob):
15
16     def mapper(self, _, line):
17         for word in WORD_RE.findall(line):
18             yield word.lower(), 1
19
20     def combiner(self, word, counts):
21         yield word, sum(counts)
22
23     def reducer(self, word, counts):
24         yield word, sum(counts)
25
26 if __name__ == '__main__':
27     MRWordFreqCount.run()
```

```
Administrator: C:\Windows\system32\cmd.exe

C:\>python c:\TIM251\MrJobCode\WordCount.py word_count.txt word_count2.txt_
```

```
writing to step-0-mapper_part-00000
writing to step-0-mapper_part-00001
Counters from step 1:
  (no counters found)
writing to c:\users\liang.dai\appdata\local\temp\WordCount.liang.dai.20140421.05
3402.399000\step-0-mapper-sorted
> sort 'c:\users\liang.dai\appdata\local\temp\WordCount.liang.dai.20140421.05340
2.399000\step-0-mapper_part-00000' 'c:\users\liang.dai\appdata\local\temp\WordCo
unt.liang.dai.20140421.053402.399000\step-0-mapp
Piping files into sort for Windows compatibility
> sort
writing to step-0-reducer_part-00000
Counters from step 1:
  (no counters found)
Moving c:\users\liang.dai\appdata\local\temp\Wor
.399000\step-0-reducer_part-00000 -> c:\users\li
ount.liang.dai.20140421.053402.399000\output\par
Streaming final output from c:\users\liang.dai\a
g.dai.20140421.053402.399000\output
"hello" 5
"hi"    6
removing tmp directory c:\users\liang.dai\appdat
.20140421.053402.399000
```

```
"hello" 3
"hi"    4
"hello" 2
"hi"    2
removing tmp direc
.20140421.053209.9
```

Cimbiner only output

Reducer combines the results from the mappers and combiners

Multistep MR Jobs

Multi-Step

- Not all computations can be done in a single MapReduce step
- Map Input: <key, value>
- Reducer Output: <key, value>
- Compose MapReduce steps!

Output as Input

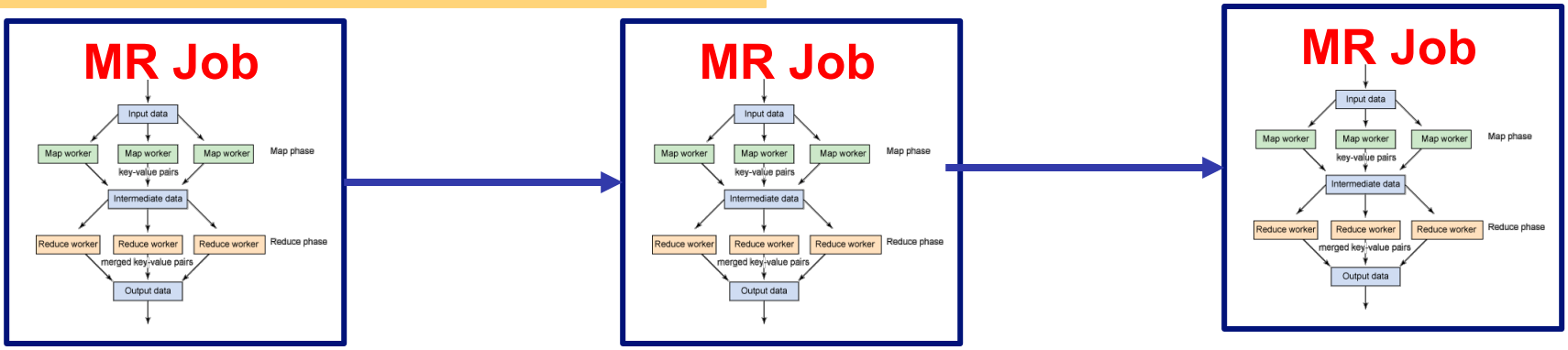
- The output of one MapReduce job can be used as the input to another

Examples

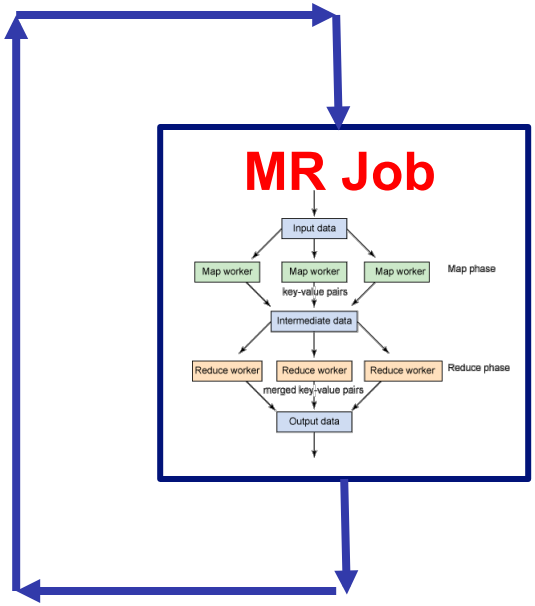
- PageRank: Multiple steps till solution converges
- Multi-level summaries

Pipeline of jobs

Pipeline of jobs in sequence



Iterative Pipeline



Example: find most used word

- **Example 2: find most frequent word**
  - 1st MapReduce Job
    - Mapper
      - Read data from InputFile
      - Split it into words
      - Output <word, 1 > tuples
    - Reducer
      - Read the results of Mapper
      - Sum the occurrences of each word to a final count
      - Output its results <word, Sum > tuples to
  - 2nd MapReduce Job
    - Mapper
      - Nothing to do
    - Reducer
      - Select the maximum count tuples

Writing Mrjob code

- **Example 2: MostUsedWord (Cont.)**

  Command Line:

  $python MostUsedWord.py inputfile.txt

Example: find most used word

Wrtie some words to MostUsedWord file

Generate data

```
!echo foo foo quux labs foo bar quux hell > MostUsedWord.txt
!echo labs labs hello fool labs >>MostUsedWord.txt
!echo california new york LA San Francisco >>MostUsedWord.txt
```

Write MostusedWord MrJob class

Need to override steps() function
It has two MapReduce Jobs.

```
%%writefile Mostused.py
from mrjob.job import MRJob
from mrjob.step import MRJobStep
import re
WORD_RE = re.compile(r"[\w']+")
class MRMostUsedWord(MRJob):
    def steps(self):
        return [
            MRJobStep(mapper=self.mapper_get_words,
                      combiner=self.combiner_count_words,
                      reducer=self.reducer_count_words),
            MRJobStep(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 have seen so far
        yield (word, sum(counts))
    def reducer_count_words(self, word, counts):
        # send all (num_occurrences,work) paris to the same reducer.
        # num_occurrences is the key, then we can easily use max funciton to get most used word
        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()
```

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

Step1:Job1

Step2: Job2

Word Count job

WordCount Yield(count, word

Reducer 2: Max

```
Overwriting Mostused.py
```

It has two mapreduce. The first Mapreduce Mapper outputs (word, 1) key value pairs, and then conbiner combines the sum locally. At last, Reducer sums them up. The second mapreduce output the word having max number of count.

Run the code in command line

```
!python Mostused.py MostUsedWord.txt
```

```
4       "labs"
```

This works well when you have a single reducer BUT not for mulitple reducers

Example: find most used word

Wrtie some words to MostUsedWord file

```
!echo foo foo quux labs foo bar quux hell > MostUsedWord.txt
!echo labs labs hello fool labs >>MostUsedWord.txt
!echo california new york LA San Francisco >>MostUsedWord.txt
```

Write MostusedWord MrJob class

```
%%writefile Mostused.py
from mrjob.job import MRJob
from mrjob.step import MRJobStep
import re
WORD_RE = re.compile(r"[\w']+")
class MRMostUsedWord(MRJob):
    def steps(self):
        return [
            MRJobStep(mapper=self.mapper_get_words,
               combiner=self.combiner_count_words,
               reducer=self.reducer_count_words),
            MRJobStep(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 have seen so far
        yield (word, sum(counts))
    def reducer_count_words(self, word, counts):
        # send all (num_occurrences,work) paris to the same reducer.
        # num_occurrences is the key, then we can easily use max funciton to get most used word
        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()
```

```
Overwriting Mostused.py
```

It has two mapreduce. The first Mapreduce Mapper outputs (word, 1) key value pairs, second mapreduce output the word having max number of count.

Run this program from the command line
The most frequent word is “labs”

Run the code in command line

```
!python Mostused.py MostUsedWord.txt
```

```
4	"labs"
```


Run the code in command line

```
!python Mostused.py MostUsedWord.txt
```

```
4	"labs"
```

Running the Example: find most used word

Run the code in command line

```
1 !python Mostused.py MostUsedWord.txt
```

```
4       "labs"
```

Versus

- **Or run the driver from the Notebook and get the result**

Run the code though python driver

```
1 from Mostused import MRMostUsedWord
2 mr_job = MRMostUsedWord(args=['MostUsedWord.txt'])
3 with mr_job.make_runner() as runner:
4     runner.run()
5     # stream_output: get access of the output
6     for line in runner.stream_output():
7         print mr_job.parse_output_line(line)
```

```
(4, u'labs')
```

Puzzle: word frequency distribution

- **Let's take a look at another example**
  - instead of counting words overall let's now generate a frequency distribution of word counts per line
  - E.g.,
    - Line 1, word is house. House occurs twice
    - Line 2, word is not house
  - Generates the following output for the work house
    - house, 2, 1
    - house, 1, 1
- **Same general idea though. Tunnel vision on one line (even one word!)**
-

Map Reduce Algorithm Design

- 7.1 Background and Motivation (5)
- 7.2 MrJob
  - Installation (5)
  - MrJob Fundamentals and Concepts (5)
  - Writing Mrjob code (10)
  - BLT Word frequency challenge (15)
  - Log file processing (10)
  - BLT log file processing challenge (15)
  - Serializable, JSON and other MrJob info (10)
- END of Lecture
- Clustering Algorithms
  - Clustering overview (10 mins)
  - Kmeans algorithm (5)
  - Distributed Kmeans in MrJob [ScreenFlow] (10 minutes)
    - Initialization (Canopy Clustering)
- Sync time
  - MrJob : cluster tweets
  - Model-based clustering [Sync time or some in lecture 5]
    - Intro [5]

V4

Log Files

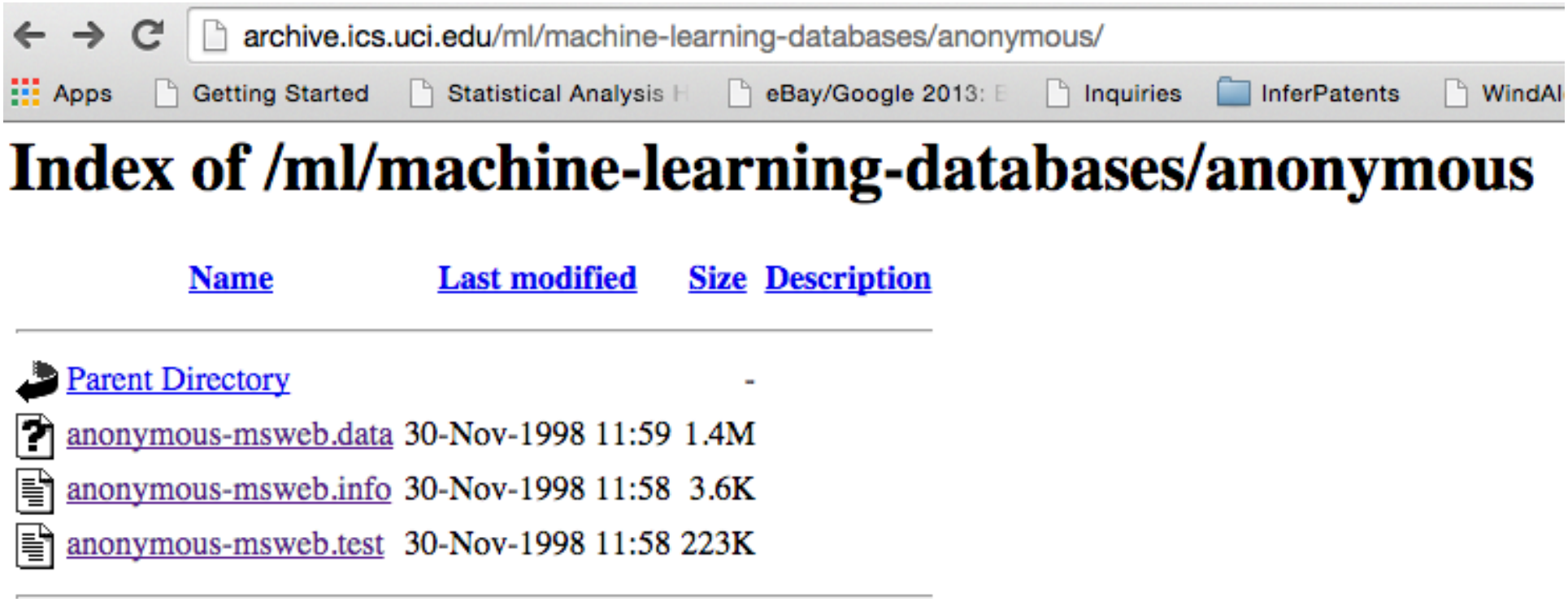
- **In computing, a logfile is a file that records events**
  - Human or system related event
  - that occur in an operating system or other software runs, or messages between different users of a communication software.
- **Logging is the act of keeping a log.**
- **In the simplest case, messages are written to a single logfile.**
- **Internet companies: mobile or webbased, track behavior, improve, and even machine learn over them**
- **M2M, human generated**

Microsoft customer visitor log file

- **Microsoft customer visitor log file data is located here**
  - Anonymous web data from www.microsoft.com
  - Contains information in CSV format
- **use mrjob MapReduce Framework to find answers**

Old fashioned log file

Microsoft Log Data



Log File Description

- **File tells us information about a "Vroot" (Web page)**
  - Vroot is just a fancy name for a page with a numeric ID
- **Each line is different information about the vroot. Each line begins with a single character with the following meanings:**
  - A: attributes of a vroot (id, title, url)
  - V: a visit to a vroot (id)
  - C: customer information, following which are the Visits that customer made to the vroot

38,000 visitors to Microsoft.com

-
```
A,1100,1,"MS in Education","/education"
A,1210,1,"SNA Support","/snasupport"
C,"10001",10001
V,1000,1
V,1001,1
V,1002,1
C,"10002",10002
V,1001,1
V,1003,1
C,"10003",10003
```

Meta data for Page 1100

Customer id 10001

Visit by customer 10001 to page id 1000

Visit by customer 10001 to page id 1001

Visit by customer 10001 to page id 1002

Customer id 10002

Microsoft Log Data

```
We created the data by sampling and processing the www.microsoft.com logs.
The data records the use of www.microsoft.com by 38000 anonymous,
randomly-selected users. For each user, the data lists all the areas of
the web site (Vroots) that user visited in a one week timeframe.

Users are identified only by a sequential number, for example, User #14988,
User #14989, etc. The file contains no personally identifiable information.
The 294 Vroots are identified by their title (e.g. "NetShow for PowerPoint")
and URL (e.g. "/stream"). The data comes from one week in February, 1998.

Dataset format:
    -- The data is in an ASCII-based sparse-data format called "DST".
       Each line of the data file starts with a letter which tells the line's type.
       The three line types of interest are:
           -- Attribute lines:
                 For example, 'A,1277,1,"NetShow for PowerPoint","/stream"'
                 Where:
                    'A' marks this as an attribute line,
                    '1277' is the attribute ID number for an area of the website
                              (called a Vroot),
                    '1' may be ignored,
                    '"NetShow for PowerPoint"' is the title of the Vroot,
                    '"/stream"' is the URL relative to "http://www.microsoft.com"
            -- Case and Vote Lines:
                For each user, there is a case line followed by zero or more vote lines.
                 For example:
                       C,"10164",10164
                       V,1123,1
                       V,1009,1
                       V,1052,1
                  Where:
                     'C' marks this as a case line,
                      '10164' is the case ID number of a user,
                     'V' marks the vote lines for this case,
                     '1123', 1009', 1052' are the attributes ID's of Vroots that a
                           user visited.
                      '1' may be ignored.
```

TODO

- All code is located in the following notebooks
- Please download notebooks and the Micrsosoft customer visitor logfile data

- TODO: insert location of notebooks and logfile

TODO:

Logfile Challenges + thought experiment

- **Logfile Coding Challenge 1: Pages>400 visits**
- **Logfile Thought Experiment 2:**
  - Challenge 2: Thought Experiment: User Visits
  - Challenge 2.1 Thought Experiment: User Visits
- **Logfile Coding Challenge 3: titles of the most visited pages [Left as homework]**

Logfile Challenge 1: Pages with > 400 visits

- Using the Microsoft customer visitor logfile data described above, find all pages (aka Vroots) with greater than 400 visits
- Start off with a code template and fill in the blanks code provided in the Notebook (in Section 4.2)
- Then submit your answer to the system by selecting one of the multiple choices provided.

Logfile Logfile Challenge 1: Pages with > 400 visits

- Using the (partially completed) code provided in the Notebook (in Section 4.2), find all pages (aka Vroots) with greater than 400 visits
- If possible, start off with a code template and fill in the blanks (and not the completed code)
- Using your completed code: how many pages with greater than 400 visits? Pick one of the following:

A ○ 0

B ○ Between 0 to 20

C ○ Between 20 to 35

D ○ More than 36

Logfile Challenge 1: Fill Code in Python Notebook

Problem 1: Find out the pages which have more than 400 visits

Complete code in MrJob class file

```
%load top_pages.py
```

```
"""Find Vroots with more than 400 visits.

This program will take a CSV data file and output tab-seperated lines of

    Vroot -> number of visits

To run:

    python top_pages.py anonymous-msweb.data

To store output:

    python top_pages.py anonymous-msweb.data > top_pages.out
"""

from mrjob.job import MRJob
import csv

def csv_readline(line):
    """Given a sting CSV line, return a list of strings."""
    for row in csv.reader([line]):
        return row

class TopPages(MRJob):

    def mapper(self, line_no, line):
        """Extracts the Vroot that was visited"""
        cell = csv_readline(line)
        if cell[0] == 'V':
            yield ### FILL IN
                  # What  Key, Value  do we want to output?

    def reducer(self, vroot, visit_counts):
        """Sumarizes the visit counts by adding them together.  If total visits
        is more than 400, yield the results"""
        total = ### FILL IN
                # How do we calculate the total visits from the visit_counts?
        if total > 400:
            yield ### FILL IN
                  # What  Key, Value  do we want to output?

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

```
A,1100,1,"MS in Education","/education"
A,1210,1,"SNA Support","/snasupport"
C,"10001",10001
V,1000,1
V,1001,1
V,1002,1
C,"10002",10002
V,1001,1
V,1003,1
```

Meta data for Page 1100

Customer id 10001

Visit by customer 10001 to page id 1000

Visit by customer 10001 to page id 1001

Visit by customer 10001 to page id 1002

Customer id 10002

Logfile Challenge 1: Fill Code in Python Notebook

Driver code

Challenge 1: Pages with > 400 visits

```
1 from top_pages import TopPages
2 import csv
3
4 mr_job = TopPages(args=['anonymous-msweb.data'])
5 with mr_job.make_runner() as runner:
6     runner.run()
7     for line in runner.stream_output():
8         print mr_job.parse_output_line(line)
```

Logfile Challenge 1: Fill Code in Python Notebook: All code

Challenge 1: Pages with > 400 visits

```
%%writefile top_pages.py
#Find Vroots with more than 400 visits.
from mrjob.job import MRJob
import csv

def csv_readline(line):
    """Given a sting CSV line, return a list of strings."""
    for row in csv.reader([line]):
        return row

class TopPages(MRJob):

    def mapper(self, line_no, line):
        """Extracts the Vroot that was visited"""
        cell = csv_readline(line)
        if cell[0] == 'V':
            yield ### FILL IN
                  # What  Key, Value  do we want to output?

    def reducer(self, vroot, visit_counts):
        """Sumarizes the visit counts by adding them together.  If total visits
        is more than 400, yield the results"""
        total = ### FILL IN
                # How do we calculate the total visits from the visit counts?
        if total > 400:
            yield ### FILL IN
                  # What  Key, Value  do we want to output?

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

```
Overwriting top_pages.py
```

Driver code is ready

```
from top_pages import TopPages
import csv

mr_job = TopPages(args=['anonymous-msweb.data'])
with mr_job.make_runner() as runner:
    runner.run()
    for line in runner.stream_output():
        print mr_job.parse_output_line(line)
```

Run driver to get results

Driver code is ready

Logfile Challenge 1: Pages with > 400 visits

```
from top_pages_solution import TopPages
import csv

mr_job = TopPages(args=['anonymous-msweb.data'])
with mr_job.make_runner() as runner:
    runner.run()
    for line in runner.stream_output():
        print mr_job.parse_output_line(line)
```

```
(u'1000', 912)
(u'1001', 4451)
(u'1002', 749)
(u'1003', 2968)
(u'1004', 8463)
(u'1007', 865)
(u'1008', 10836)
(u'1009', 4628)
(u'1010', 698)
(u'1014', 728)
(u'1017', 5108)
(u'1018', 5330)
(u'1020', 1087)
(u'1024', 521)
(u'1025', 2123)
(u'1026', 3220)
(u'1027', 507)
(u'1030', 1115)
(u'1031', 574)
(u'1032', 1446)
(u'1034', 9383)
(u'1035', 1791)
(u'1036', 759)
(u'1037', 1160)
(u'1038', 1110)
(u'1040', 1506)
(u'1041', 1500)
(u'1045', 474)
(u'1046', 636)
(u'1052', 842)
(u'1053', 670)
(u'1058', 672)
(u'1067', 548)
(u'1070', 602)
(u'1074', 584)
(u'1076', 444)
(u'1078', 462)
(u'1295', 716)
```

- **solution**

Solutions

Logfile Challenge 1: Pages with > 400 visits

```
from mrjob.job import MRJob
import csv

def csv_readline(line):
    """Given a sting CSV line, return a list of strings."""
    for row in csv.reader([line]):
        return row

class TopPages(MRJob):

    def mapper(self, line_no, line):
        """Extracts the Vroot that was visited"""
        cell = csv_readline(line)
        if cell[0] == 'V':
            yield cell[1],1

    def reducer(self, vroot, visit_counts):
        """Sumarizes the visit counts by adding them together.  If total visits
        is more than 400, yield the results"""
        total = sum(i for i in visit_counts)
        if total > 400:
            yield vroot, total

if __name__ == '__main__':
    TopPages.run()

```

Logfile Challenge 1: Pages with > 400 visits

- Using the (partially completed) code provided in the Notebook located here (and presented in Section 4.7), find all pages (aka Vroots) with greater than 400 visits
- If possible, start off with a code template and fill in the blanks (please do NOT use the completed notebook.)
- Using your completed code: how many pages with greater than 400 visits? Pick one of the following:

A ○ 0
B ○ Between 0 to 20
C ○ Between 20 to 35
D ○ More than 36

- **End of section**

Map Reduce Algorithm Design

- 7.1 Background and Motivation (5)
- 7.2 MrJob
  - Installation (5)
  - MrJob Fundamentals and Concepts (5)
  - Writing Mrjob code (10)
  - BLT Word frequency challenge (15)
  - Log file processing (10)
  - BLT log file processing challenge (15)
  - Serializable, JSON and other MrJob info (10)
- END of Lecture
- Clustering Algorithms
  - Clustering overview (10 mins)
  - Kmeans algorithm (5)
  - Distributed Kmeans in MrJob [ScreenFlow] (10 minutes)
    - Initialization (Canopy Clustering)
- Sync time
  - MrJob : cluster tweets
  - Model-based clustering [Sync time or some in lecture 5]
    - Intro [5]

V4

- **Challenge 15 minutes**
  - Type code in
  - Run code and report result

Logfile Challenge 1: Fill Code in Python Notebook: Solutions

Challenge 1: Pages with > 400 visits

```
16 from mrjob.job import MRJob
17 import csv
18
19 def csv_readline(line):
20     """Given a sting CSV line, return a list of strings."""
21     for row in csv.reader([line]):
22         return row
23
24 class TopPages(MRJob):
25
26     def mapper(self, line_no, line):
27         """Extracts the Vroot that was visited"""
28         cell = csv_readline(line)
29         if cell[0] == 'V':
30             yield cell[1],1
31
32     def reducer(self, vroot, visit_counts):
33         """Sumarizes the visit counts by adding them together.  If total visits
34         is more than 400, yield the results"""
35         total = sum(i for i in visit_counts)
36         if total > 400:
37             yield vroot, total
38
39 if __name__ == '__main__':
40     TopPages.run()
41
```

Logfile Challenge 1: Fill Code in Python Notebook:

Run driver to get results

All code

Driver code is ready

Challenge 1: Pages with > 400 visits

```
from top_pages_solution import TopPages
import csv

mr_job = TopPages(args=['anonymous-msweb.data'])
with mr_job.make_runner() as runner:
    runner.run()
    for line in runner.stream_output():
        print mr_job.parse_output_line(line)
```

```
(u'1000', 912)
(u'1001', 4451)
(u'1002', 749)
(u'1003', 2968)
(u'1004', 8463)
(u'1007', 865)
(u'1008', 10836)
(u'1009', 4628)
(u'1010', 698)
(u'1014', 728)
(u'1017', 5108)
(u'1018', 5330)
(u'1020', 1087)
(u'1024', 521)
(u'1025', 2123)
(u'1026', 3220)
(u'1027', 507)
(u'1030', 1115)
(u'1031', 574)
(u'1032', 1446)
(u'1034', 9383)
(u'1035', 1791)
(u'1036', 759)
(u'1037', 1160)
(u'1038', 1110)
(u'1040', 1506)
(u'1041', 1500)
(u'1045', 474)
(u'1046', 636)
(u'1052', 842)
(u'1053', 670)
(u'1058', 672)
(u'1067', 548)
(u'1070', 602)
(u'1074', 584)
(u'1076', 444)
(u'1078', 462)
(u'1295', 716)
```

Logfile Challenge 1: The multiple choice solution

- How many pages with greater than 400 visits?

  - ○ 0
  - ○ Between 0 to 20
  - ● Between 20 to 50
  - ○ More than 50

Map Reduce Algorithm Design

- 7.1 Background and Motivation (5)
- 7.2 MrJob
  - Installation (5)
  - MrJob Fundamentals and Concepts (5)
  - Writing Mrjob code (10)
  - BLT Word frequency challenge (15)
  - Log file processing (10)
  - BLT log file processing challenge (15)
  - Serializable, JSON and other MrJob info (10)
- END of Lecture
- Clustering Algorithms
  - Clustering overview (10 mins)
  - Kmeans algorithm (5)
  - Distributed Kmeans in MrJob [ScreenFlow] (10 minutes)
    - Initialization (Canopy Clustering)
- Sync time
  - MrJob : cluster tweets
  - Model-based clustering [Sync time or some in lecture 5]
    - Intro [5]

V4

Section 4.8

- Oyster Thought Experiment
- Bad Logs

- Good logs, bad logs, old logs,

- Do EDA,

- Do machine learning

Homework challenge

Logile Challenge 2: Thought Experiment: User Visits

```
Line
1 A,1100,1,"MS in Education","/education"   Meta data for Page 1100
2 A,1210,1,"SNA Support","/snasupport"
3 C,"10001",10001                          Customer id 10001
4 V,1000,1                                 Visit by customer 10001 to page id 1000
5 V,1001,1                                 Visit by customer 10001 to page id 1001
6 V,1002,1                                 Visit by customer 10001 to page id 1002
7 C,"10002",10002                          Customer id 10002
8 V,1001,1
9 V,1003,1
  C,"10003",10003
  V,1001,1
```

- Can we calculate the number of visits per user using Map Reduce?
- Pick an answer and justify in 100 words or less why. Please use the line numbers to explain.
- A: No
- B: Yes

Logfile Challenge 2: User Visits: Solution

```
Line
1 A,1100,1,"MS in Education","/education"   Meta data for Page 1100
2 A,1210,1,"SNA Support","/snasupport"
3 C,"10001",10001                          Customer id 10001
4 V,1000,1                                 Visit by customer 10001 to page id 1000
5 V,1001,1                                 Visit by customer 10001 to page id 1001
6 V,1002,1                                 Visit by customer 10001 to page id 1002
7 C,"10002",10002                          Customer id 10002
8 V,1001,1
9 V,1003,1
  C,"10003",10003
  V,1001,1
```

- **Can we calculate the number of visits per user using Map Reduce?**
- **No!**
  - User visit information is spread over multiple lines. As such there is a sequential dependency different between each line. E.g., Line 3 refers to customer 10001 and the following three lines refer to visits by this customer.
  - We cannot assume linear/sequential processing of the data by the Hadroop framework. It splits the input file at arbitrary locations. Meaning, e.g., lines 1 to 4 could go to one mapper and line 5 to 9 could go to another mapper. In this case it would be impossible to recover that lines 5 and 6 we associated with customer 10001
- **So what can we do?**

- **Challenge 2.1:**
- **Oyster Thought Experiment**

Challenge 2.1 : User Visit

```
Line
1 A,1100,1,"MS in Education","/education"   Meta data for Page 1100
2 A,1210,1,"SNA Support","/snasupport"
3 C,"10001",10001                          Customer id 10001
4 V,1000,1                                 Visit by customer 10001 to page id 1000
5 V,1001,1                                 Visit by customer 10001 to page id 1001
6 V,1002,1                                 Visit by customer 10001 to page id 1002
7 C,"10002",10002                          Customer id 10002
8 V,1001,1
9 V,1003,1
  C,"10003",10003
  V,1001,1
```

- So what preprocessing step do we need to take so that we can calculate the number of visits per user using Map Reduce?

Challenge 2.1 : User Visit: Solution

```
Line
1 A,1100,1,"MS in Education","/education"   Meta data for Page 1100
2 A,1210,1,"SNA Support","/snasupport"
3 C,"10001",10001                          Customer id 10001
4 V,1000,1                                 Visit by customer 10001 to page id 1000
5 V,1001,1                                 Visit by customer 10001 to page id 1001
6 V,1002,1                                 Visit by customer 10001 to page id 1002
7 C,"10002",10002                          Customer id 10002
8 V,1001,1
9 V,1003,1
  C,"10003",10003
  V,1001,1
```

- **So what preprocessing step do we need to take so that we can calculate the number of visits per user using Map Reduce?**
- **Transform Data on a single machine**
  - MapReduce needs all information for a customer and for a visit on one line
  - Add user id to a visit record
  - Write single threaded program (not mrjob) to transform it

Homework: preprocess the data

- Homework
- Write the single program to put the user id into each Visitor record
- Count up the number for visit per user
- Who is the most frequent visitor

Homework challenge

- **End of section**

L4: Map Reduce Algorithm Design

- 4.1 Background and Motivation (5)
- 4.2 MrJob
  - Installation (5)
  - BLT Install MrJob and verify [5]
  - MrJob Fundamentals and Concepts (5)
  - Writing Mrjob code (10)
  - BLT Join operation (15)
  - Log file processing (10)
  - BLT: most frequently visited pages [15]
  - Oyster: Thought experiment: bad logs [5]
  - BLT Challenge: Most frequently visited titles (15)
  - Serializable, JSON and MrJob benchmarks (10) [75]
- 4.4 Clustering Algorithms [35]
  - Clustering overview (10 mins)
  - Kmeans algorithm (5)
  - Distributed Kmeans in MrJob [ScreenFlow] (15 minutes)
    - Initialization (Canopy Clustering)

V5

Logfile Challenge: Most frequently visited titles

- **Using the provided iPython notebook (see assigned readings and code section) as a starting point, find the titles of the most visited pages in the microsoft logs. What is the ranking for the following page titles?**
  - 1. “isapi”
  - 2. “Free Downloads”
  - 3. "Microsoft.com Search"
  - 4. “Internet Explorer”

Homework challenge

Options:

- **A. 1, 2, 3, 4**
- **B. 2, 4, 3, 1**
- **C. 3, 4, 1, 2**
- **D. 4, 2, 1, 3**

Top titles: Fill Code in Python Notebook

Problem 1: output title name of vroot and number of visit for that vroot

Complete code in MrJob class file

```
%load count_titles.py
```

```
"""Find the titles of the top visited Vroots.

This program will take a CSV data file and output lines of

    Vroot Title

To run:

    python count_titles.py anonymous-msweb.data
"""

from mrjob.job import MRJob
import csv

def csv_readline(line):
    """Given a sting CSV line, return a list of strings."""
    for row in csv.reader([line]):
        return row

class CountTitles(MRJob):

    def mapper(self, line_no, line):
        """Extracts the Vroot that was visited"""
        cell = csv_readline(line)
        if cell[0] == 'V':
            yield ### FILL IN
                  # How to "tag" this value for a given Key
        elif cell[0] == 'A':
            yield ### FILL IN
                  # How to "tag" this value for a given Key

    def reducer(self, vroot, visit_counts_and_title):
        """Sumarizes the visit counts by adding them together.  Extracts title,
        sums visit counts, returns both."""
        total = 0
        title = ''

        for value in visit_counts_and_title:
            if # FILL IN: value is a visit type:
                total += # FILL IN: extract untagged value
            elif # FILL IN: value is a attribute type:
                title = # FILL IN: extract untagged title

        yield title, total

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

Top titles: Driver code is ready

```
1  from count_titles import CountTitles
2  import csv
3
4  mr_job = CountTitles(args=['anonymous-msweb.data'])
5  with mr_job.make_runner() as runner:
6      runner.run()
7      title_counts = [mr_job.parse_output_line(line) for line in
8              runner.stream_output()]
9      results = sorted(title_counts, key=lambda (k,v): v, reverse=True)
10     print results[:10]
```

A common pattern is to sort complex objects using some of the object's indices as a key. For example:

```
>>> student_tuples = [
        ('john', 'A', 15),
        ('jane', 'B', 12),
        ('dave', 'B', 10),
]
>>> sorted(student_tuples, key=lambda student: student[2])   # sort by age
[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]
```

- **Insert solution**

Top titles: Solutions

```
20 class CountTitles(MRJob):
21
22     def mapper(self, line_no, line):
23         """Extracts the Vroot that was visited"""
24         cell = csv_readline(line)
25         if cell[0] == 'V':
26             yield cell[1],1
27         elif cell[0] == 'A':
28             yield cell[1],cell[3]
29
30     def reducer(self, vroot, visit_counts_and_title):
31         """Sumarizes the visit counts by adding them together.  Extracts title,
32         sums visit counts, returns both."""
33         total = 0
34         title = ''
35
36         for value in visit_counts_and_title:
37             if isinstance(value, int):
38                 total += value
39             elif isinstance(value, basestring):
40                 title = value
41
42         yield title, total
43
44 if __name__ == '__main__':
45     CountTitles.run()
46
```

Top titles:Driver: Run driver to get results

Driver code is ready

```
from count_titles_solution import CountTitles
import csv

mr_job = CountTitles(args=['anonymous-msweb.data'])
with mr_job.make_runner() as runner:
    runner.run()
    title_counts = [mr_job.parse_output_line(line) for line in
            runner.stream_output()]
    results = sorted(title_counts, key=lambda (k,v): v, reverse=True)
    print results[:10]
```

```
[(u'Free Downloads', 10836), (u'Internet Explorer', 9383), (u'Microsoft.com Search', 8463), (u'isapi', 5330), (u'Products ', 5108), (u'Win
dows Family of OSs', 4628), (u'Support Desktop', 4451), (u'Internet Site Construction for Developers', 3220), (u'Knowledge Base', 2968),
(u"Web Site Builder's Gallery", 2123)]
```

Challenge 3: Most frequently visited titles: Solution

- **What's the ranking for the following pages?**
  - 1. "isapi"
  - 2. "Free Downloads"
  - 3. "Microsoft.com Search"
  - 4. "Internet Explorer"

Options:

- ○ **A. 1, 2, 3, 4**
- ● **B. 2, 4, 3, 1**
- ○ **C. 3, 4, 1, 2**
- ○ **D. 4, 2, 1, 3**

- **End BLT**

L4: Map Reduce Algorithm Design

- 4.1 Background and Motivation (5)
- 4.2 MrJob
  - Installation (5)
  - BLT Install MrJob and verify [5]
  - MrJob Fundamentals and Concepts (5)
  - Writing Mrjob code (10)
  - BLT Join operation (15)
  - Log file processing (10)
  - BLT: most frequently visited pages [15]
  - Oyster: Thought experiment: bad logs [5]
  - BLT Challenge: Most frequently visited titles (15)
  - Serializable, JSON and MrJob benchmarks (10) [75]
- 4.4 Clustering Algorithms [35]
  - Clustering overview (10 mins)
  - Kmeans algorithm (5)
  - Distributed Kmeans in MrJob [ScreenFlow] (15 minutes)
    - Initialization (Canopy Clustering)

V5

- **JSON (JavaScript Object Notation)**

- **Logfiles are core to any modern day company:**
- **How we store event records is critical**
- **Until now we have focused on text based inputs and outputs**
- **In this section we look at more informative logfiles via a standard**

JSON (JavaScript Object Notation)

- JSON (JavaScript Object Notation): an open standard format that uses human-readable text to transmit data objects
- Consists of attribute–value pairs
- It is used primarily to transmit data between a server and web application, as an alternative to XML.

The following example shows a possible JSON representation describing a person.

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "height_cm": 167.6,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

Values can be

- String
- Integer
- Real
- Boolean
- Hierarchical JSON

JSON Schema specifies the grammar of legal JSON records

- JSON Schema specifies a JSON-based format to define the structure of JSON data for validation, documentation, and interaction control.

- A JSON Schema provides a contract for the JSON data required by a given application, and how that data can be modified.

Example JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-03/schema#",
  "name": "Product",
  "type": "object",
  "properties": {
    "id": {
      "type": "number",
      "description": "Product identifier",
      "required": true
    },
    "name": {
      "type": "string",
      "description": "Name of the product",
      "required": true
    },
    "price": {
      "type": "number",
      "minimum": 0,
      "required": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string"
      }
    },
    "stock": {
      "type": "object",
      "properties": {
        "warehouse": {
          "type": "number"
        },
        "retail": {
          "type": "number"
        }
      }
    }
  }
}
```

Product: list of properties some of which are optional

Number Reqd

Number Min

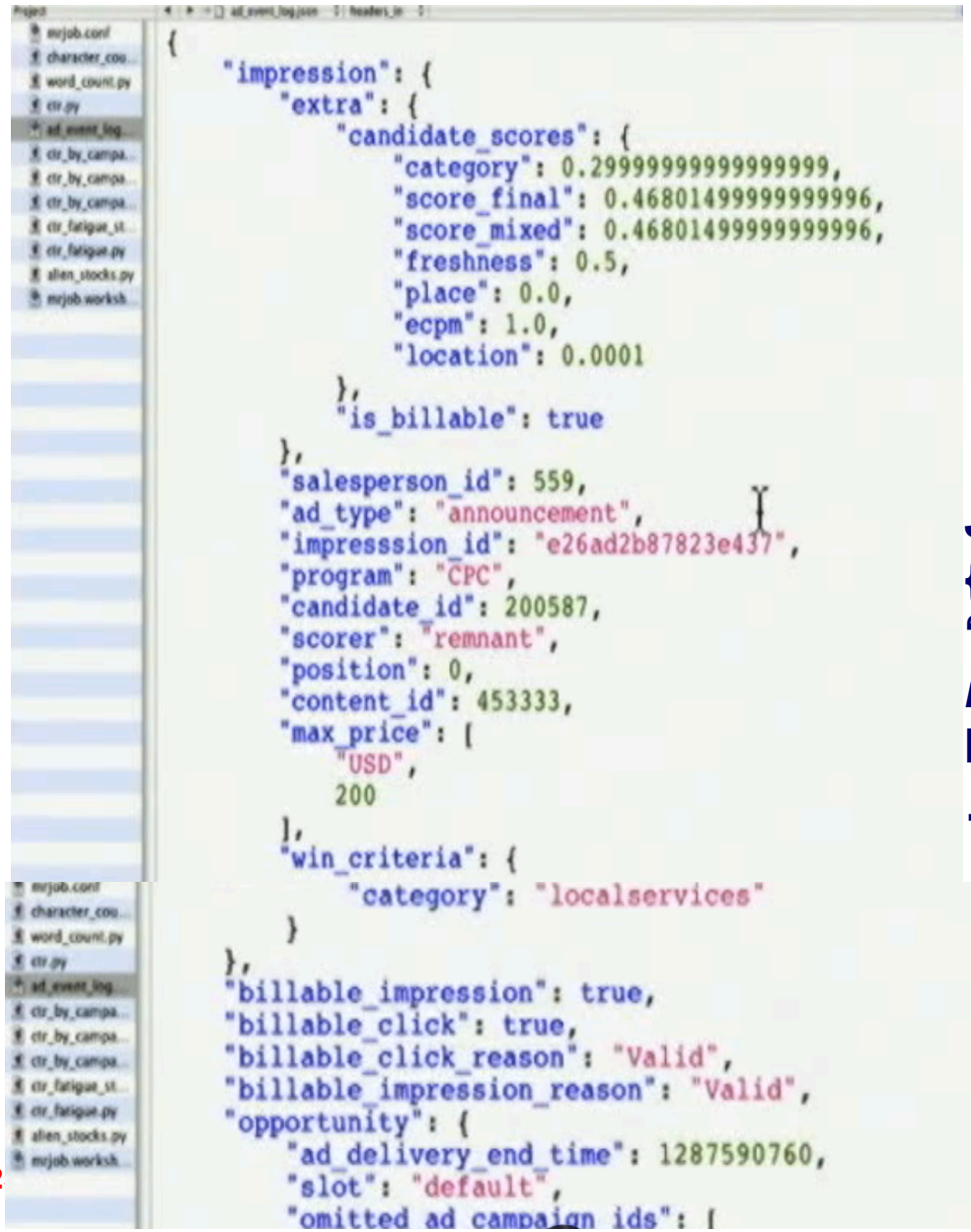
Array

Object

This JSON Schema can be used to test the validity of the JSON record below:

```
{
  "id": 1,
  "name": "Foo",
  "price": 123,
  "tags": [
    "Bar",
    "Eek"
  ],
  "stock": {
    "warehouse": 300,
    "retail": 20
  }
}
```

Hierarchical JSON object



```
{
    "impression": {
        "extra": {
            "candidate_scores": {
                "category": 0.29999999999999999,
                "score_final": 0.46801499999999996,
                "score_mixed": 0.46801499999999996,
                "freshness": 0.5,
                "place": 0.0,
                "ecpm": 1.0,
                "location": 0.0001
            },
            "is_billable": true
        },
        "salesperson_id": 559,
        "ad_type": "announcement",
        "impresssion_id": "e26ad2b87823e437",
        "program": "CPC",
        "candidate_id": 200587,
        "scorer": "remnant",
        "position": 0,
        "content_id": 453333,
        "max_price": [
            "USD",
            200
        ],
        "win_criteria": {
            "category": "localservices"
        }
    },
    "billable_impression": true,
    "billable_click": true,
    "billable_click_reason": "Valid",
    "billable_impression_reason": "Valid",
    "opportunity": {
        "ad_delivery_end_time": 1287590760,
        "slot": "default",
        "omitted_ad_campaign_ids": [
```

JSON OBJECT
{….
“billable_impression”: true ***#drop robot clicks***
billable_click”: true
….

Accessing a component of a component

{“impression:” {
“extra:”{…
}
“Candidate_id:” 200587
….}
In python access object attributes as follows:
ad_campaign = ad_event[‘impression’][‘candidate_id’]

Input/output format from STDIN/STDOUT

mrjob: serialization

Defaults

```
class MyMRJob(mrjob.job.MRJob):
    INPUT_PROTOCOL = mrjob.protocol.RawValueProtocol
    INTERNAL_PROTOCOL = mrjob.protocol.JSONProtocol
    OUTPUT_PROTOCOL = mrjob.protocol.JSONProtocol
```

Available

RawProtocol / RawValueProtocol
JSONProtocol / JSONValueProtocol
PickleProtocol / PickleValueProtocol
ReprProtocol / ReprValueProtocol

Custom protocols can be written.
No current support for binary serialization schemes.

MrJob: INPUT PROTOCOL

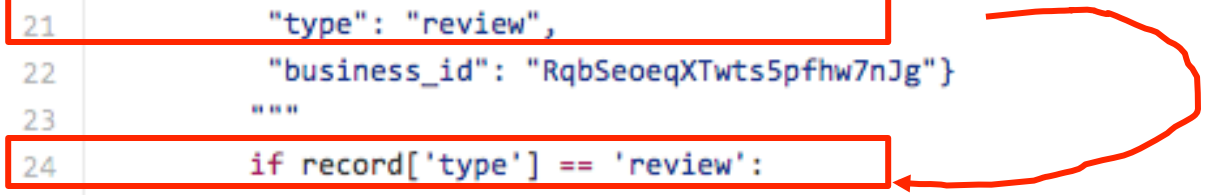
- **Previous examples manually parsed line**
- **mrjob provides "protocols" to automatically parse and recover from error**
- **Specify the class desired to parse the lines**
- **Examples**
  - Open up code/unique_review.py
  - We're using a JSON object per line. Other options include JSON key-values,
  - or writing your own
  - Instead of raw text, *record* will now be a Python dict
-

MrJob: Template Job with input and output serialization formats

mrjob

```
class NgramNeighbors(MRJob):
    # specify input/intermed/output serialization
    # default output protocol is JSON; here we set it to text
    OUTPUT_PROTOCOL = RawProtocol

    def mapper(self, key, line):
        pass

    def combiner(self, key, counts):
        pass

    def reducer(self, key, counts):
        pass

if __name__ == '__main__':
    # sets up a runner, based on command line options
    NgramNeighbors.run()
```

```
1  from mrjob.job import MRJob
2  from mrjob.protocol import JSONValueProtocol
3
4  import re
5
6  # Use this regular expresion to break up text via findall()
7  WORD_RE = re.compile(r"[\w']+")
8
9  class UniqueReview(MRJob):
10     INPUT_PROTOCOL = JSONValueProtocol
11
12     def extract_words(self, _, record):
13         """Take in a record, filter by type=review, yield <word, review_id>
14         records look like:
15         {"votes": {"funny": 3, "useful": 3, "cool": 2},
16          "user_id": "dYtkphUrU7S2_bjif6k2uA",
17          "review_id": "gjtWdiEMMfoOTCfdd3hPmA",
18          "stars": 4,
19          "date": "2009-04-24",
20          "text": "Man, if these guys were trying to replicate a gringo bar in Mexico...",
21          "type": "review",
22          "business_id": "RqbSeoeqXTwts5pfhw7nJg"}
23         """
24         if record['type'] == 'review':
25             ###
26             # TODO: for each word in the review, yield the correct key,value
27             # pair:
28             # for word in ____:
29             #   yield [ ___ , ___ ]
30             ##/
31
```



Fundamentals and Concept

- **Protocols: each job has an input protocol, an output protocol, and an internal protocol**
  - Input protocol: read the bytes sent to the first mapper (or reducer, if your first step doesn't use a mapper)
  - Internal protocol: converts the output of one step to the input of the next if the job has more than one step
  - Output protocol: write the output of the last step to bytes written to the output file
  - Defaut is

    INPUT_PROTOCOL = mrjob.protocol.RawValueProtocol

    INTERNAL_PROTOCOL = mrjob.protocol.JSONProtocol

    OUTPUT_PROTOCOL = mrjob.protocol.JSONProtocol

  But it can be changed.

Serialization

- **Serialization is the process of turning structured objects into a byte stream for transmission over a network or for writing to persistent storage.**
- **Deserialization is the reverse process of turning a byte stream back into a series of structured objects.**
- **Serialization appears in two quite distinct areas of distributed data processing:**
  - for interprocess communication
  - for persistent storage

Sort word freq

In simple word count map reduce program the output we get is sorted by words. Sample output can be :
Apple 1
Boy 30
Cat 2
Frog 20
Zebra 1
If you want output to be sorted on the basis of number of occrance of words, i.e in below format
1 Apple
1 Zebra
2 Cat
20 Frog
30 Boy
You can create another MR program using below mapper and reducer where the input will be the output got from simple word count program.

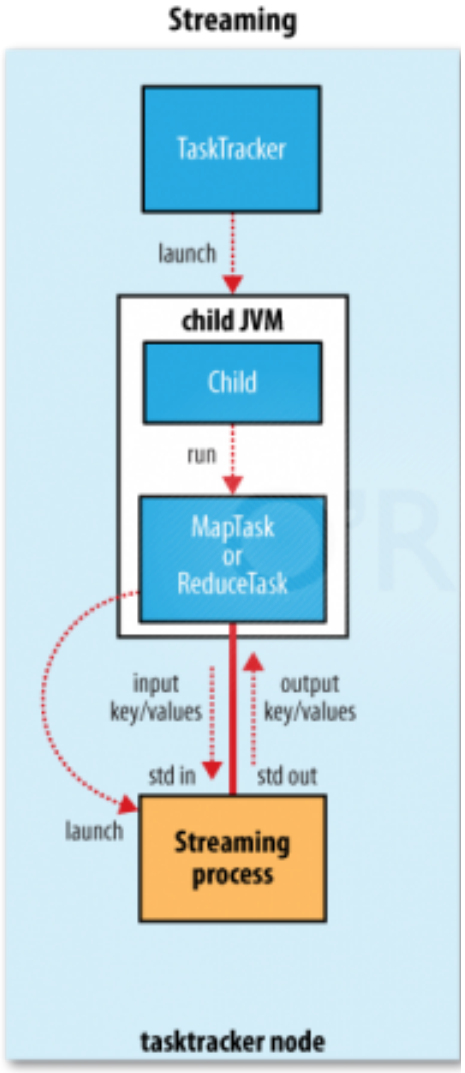
```
class Map1 extends MapReduceBase implements Mapper<Object, Text, IntWritable, Text>
{
    public void map(Object key, Text value, OutputCollector<IntWritable, Text> collector, Re
    {
        String line = value.toString();
        StringTokenizer stringTokenizer = new StringTokenizer(line);
        {
            int number = 999;
            String word = "empty";

            if(stringTokenizer.hasMoreTokens())
            {
                String str0= stringTokenizer.nextToken();
                word = str0.trim();
            }

            if(stringTokenizer.hasMoreElements())
            {
                String str1 = stringTokenizer.nextToken();
                number = Integer.parseInt(str1.trim());
            }

            collector.collect(new IntWritable(number), new Text(word));
        }
```

http://stackoverflow.com/questions/2550784/sorted-word-count-using-hadoop-mapreduce

```
class Reduce1 extends MapReduceBase implements Reducer<IntWritable, Text, IntWritable, Text>
{
    public void reduce(IntWritable key, Iterator<Text> values, OutputCollector<IntWritable,
    {
        while((values.hasNext()))
        {
            arg2.collect(key, values.next());
        }
```

MrJob accepts as input text formats: Raw text and JSON

- **MrJob accepts as input text formats: Raw text and JSON**
- **Binary formats first need to be converted.**
- **Text is processing is slow, incurs extra storage and network costs (serialization and deserialization for mappers/reducers in Hadoop Streaming).**
- **E.g., Avro**
  - Since Avro is a binary format (by default), you won't be able to use it directly with Hadoop Streaming.
  - You could provide a JVM input/output format which converted the Avro data to lines (eg JSON), then use MRJob to process those lines normally.
  - You'd want to do this primarily for compatibility, though, since you'll have an overhead per line.



L4: Map Reduce Algorithm Design

- 4.1 Background and Motivation (5)
- 4.2 MrJob
  - Installation (5)
  - BLT Install MrJob and verify [5]
  - MrJob Fundamentals and Concepts (5)
  - Writing Mrjob code (10)
  - BLT Join operation (15)
  - Log file processing (10)
  - BLT: most frequently visited pages [15]
  - Oyster: Thought experiment: bad logs [5]
  - BLT Challenge: Most frequently visited titles (15)
  - Serializable, JSON and MrJob benchmarks (15) [75]
  - MrJob benchmark study(5) (60)
- 4.4 Clustering Algorithms [35]
  - Clustering overview (10 mins)
  - Kmeans algorithm (5)
  - Distributed Kmeans in MrJob [ScreenFlow] (15 minutes)

V5

- Hadoop comes with a set of primitives for data I/O; it can process text and binary formats.
- Text based I/O of object versus binary serialization?
- How good is MrJob, say, versus Hadoop streaming or other frameworks?
- This is a question that folks at Cloudera asked a couple of years ago..

- Hadoop comes with a set of primitives for data I/O.
- Some of these are techniques that are more general than Hadoop, such as data integrity and compression, but deserve special consideration when dealing with multiterabyte datasets. Others are Hadoop tools or APIs that form the building blocks for developing distributed systems, such as serialization frameworks and on-disk data structures.

Evaluate Python frameworks

Benchmark study

- **Evaluate Python frameworks that exist for working with Hadoop on a Benchmark dataset including:**
  - Hadoop Streaming
  - mrjob
  - dumbo
  - hadoopy
  - pydoop
  - and others

http://blog.cloudera.com/blog/2013/01/a-guide-to-python-frameworks-for-hadoop/

Hadoop Streaming comparison

- **We would like to aggregate the data to count the number of times any pair of words are observed near each other, grouped by year. This would allow us to determine if any pair of words are statistically near each other more often than we would expect by chance.**
- **Two words are "near" if they are observed within 4 words of each other.**

To test out the different frameworks, we will **not** be doing "word count". Instead, we will be transforming the **Google Books Ngram data**. An **n-gram** is a synonym for a tuple of *n* words. The n-gram data set provides counts for every single 1-, 2-, 3-, 4-, and 5-gram observed in the Google Books corpus grouped by year. Each row in the n-gram data set is composed of 3 fields: the n-gram, the year, and the number of observations. (You can explore the data interactively **here**.)

We would like to aggregate the data to count the number of times any pair of words are observed *near* each other, grouped by year. This would allow us to determine if any pair of words are statistically near each other more often than we would expect by chance. Two words are "near" if they are observed within 4 words of each other. Or equivalently, two words are near each other if they appear together in any 2-, 3-, 4-, or 5-gram. So a row in the resulting data set would be comprised of a 2-gram, a year, and a count.

There is one subtlety that must be addressed. The n-gram data set for each value of n is computed across the whole Google Books corpus. In principle, given the 5-gram data set, I could compute the 4-, 3-, and 2-gram data sets simply by aggregating over the correct n-grams. For example, if the 5-gram data set contains

```
(the, cat, in, the, hat)       1999     20
(the, cat, is, on, youtube)    1999     13
(how, are, you, doing, today)  1986   5000
```

Word coocurrence problem using ngram data

then we could aggregate this into 2-grams which would result in records like

```
(the, cat)  1999    33      // i.e., 20 + 13
```

a word is near a word if it occurs in 2-gram, 3-gram, 4-gram, 5-gram

The n grams in this dataset were produced by passing a sliding window of the text of books and outputting a record for each new token. For example, the following sentence.

The yellow dog played fetch.

Would produce the following 2-grams:

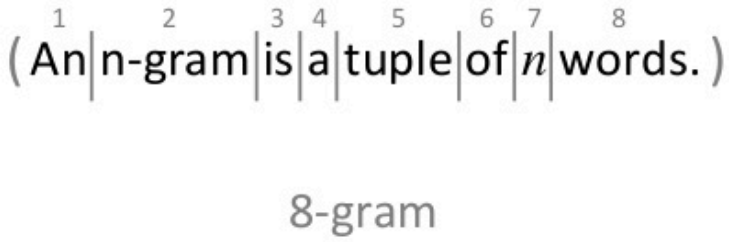
["The", "yellow"]
["yellow", 'dog"]
["dog", "played"]
["played", "fetch"]
["fetch", "."]

Or the following 3-grams:

["The", "yellow", "dog"]
["yellow", "dog", "played"]
["dog", "played", "fetch"]
["played", "fetch", "."]

Problem: aggregating the Google n-gram data

http://books.google.com/ngrams



Find pairs of words within 4 words of each other
I.e., a word is near a word if it occurs in 2-gram, 3-gram, 4-gram, 5-gram
E.g., *An* can be paired with *n-gram, is, a, tuple*

Google books Ngram Viewer

http://storage.googleapis.com/books/ngrams/books/datasetsv2.html

The Google Books Ngram Viewer is optimized for quick inquiries into the usage of small sets of phrases. If you're interested in performing a large scale analysis on the underlying data, you might prefer to download a portion of the corpora yourself. Or all of it, if you have the bandwidth and space. We're happy to oblige.

These datasets were generated in July 2012 (Version 2) and July 2009 (Version 1); we will update these datasets as our book scanning continues, and the updated versions will have distinct and persistent version identifiers (20120701 and 20090715 for the current sets).

Each of the numbered links below will directly download a fragment of the corpus. In Version 2 the ngrams are grouped alphabetically (languages with non-Latin scripts were transliterated); in Version 1 the ngrams are partitioned into files of equal size. In addition, for each corpus we provide a file named `total_counts`, which records the total number of 1-grams contained in the books that make up the corpus. This file is useful for computing the relative frequencies of ngrams.

A summary of how the corpora were constructed can be found here. We explain it in greater depth here (Version 2) and here (Version 1). In both, we point out that we've included only ngrams that appear over 40 times across the corpus. That's why the sum of the 1-gram occurrences in any given corpus is smaller than the number given in the `total_counts` file.

File format: Each of the files below is compressed *tab*-separated data. In Version 2 each line has the following format:

```
ngram TAB year TAB match_count TAB volume_count NEWLINE
```

As an example, here are the 3,000,000th and 3,000,001st lines from the `a` file of the English 1-grams (googlebooks-eng-all-1gram-20120701-a.gz):

```
circumvallate   1978   335    91
circumvallate   1979   261    91
```

The first line tells us that in 1978, the word "circumvallate" (which means "surround with a rampart or other fortification", in case you were wondering) occurred 335 times overall, in 91 distinct books of our sample.

The files vary widely in size because some patterns of letters are more common than others: the "na" file will be larger than the "ng" file since so many more words begin with "na" than "ng". Files with a letter followed by an underscore (e.g., `s_`) contain ngrams that begin with the first letter, but have an unusual second character.

We've included separate files for ngrams that start with punctuation or with other non-alphanumeric characters. Finally, we have separate files for ngrams in which the first word is a part of speech tag (e.g., `_ADJ_`, `_ADP_`).

Word pairs near each other (window = 4 words)

Streaming has the lowest overhead

	Java	Streaming*	mrjob*	dumbo*	hadoopy*
FILE: bytes read	22,726,677,381	0.94	1.34	2.55	1.97
FILE: bytes written	33,468,535,411	0.93	1.35	2.57	1.99
HDFS: bytes read	21,934,848,598	1.00	1.00	1.00	1.00
HDFS: bytes written	7,629,045,090	1.00	0.99	1.00	1.06
Map output bytes	12,978,686,993	0.91	1.40	2.11	2.11
Reduce shuffle bytes	11,336,515,993	0.92	1.35	2.53	1.97
Reduce input records	428,755,439	1.04	1.00	1.46	1.04
Time spent all maps (ms)	14,256,288	1.37	5.98	2.39	3.76
Time spent in all reduces (ms)	4,348,716	1.76	8.91	6.14	4.86
CPU time (ms)	14,016,540	1.17	4.68	3.68	2.76
Job run time (s)	1,074	1.54	7.31	3.90	4.20

*Ratios are relative to Java values

MrJob accepts as input/output text formats:
Raw text and JSON
Deserialization and serialization of records incurs a lot of CPU/Storage/Network overhead

http://blog.cloudera.com/blog/2013/01/a-guide-to-python-frameworks-for-hadoop/

MrJob is very attractive but a little slow

	Java	Streaming	mrjob	dumbo	hadoopy	pydoop
Underlying framework	Hadoop	Hadoop Streaming	Hadoop Streaming	Hadoop Streaming	Hadoop Streaming	Hadoop Pipes
Ease of installing framework	Easy with CDH (Whirr for cloud)	Easy with CDH (Whirr for cloud)	pip on client only	Build as egg, manually distribute to cluster	pip on client only (performance penalty) or manually install across cluster	Failed to build
Documentation quality	Extensive/complex	Good	Very good	Poor	Good	Good
Work with non-text objects	Yes	Manual ser/de	Built-in JSON	Built-in typedbytes	Built-in typedbytes	Unclear
Data formats supported	All	Text/manual	Text, Repr (string), JSON, Pickle	Text, SequenceFile (typedbytes), any Java InputFormat	Text, SequenceFile (typedbytes), Pickle	Text, SequenceFile for I/O (but unclear)
Implement custom SerDe	Yes	manual	Yes	Yes	No	No
Integrate with arbitrary Java classes	Yes	Yes	Experimental	Yes	Unclear	Unclear
Multistep MapReduce workflows	Yes, but awkward	No	Yes	Yes	No	No
Implement Partitioner in Python		No	No	No	No	Yes
HBase integration	Yes	No	No	Experimental / unofficial	Experimental	No
Support for AWS/EMR	Manual	Yes; EMR exposes Streaming API	Yes; flawlessly integrated through boto	Manually setup EC2 cluster	Setup EC2 with supplied Whirr config	No
Actively developed			Very	Somewhat	Yes	Yes
Commits in last year			1063	19	167	272
SLOC add+delete in last year			66094	1314	75091	172223
First commit			10/13/2010	6/15/2008	10/17/2009	3/10/2009
Sponsored?			Yelp	Individual (Last.fm?)	Individual	CRS4
Hosted on			GitHub	GitHub	GitHub	Sourceforge
License			Apache v2.0	Apache v2.0	GPL v3	Apache v2.0

- **All the Python frameworks look like pseudocode, which is a huge plus.**
- **mrjob seems highly active, easy-to-use, and mature.**
- **It makes multistep MapReduce flows easy, and can easily work with complex objects. It also works seamlessly with EMR.**
- **…But it appears to perform the slowest.**

http://blog.cloudera.com/blog/2013/01/a-guide-to-python-frameworks-for-hadoop/

- **Despite all of this we love MRJob**

- **End of section**

Fundamentals and Concept

- **Advice for Debugging**
  - Run in inline mode to make sure result is tractable
  - Use small data set, then you do not need to wait too long to receive result
  - Add mapper, combiner, reducer one by one and make sure the output of them is what you want
  - Test your functions outside hadoop/Mrjob

- **https://github.com/jblomo/pycon-mrjob/blob/master/code/unique_review.py**