L02: MapReduce

ANLY 502: Massive Data Fundamentals

Simson Garfinkel & Ghaleb Abdulla

January 25, 2016

Don’t be confused

Outline for today's class

Administrivia

Student Presentations

Review of Kalavri & Vlassov Paper

Understanding PS01: MapReduce in Java, Hadoop Streaming, and with mrjob

Administrivia 1 — Georgetown Student Pledge, Google Apps, FERPA, etc.

Some students haven't taken the Georgetown Student Pledge.

- You must take the pledge to pass this course.

You must use your Georgetown NetID to access the Google Drive.

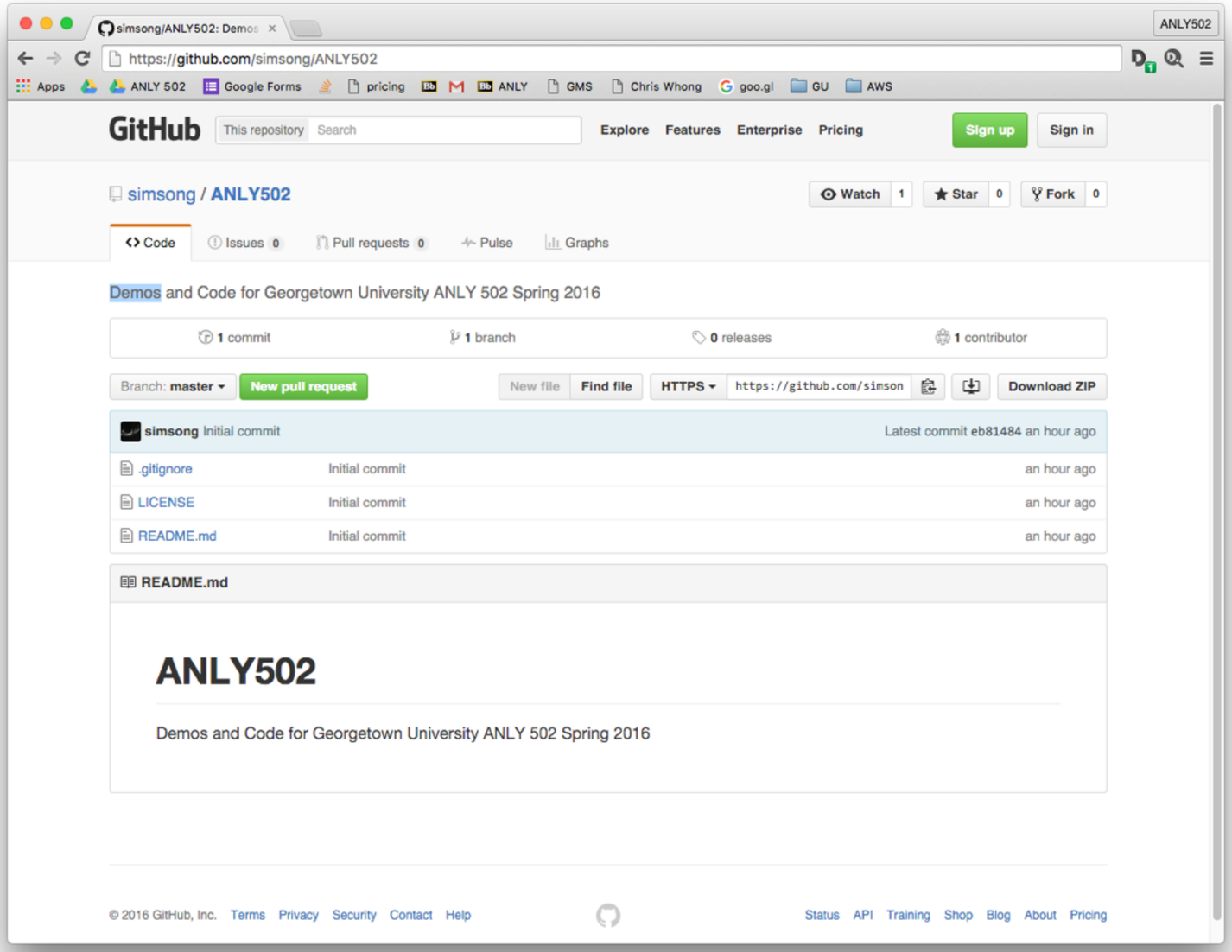
FERPA — Family Educational Rights and Privacy Act

- Protects everything you do, even your names!
- Limits use of non-Georgetown systems.

Administrivia 2 — GitHub Repo

Demo code.

Some solutions.



Administrivia 3 — Complications with online tutorial

Beware when looking at online tutorials:

- Version mismatch — tutorial may be for an old version of the software
- Installation mismatch — software may be installed at the wrong location

Administrivia 3 — Student Presentations

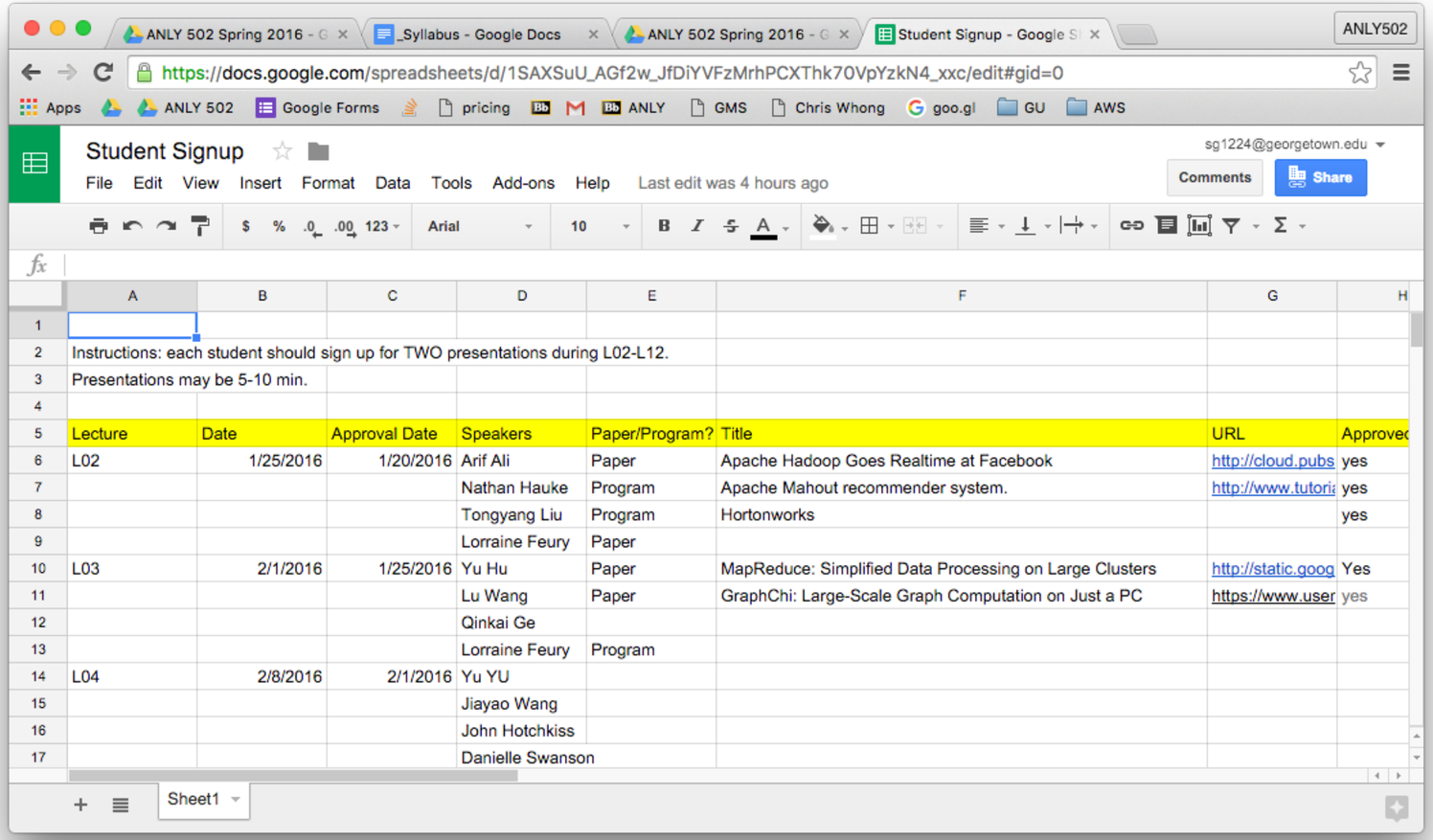
You all signed up to give a presentation!

Now you have to tell me what you will be presenting.

Please chose a ***paper*** and a ***program*** at least 2 weeks before each presentation!

You can prepare your presentation on Google Docs and share it with me!

- I'll add comments.
- No need to upload to BlackBoard (unless you want).



Lecture	Date	Approval Date	Speakers	Paper/Program?	Title	URL	Approved
L02	1/25/2016	1/20/2016	Arif Ali	Paper	Apache Hadoop Goes Realtime at Facebook	http://cloud.pubs	yes
			Nathan Hauke	Program	Apache Mahout recommender system.	http://www.tutoria	yes
			Tongyang Liu	Program	Hortonworks		yes
			Lorraine Feury	Paper			
L03	2/1/2016	1/25/2016	Yu Hu	Paper	MapReduce: Simplified Data Processing on Large Clusters	http://static.goog	Yes
			Lu Wang	Paper	GraphChi: Large-Scale Graph Computation on Just a PC	https://www.user	yes
			Qinkai Ge				
			Lorraine Feury	Program			
L04	2/8/2016	2/1/2016	Yu YU				
			Jiayao Wang				
			John Hotchkiss				
			Danielle Swanson				

Student Presentations

L02 — Student Presentations

Arif Ali	Paper	Apache Hadoop Goes Realtime at Facebook	http://cloud.pubs.dbs.uni-leipzig.de/sites/cloud.pubs.dbs.uni-leipzig.de/files/RealtimeHadoopSigmod2011.pdf
Nathan Hauke	Program	Apache Mahout recommender system.	http://www.tutorialspoint.com/mahout/mahout_quick_guide.htm
Tongyang Liu	Program	Hortonworks	
Lorraine Feury	Paper	Who supported Obama in 2012?	http://sethrf.com/files/ecological.pdf

Introducing Hadoop and MapReduce

Remember the basic map reduce idea:

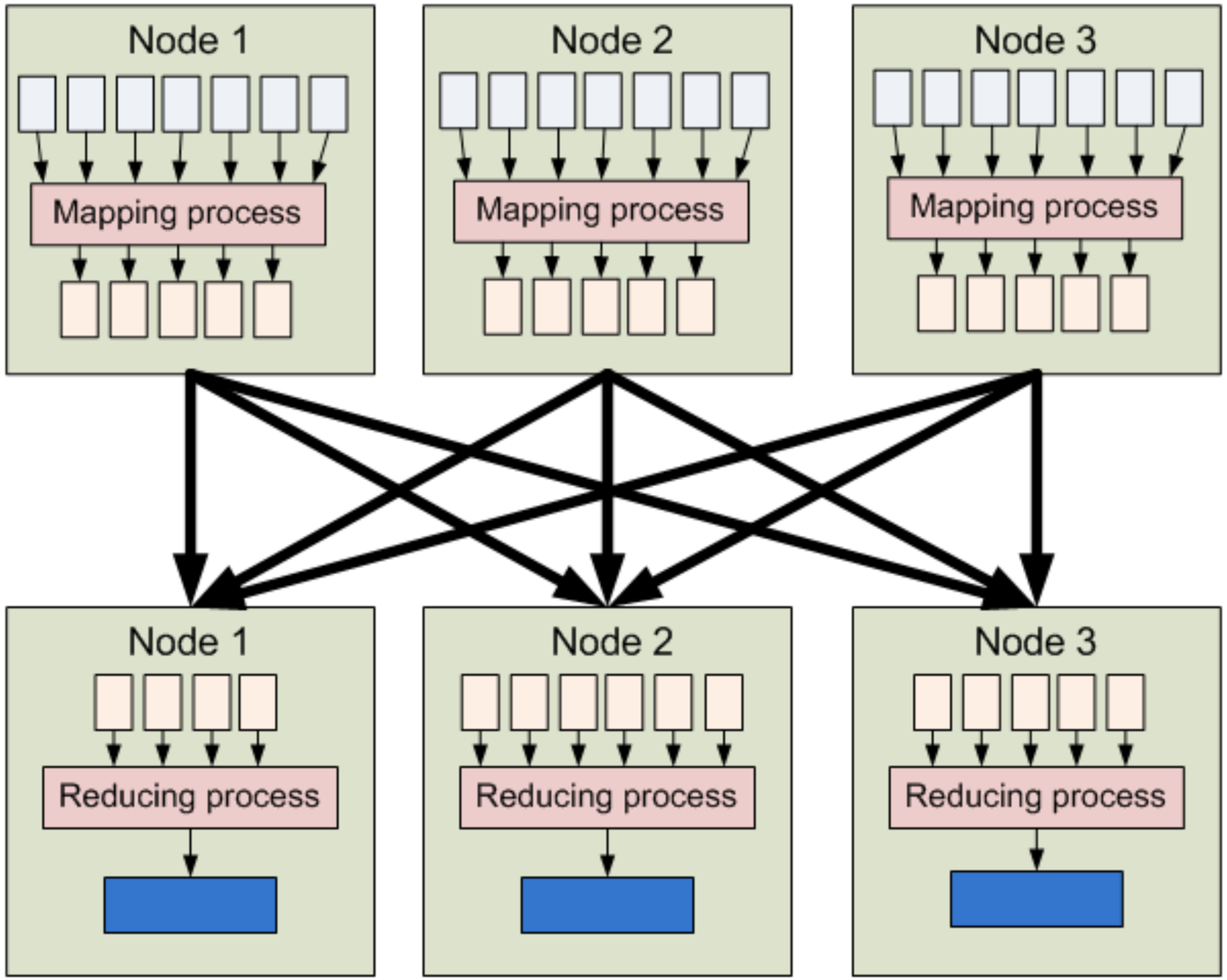
1. Data are distributed on nodes.
2. Data are split
3. Data sent to mapping processes
3. Map data are “shuffled” (sorted)
4. Data are reduced.

You must write:

- mapper
- reducer

You may also write:

- partitioner — describes how data are split



https://developer.yahoo.com/hadoop/tutorial/module1.html

"Word Count" is a common MapReduce demonstration program. This Word Count generates a word histogram.

The mapper:

```
// input_key: document name – ignored
// input_value: document contents

map(String input_key, String input_value):
   for each word w in input_value:
      EmitIntermediate(w, "1");
```

The reducer:

```
// output_key: a word
// output_values: a list of counts

reduce(String output_key, Iterator intermediate_values):
 int result = 0;
  for each v in intermediate_values:
    result += ParseInt(v);
  Emit(AsString(result));
```

"to be or not to be"
becomes:
to:1
be:1
or:1
not:1
to:1
be:1

The framework guarantees that "reduce" is called with all pairs of the same key.

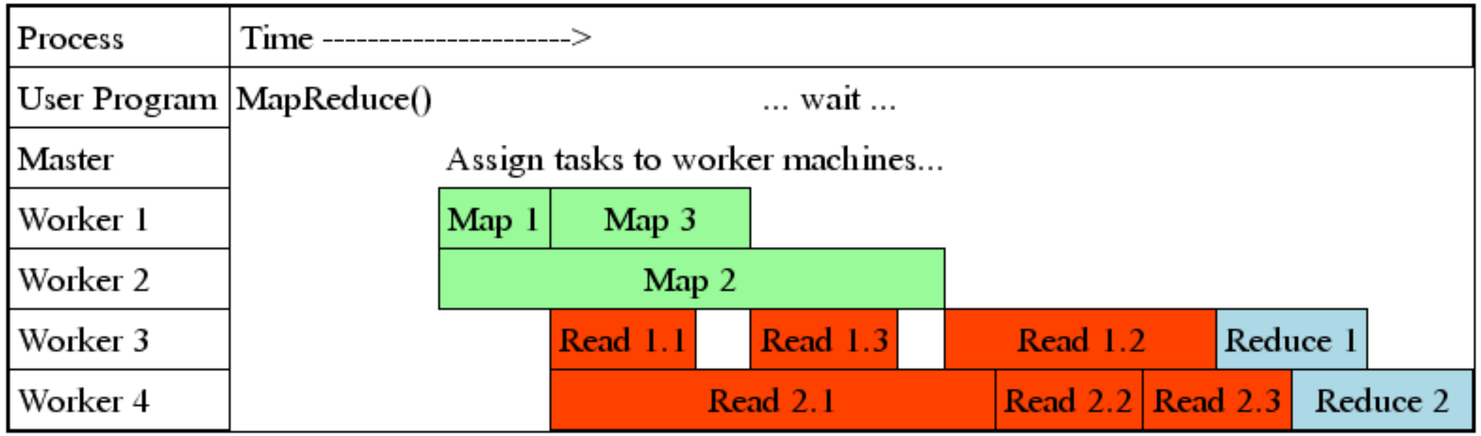
to:2
be:2
or:1
not:1

MapReduce pipelines execution and provides fault recovery

Workers run both map & reduce tasks.

- Each task is scheduled when data are available.
- Failed tasks (or slow machines) are automatically rescheduled.
- If the same data causes two mappers to fail, the data is ignored.

"Often use 200,000 map/5000 reduce tasks with 2000 machines"

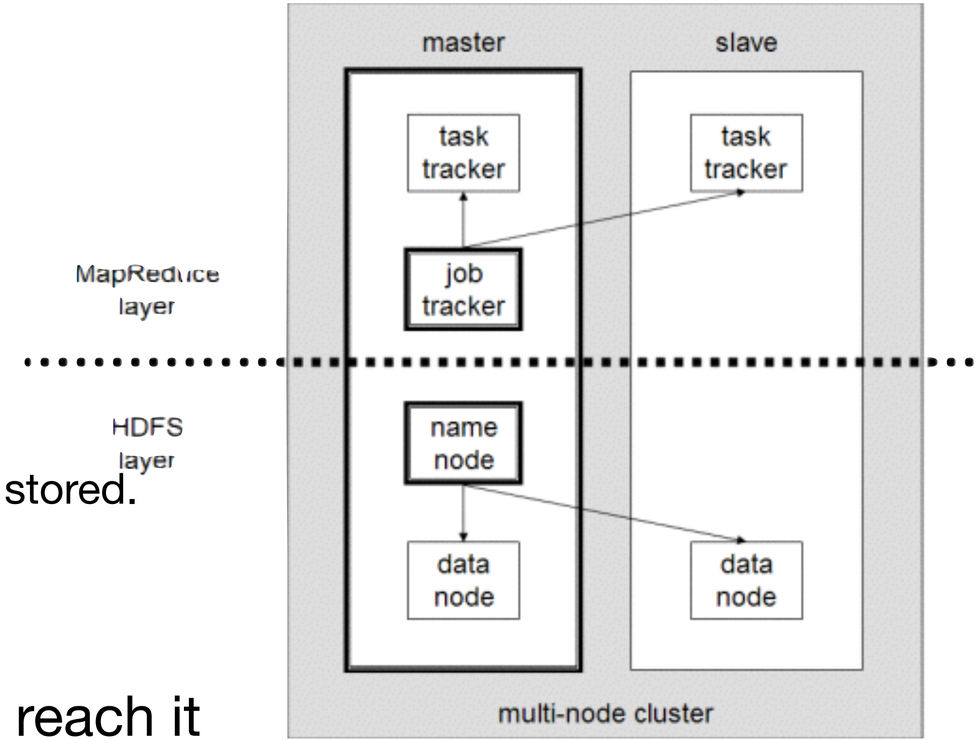


http://research.google.com/archive/mapreduce-osdi04-slides/index-auto-0009.html

Hadoop has two main systems: MapReduce and HDFS

MapReduce — Performs computation

- Job Tracker — Master planner
- Task Tracker — Runs each task



HDFS — Stores the data

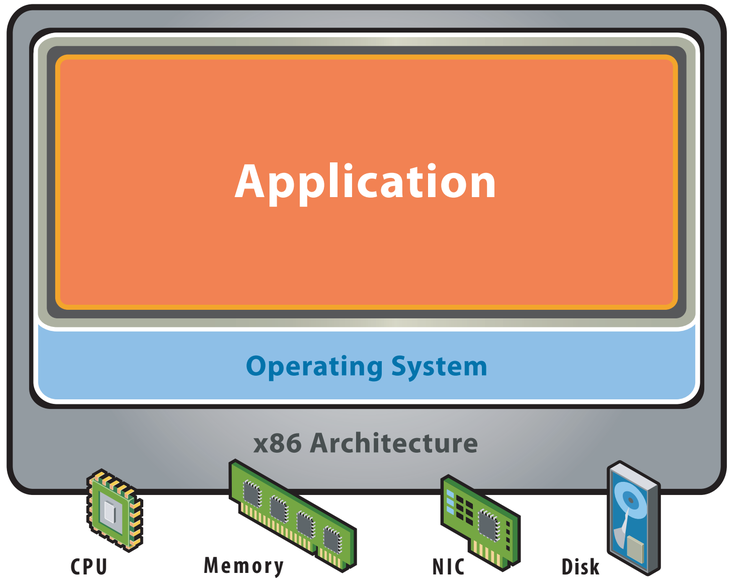
- Data Node — Stores the blocks for each file.
- Name Node — Keeps track of every file and where it is stored.
  —*Controls the Data Nodes.*

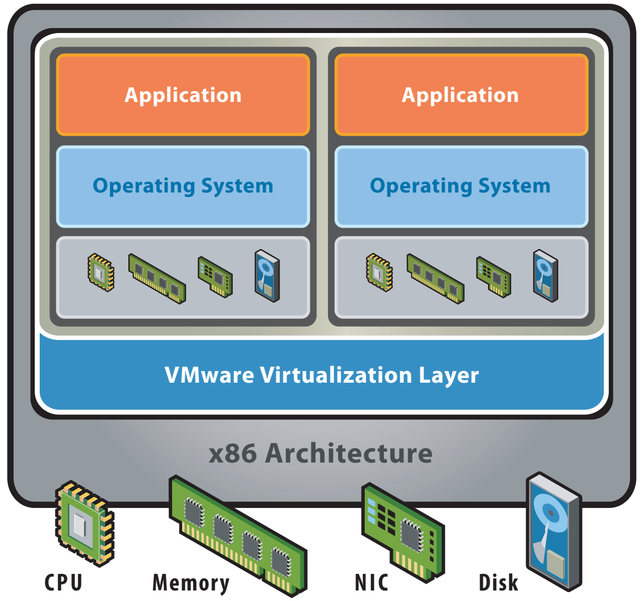
Data must be stored where MapReduce can reach it

- Hadoop MapReduce: HDFS or Amazon S3
- mrjob: in local file system, HDFS or S3

Getting to know the Cloudera QuickStart VM

Basic Idea: A computer within a computer





Before Virtualization:

- Single OS image per machine
- Software and hardware tightly coupled
- Running multiple applications on same machine often creates conflict
- Underutilized resources
- Inflexible and costly infrastructure

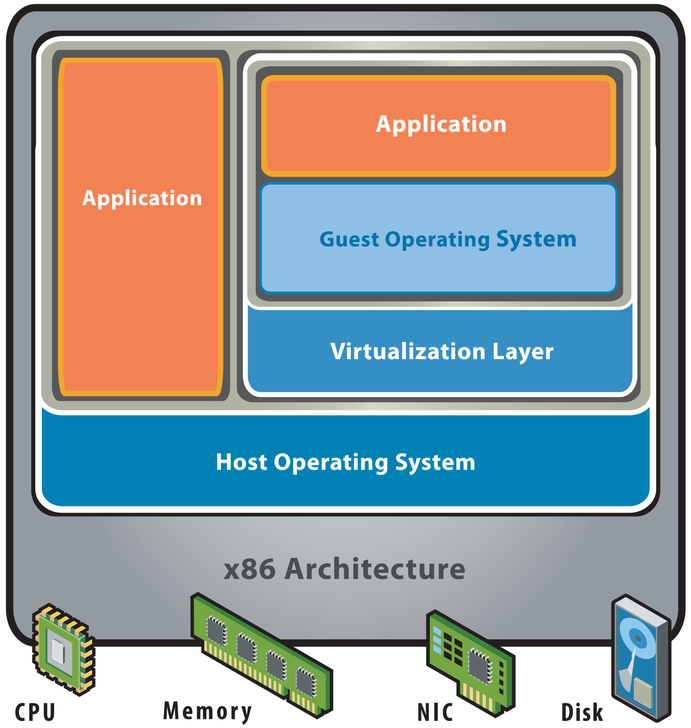
After Virtualization:

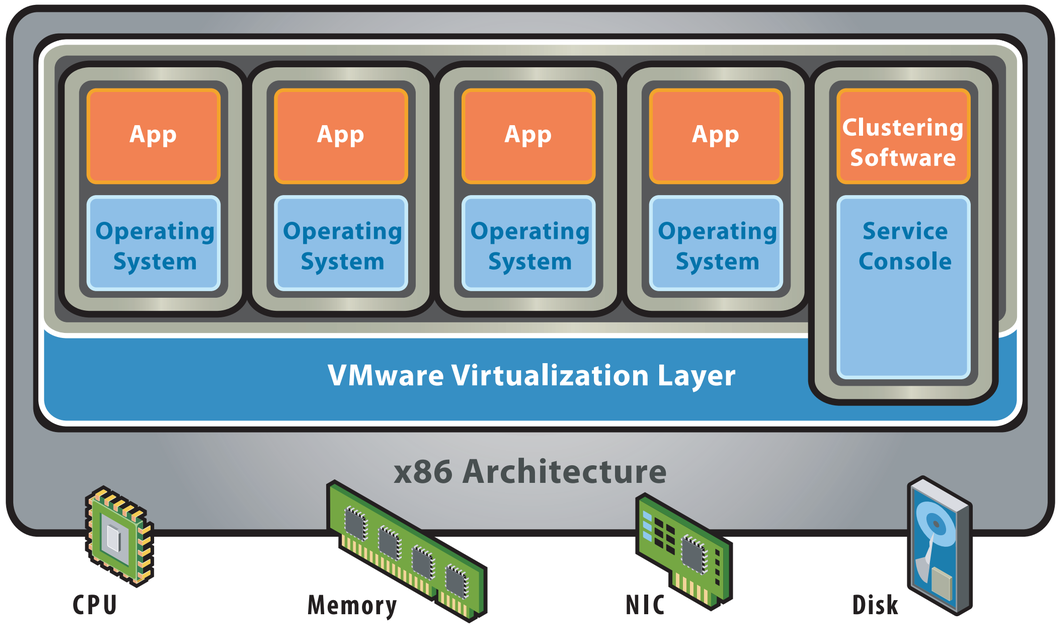
- Hardware-independence of operating system and applications
- Virtual machines can be provisioned to any system
- Can manage OS and application as a single unit by encapsulating them into virtual machines

VMWare Virtualization Overview

- https://www.vmware.com/pdf/virtualization.pdf

Two virtualization architectures: "Hosted" and "Bare-Metal"





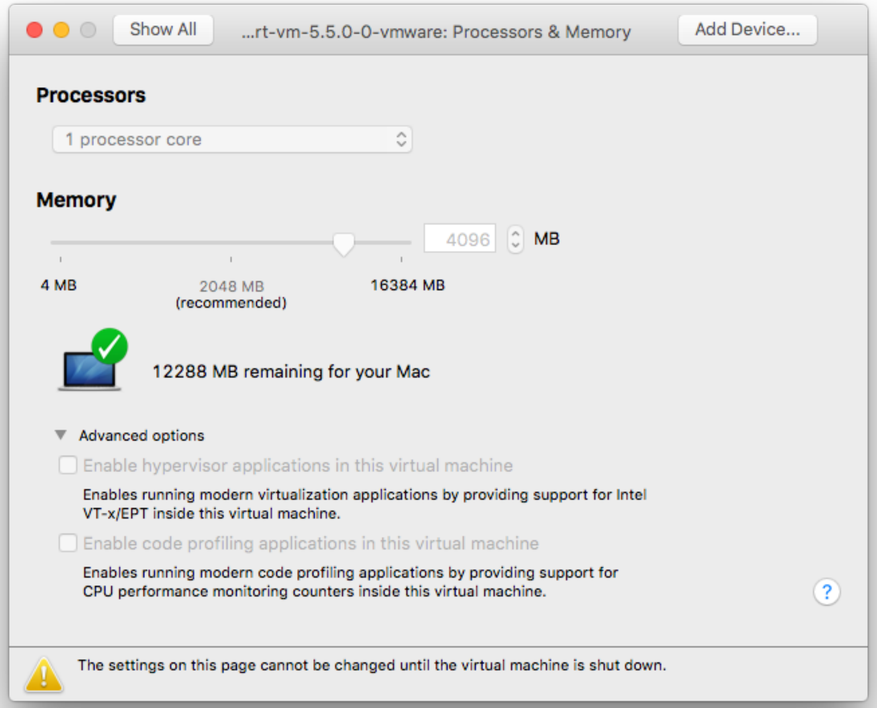
Hosted Architecture

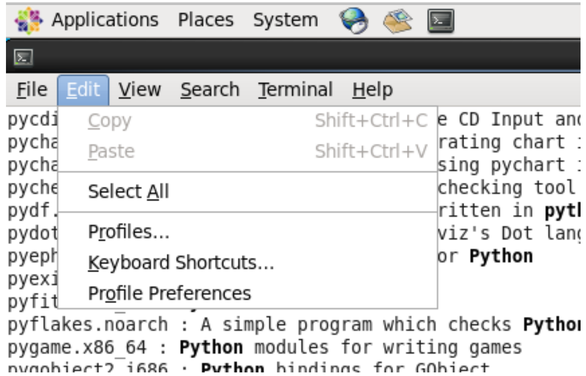
- Installs and runs as an application
- Relies on host OS for device support and physical resource management

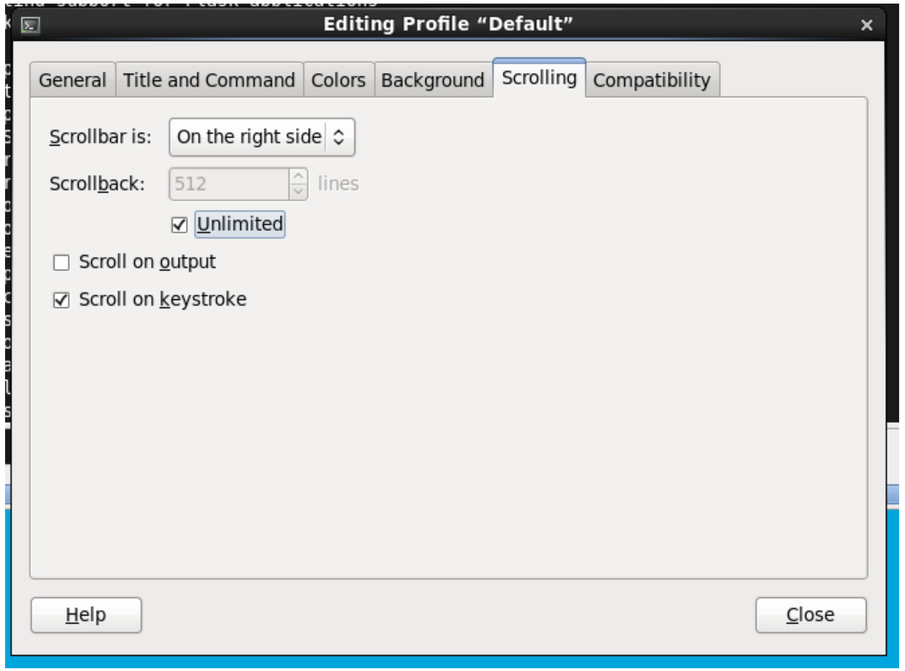
Bare-Metal (Hypervisor) Architecture

- Lean virtualization-centric kernel
- Service Console for agents and helper applications

Check VMWare Configuration




Applications Places System
File Edit View Search Terminal Help
Copy Shift+Ctrl+C
Paste Shift+Ctrl+V
Select All
Profiles...
Keyboard Shortcuts...
Profile Preferences
pyflakes.noarch : A simple program which checks Python
pygame.x86_64 : Python modules for writing games


Editing Profile "Default"
General
Title and Command
Colors
Background
Scrolling
Compatibility
Scrollbar is:
On the right side
Scrollback:
512
lines
Unlimited
Scroll on output
Scroll on keystroke
Help
Close

Setting up the VM for ANLY502

Install mrjob:

```
$ sudo yum install python-pip
$ sudo yum install mrjob
```

- Keep it up to date:

```
[cloudera@quickstart ~]$ sudo pip install mrjob
You are using pip version 7.1.0, however version 7.1.2 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
Requirement already satisfied (use --upgrade to upgrade): mrjob in /usr/lib/python2.6/site-packages
Requirement already satisfied (use --upgrade to upgrade): PyYAML>=3.08 in /usr/lib64/python2.6/site-packages (from mrjob)
Requirement already satisfied (use --upgrade to upgrade): boto>=2.6.0 in /usr/lib/python2.6/site-packages (from mrjob)
Requirement already satisfied (use --upgrade to upgrade): simplejson>=2.0.9 in /usr/lib64/python2.6/site-packages (from mrjob)
Requirement already satisfied (use --upgrade to upgrade): filechunkio in /usr/lib/python2.6/site-packages (from mrjob)
[cloudera@quickstart ~]$
```

Check to see if it is installed:

```
[cloudera@quickstart ~]$ python
Python 2.6.6 (r266:84292, Feb 22 2013, 00:00:18)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-3)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import mrjob
>>>
[cloudera@quickstart ~]$
```

You can also:

```
$ sudo yum install emacs
```

Python 2.6 / 2.7 / 3.x

Cloudera runs Python 2.6. It's out of date.

You can install some Python 2.7 features on Python 2.6:

```
[cloudera@quickstart ~]$ sudo yum search python | grep 7
 * base: mirrors.lga7.us.voxel.net
python-backport_collections.noarch : Backport of Python 2.7's collections module
python-fpconst.noarch : Python module for handling IEEE 754 floating point
python-importlib.noarch : Backport of importlib.import_module() from Python 2.7
python-ordereddict.noarch : Implementation of Python 2.7's OrderedDict
python-sqlalchemy0.7.x86_64 : Modular and flexible ORM library for python
python-unittest2.noarch : Backport of new unittest features for Python 2.7 to
[cloudera@quickstart ~]$
```

Then install it:

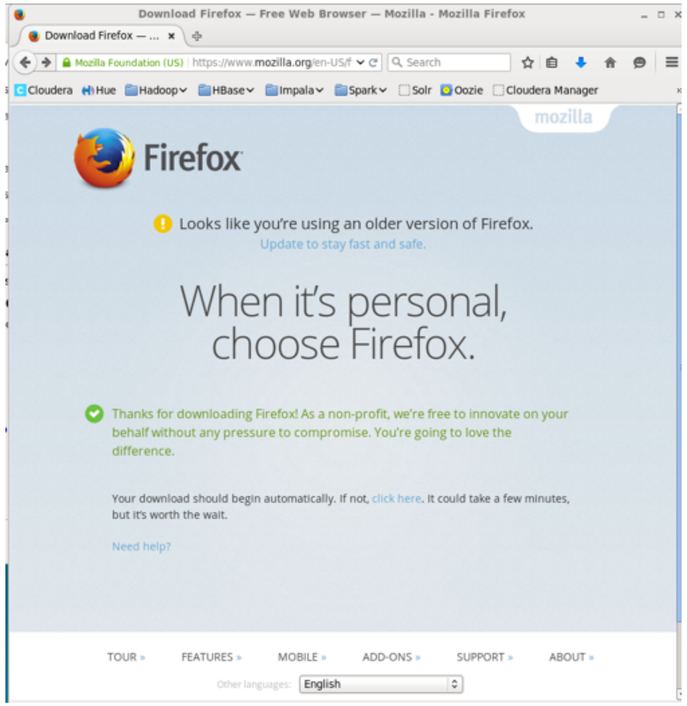
```
[cloudera@quickstart ~]$ sudo yum install python-backport_collections python-importlib
```

Keeping your VM up to date: Firefox

Manually download the new Firefox for Linux

—*For some reason, the automatic upgrade doesn't work.*

```
$ cd Downloads/
$ tar xfv firefox-f1.0.2.tar.gz2
$ sudo mv /usr/local/firefox /usr/local/firefox.old.$$
$ sudo mv firefox /usr/local/firefox
```



It's traditional to do "word count" as the map reduce equivalent of "Hello World."

Check out the git repository:

```
[cloudera@quickstart ~]$ git clone https://github.com/simsong/ANLY502.git
Initialized empty Git repository in /home/cloudera/ANLY502/.git/
remote: Counting objects: 18, done.
remote: Compressing objects: 100% (15/15), done.
remote: Total 18 (delta 1), reused 14 (delta 1), pack-reused 0
Unpacking objects: 100% (18/18), done.
[cloudera@quickstart ~]$


[cloudera@quickstart ~]$ ls -l ANLY502/
total 24
drwxrwxr-x 2 cloudera cloudera 4096 Jan 23 11:30 L01
drwxrwxr-x 2 cloudera cloudera 4096 Jan 23 11:30 L02
-rw-rw-r-- 1 cloudera cloudera 6555 Jan 23 11:30 LICENSE
drwxrwxr-x 2 cloudera cloudera 4096 Jan 23 11:30 PS01
-rw-rw-r-- 1 cloudera cloudera   72 Jan 23 11:30 README.md
[cloudera@quickstart ~]$
```

Key git commands you should know to get course material:

- git clone
- git pull

—*want to know more?*

- https://try.github.io/levels/1/challenges/1

PS01 has three kinds of word count

Approach #1 — WordCount in Java.

- https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html
- http://www.cloudera.com/documentation/other/tutorial/CDH5/Hadoop-Tutorial.html

- Requires text stored in HDFS. You need HDFS commands:
  - *hdfs dfs -put*
  - *hdfs dfs -ls*
  - *hdfs dfs -cat*
- Notes:
  - *Apache tutorial is better written; Cloudera tutorial has broken links.*
  - *Cloudera VM uses "hadoop fs" or "hdfs dfs"*
  - *Apache tutorial uses bin/hadoop fs*

Simple Hadoop with Java

Setting up the data

```
[cloudera@quickstart ~]$ hdfs dfs -mkdir data
[cloudera@quickstart ~]$ hdfs dfs -put file00? data/


[cloudera@quickstart ~]$ hdfs dfs -ls
Found 2 items
drwxr-xr-x   - cloudera cloudera          0 2016-01-17 19:08 data
drwxr-xr-x   - cloudera cloudera          0 2016-01-17 19:13 tmp


[cloudera@quickstart ~]$ hdfs dfs -ls data
Found 4 items
-rw-r--r--   1 cloudera cloudera         15 2016-01-17 19:08 data/file000
-rw-r--r--   1 cloudera cloudera         32 2016-01-17 19:08 data/file001
-rw-r--r--   1 cloudera cloudera         33 2016-01-17 19:08 data/file002
-rw-r--r--   1 cloudera cloudera         33 2016-01-17 19:08 data/file004
[cloudera@quickstart ~]$


[cloudera@quickstart ~]$ hdfs dfs -cat data/file000
this is a test
[cloudera@quickstart ~]$ hdfs dfs -cat data/file001
some days are nicer than others
[cloudera@quickstart ~]$ hdfs dfs -cat data/file002
you will never know what happens
[cloudera@quickstart ~]$ hdfs dfs -cat data/file004
you will never know what happens
[cloudera@quickstart ~]$
```

Alternative approach (from L01 slides)

Set up environment (not done for you by CVM):

```
$ export JAVA_CLASSPATH='/usr/lib/hadoop/client-0.20/*:/usr/lib/hadoop/*'
```

Compile WordCount.java and create a jar file:

```
$ javac -d wordcount_classes/ WordCount.java
$ jar -cvf wordcount.jar -C wordcount_classes
```

Put some data in HDFS:

```
$ echo "to be or not to be" > file0
$ echo "do be do be do" > file1
$ hdfs fs -mkdir /user/cloudera/wordcount
$ hdfs fs -mkdir /user/cloudera/wordcount/input
$ hdfs fs -put file0 /user/cloudera/wordcount/input/
$ hadoop fs -put file1 /user/cloudera/wordcount/input/
```

Run it!

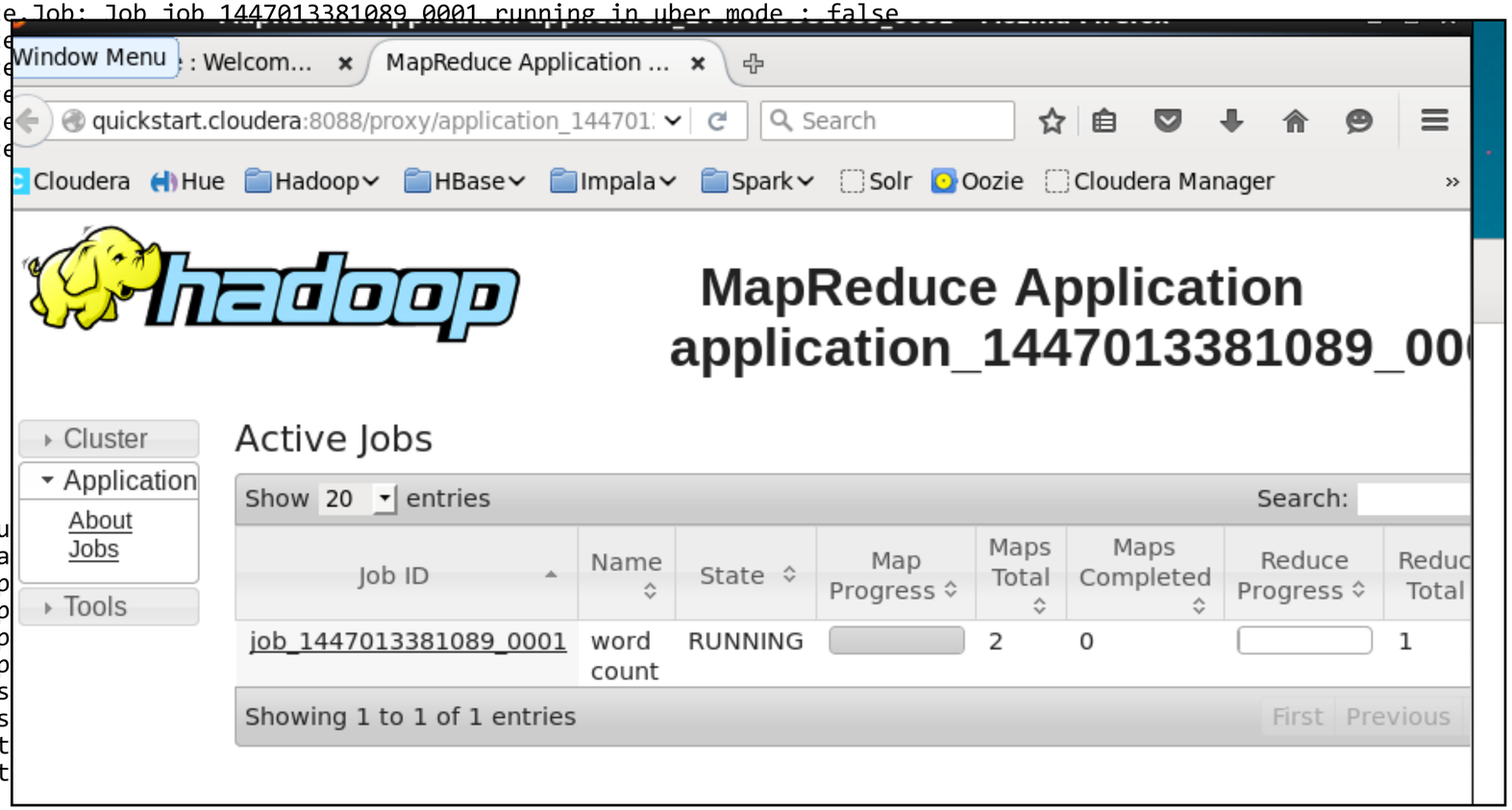
```
$ hadoop jar wordcount.jar WordCount /user/cloudera/wordcount/input/ \
                                     /user/cloudera/wordcount/output/
```

Examine the output

```
[cloudera@quickstart PS01]$ hadoop jar wc.jar WordCount data output1
16/01/23 12:02:58 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
16/01/23 12:02:59 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool
interface and execute your application with ToolRunner to remedy this.
16/01/23 12:03:00 INFO input.FileInputFormat: Total input paths to process : 4
16/01/23 12:03:00 INFO mapreduce.JobSubmitter: number of splits:4
16/01/23 12:03:00 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1453576193365_0002
16/01/23 12:03:01 INFO impl.YarnClientImpl: Submitted application application_1453576193365_0002
16/01/23 12:03:01 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/
application_1453576193365_0002/
16/01/23 12:03:01 INFO mapreduce.Job: Running job: job_1453576193365_0002
16/01/23 12:03:14 INFO mapreduce.Job: Job job_1453576193365_0002 running in uber mode : false
16/01/23 12:03:14 INFO mapreduce.Job:  map 0% reduce 0%
16/01/23 12:03:43 INFO mapreduce.Job:  map 25% reduce 0%
16/01/23 12:03:45 INFO mapreduce.Job:  map 50% reduce 0%
16/01/23 12:03:47 INFO mapreduce.Job:  map 75% reduce 0%
16/01/23 12:03:48 INFO mapreduce.Job:  map 100% reduce 0%
16/01/23 12:03:56 INFO mapreduce.Job:  map 100% reduce 100%
16/01/23 12:03:56 INFO mapreduce.Job: Job job_1453576193365_0002 completed successfully
```

Runtime output…

```
$ hadoop jar wordcount.jar WordCount /user/cloudera/wordcount/input/ /user/cloudera/wordcount/output/
15/11/08 13:57:01 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
15/11/08 13:57:02 WARN mapreduce.JobSubmitter: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application
with ToolRunner to remedy this.
15/11/08 13:57:02 INFO input.FileInputFormat: Total input paths to process : 2
15/11/08 13:57:03 INFO mapreduce.JobSubmitter: number of splits:2
15/11/08 13:57:03 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1447013381089_0001
15/11/08 13:57:03 INFO impl.YarnClientImpl: Submitted application application_1447013381089_0001
15/11/08 13:57:03 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_1447013381089_0001/
15/11/08 13:57:03 INFO mapreduce.Job: Running job: job_1447013381089_0001
15/11/08 13:57:13 INFO mapreduce.Job: Job job_1447013381089_0001 running in uber mode : false
15/11/08 13:57:13 INFO mapreduce
15/11/08 13:57:24 INFO mapreduce
15/11/08 13:57:30 INFO mapreduce
15/11/08 13:57:31 INFO mapreduce
15/11/08 13:57:31 INFO mapreduce
      File System Counters
                  FILE: Number
                  FILE: Number
                  FILE: Number
                  FILE: Number
                  FILE: Number
                  HDFS: Number
                  HDFS: Number
                  HDFS: Number
                  HDFS: Number
                  HDFS: Number
      Job Counters
                  Launched map
                  Launched redu
                  Data-local ma
                  Total time sp
                  Total time sp
                  Total time sp
                  Total time sp
                  Total vcore-s
                  Total vcore-s
                  Total megabyt
                  Total megabyt
```



To see the output:

What it looks like:

```
$ hdfs dfs -ls /user/cloudera/wordcount/output/
Found 2 items
-rw-r--r--   1 cloudera cloudera          0 2015-11-08 13:57 /user/cloudera/wordcount/output/_SUCCESS
-rw-r--r--   1 cloudera cloudera         26 2015-11-08 13:57 /user/cloudera/wordcount/output/part-r-00000
```

—*(remove “cloudera cloudera”)*

```
$ hdfs dfs -ls /user/cloudera/wordcount/output/
Found 2 items
-rw-r--r--   1          0 2015-11-08 13:57 /user/cloudera/wordcount/output/_SUCCESS
-rw-r--r--   1         26 2015-11-08 13:57 /user/cloudera/wordcount/output/part-r-00000
```

And the output:

```
$ hdfs dfs -tail /user/cloudera/wordcount/output/part-r-00000
be      4
do      3
not     1
or        1
to        2
```

Possible points of confusion

/user/cloudera — home directory in HDFS

/home/cloudera — home directory in Linux host file system (ext4)

If you follow the Apache MapReduce tutorial, you might do it like this:

VM5.5 sets JAVA_HOME and PATH but not HADOOP_CLASSPATH

```
[cloudera@quickstart PS01]$ echo $JAVA_HOME
/usr/java/jdk1.7.0_67-cloudera
[cloudera@quickstart PS01]$ echo $PATH
/usr/local/firefox:/sbin:/usr/java/jdk1.7.0_67-cloudera/bin:/usr/local/apache-ant/apache-ant-1.9.2/bin:/usr/local/apache-maven/
apache-maven-3.0.4/bin:/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin:/home/cloudera/bin
[cloudera@quickstart PS01]$ printenv | grep HADOOP
[cloudera@quickstart PS01]$ export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar
```

Compile:

```
[cloudera@quickstart PS01]$ hadoop com.sun.tools.javac.Main WordCount.java
[cloudera@quickstart PS01]$

[cloudera@quickstart PS01]$ ls -l
total 16
-rw-rw-r-- 1 cloudera cloudera 1501 Jan 23 11:57 WordCount.class
-rw-rw-r-- 1 cloudera cloudera 1739 Jan 23 11:57 WordCount$IntSumReducer.class
-rw-rw-r-- 1 cloudera cloudera 2260 Jan 23 11:30 WordCount.java
-rw-rw-r-- 1 cloudera cloudera 1736 Jan 23 11:57 WordCount$TokenizerMapper.class
[cloudera@quickstart PS01]$
```

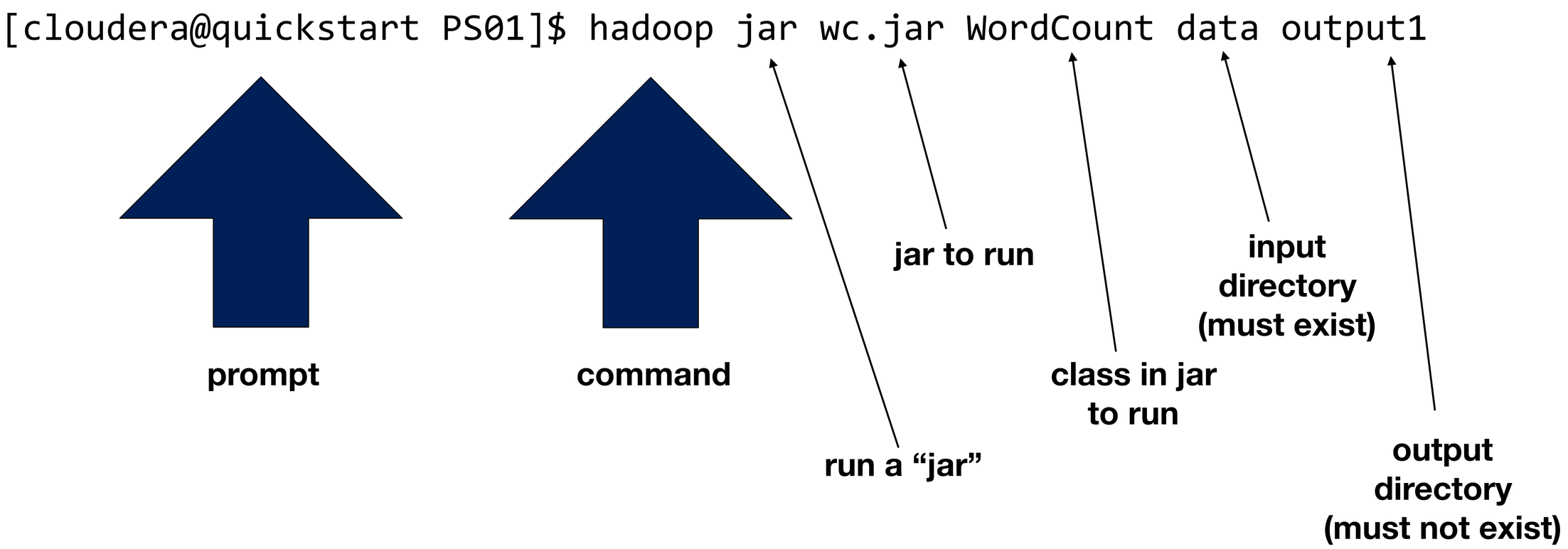
Make a “jar” file:

```
[cloudera@quickstart PS01]$ jar cf wc.jar WordCount*.class
[cloudera@quickstart PS01]$
```

Run it:

```
[cloudera@quickstart PS01]$ hadoop jar wc.jar WordCount data output1
```

It's really the same command…



Approaches for running Hadoop MapReduce jobs

Java (native) ✔

- Advantages:
  - *Fast — data stays within Java VM*
  - *Few dependencies — Everything in a .jar file*
- Disadvantages:
  - *Not everybody knows Java*
  - *Text processing in Java is hard*

Hadoop “Streaming” API

- Mapper & Reducer read from stdin to stdout. Fields separated by \t
- Advantage — Easy to integrate with existing code.
- Disadvantage — High overhead

mrjob

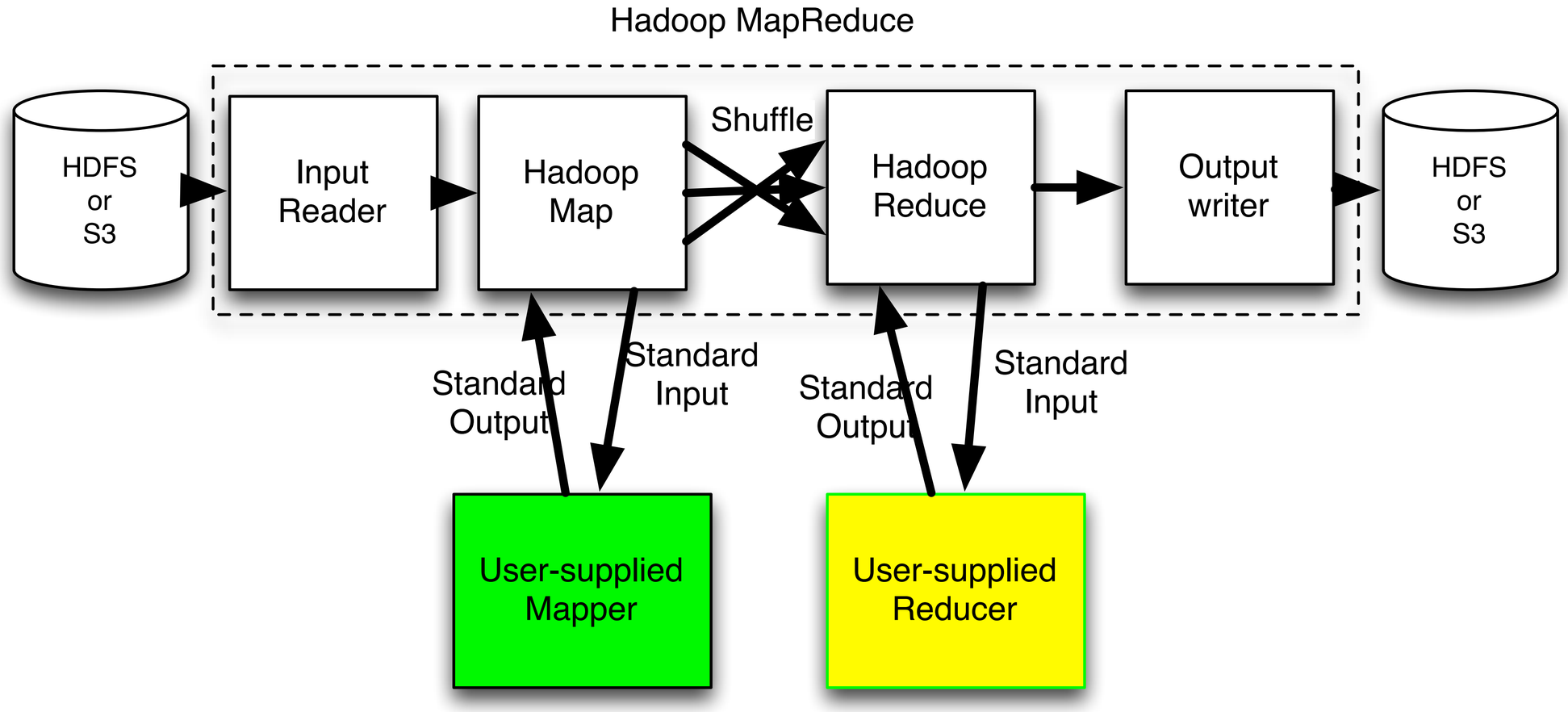
- Python implementation sits on top of Hadoop Streaming.
- Advantage — Powerful. Local testing.
- Disadvantage — High overhead

Hadoop Streaming

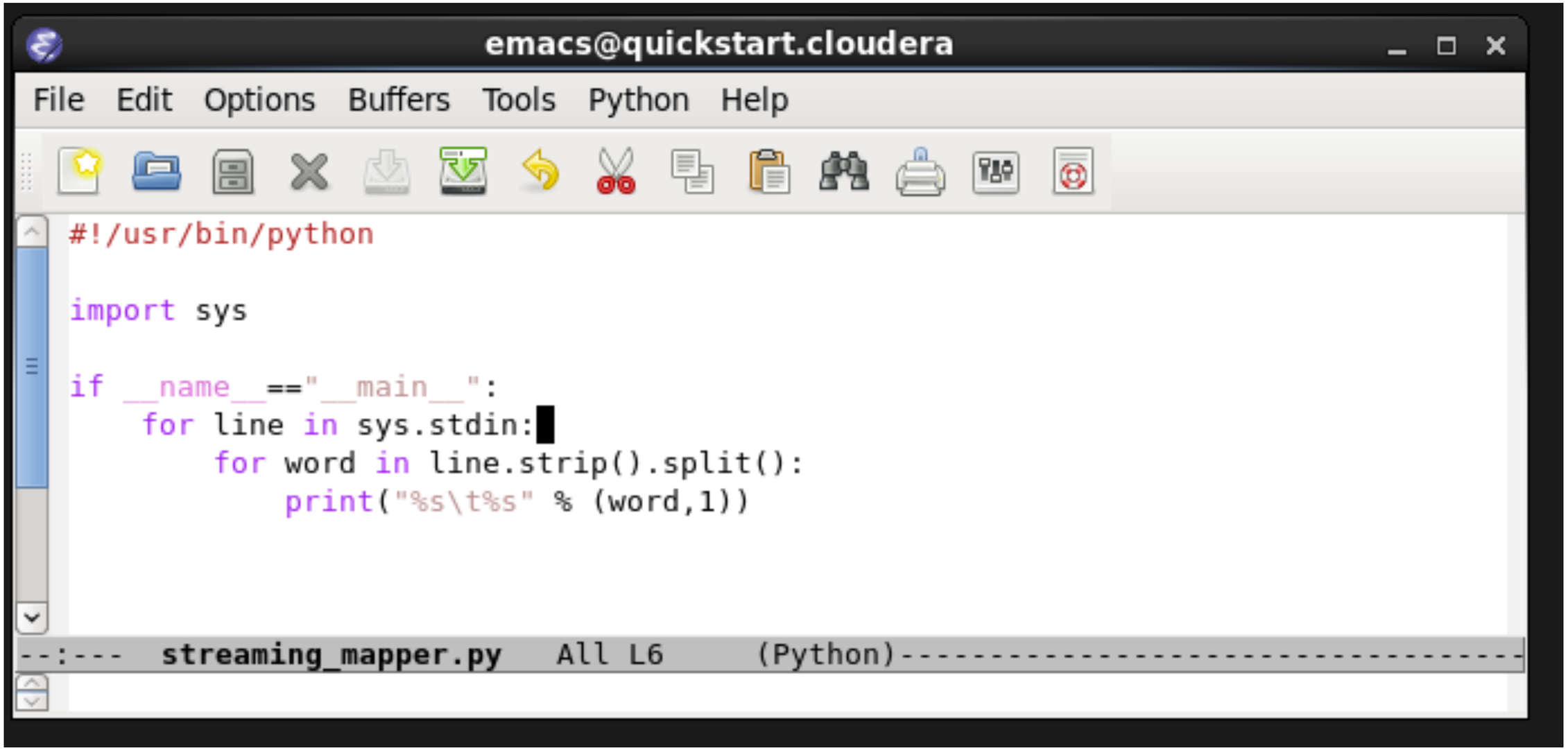
Hadoop Streaming

Hadoop streaming — reads from stdin & writes to stdout.

- Allows using Hadoop MapReduce with any language.
- Performance penalty — all I/O has to go over pipes.



Use Python programs as a mapper:



Use python program as reducer:

emacs@quickstart.cloudera

File Edit Options Buffers Tools Python Help

```
#!/usr/bin/python

import sys

word_last = None
word_count = 0

if __name__=="__main__":
    for line in sys.stdin:
        (word,count) = line.strip().split("\t")

        # convert count to int or skip this line
        try:
            count = int(count)
        except ValueError:
            continue

        # If the word has changed, output the previous total
        if word!=word_last:
            if word_last:
                print("%s\t%s" % (word_last,word_count))
            word_last = word
            word_count = count
        else:
            word_count += count

    # End of loop; print out the last value
    if word_last:
        print("%s\t%s" % (word_last,word_count))
```

-U:--- **streaming_reducer.py**   All L21   (Python)----------

You can independently test the mapper and reducer:

Mapper:

```
[cloudera@quickstart PS01]$ echo "one two two three three three" | ./streaming_mapper.py
one  1
two  1
two  1
three           1
three           1
three           1
[cloudera@quickstart PS01]$
```

Mapper and Reducer:

```
[cloudera@quickstart PS01]$ cat infile.txt
one two two three three three
[cloudera@quickstart PS01]$ cat infile.txt | ./streaming_mapper.py | sort | ./streaming_reducer.py
one  1
three           3
two  2
[cloudera@quickstart PS01]$
```

Hadoop Streaming Example

Simple Hadoop Streaming command on CDH Quickstart VM:

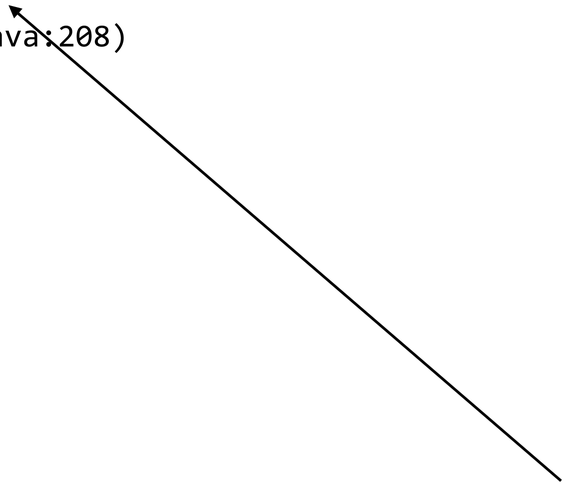
```
$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \
  -input data -output data2 \
  -mapper streaming_mapper.py -reducer streaming_reducer.py
```

Note: This may not work!

Debugging

if you see this:

```
Caused by: java.lang.reflect.InvocationTargetException
     at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
     at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
     at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
     at java.lang.reflect.Method.invoke(Method.java:606)
     at org.apache.hadoop.util.ReflectionUtils.setJobConf(ReflectionUtils.java:106)
     ... 17 more
Caused by: java.lang.RuntimeException: configuration exception
     at org.apache.hadoop.streaming.PipeMapRed.configure(PipeMapRed.java:221)
     at org.apache.hadoop.streaming.PipeMapper.configure(PipeMapper.java:66)
     ... 22 more
Caused by: java.io.IOException: Cannot run program "streaming_mapper.py": error=2, No such file or directory
     at java.lang.ProcessBuilder.start(ProcessBuilder.java:1047)
     at org.apache.hadoop.streaming.PipeMapRed.configure(PipeMapRed.java:208)
     ... 23 more
Caused by: java.io.IOException: error=2, No such file or directory
     at java.lang.UNIXProcess.forkAndExec(Native Method)
     at java.lang.UNIXProcess.<init>(UNIXProcess.java:186)
     at java.lang.ProcessImpl.start(ProcessImpl.java:130)
     at java.lang.ProcessBuilder.start(ProcessBuilder.java:1028)
     ... 24 more
```



Look for errors that you can understand

Hadoop Streaming Coudn't find the executable

```
Caused by: java.io.IOException: Cannot run program "streaming_mapper.py": error=2, No such file or directory
```

Streaming tries to run the programs from the your home directory!

- streaming_mapper.py and streaming_reducer.py were in the *current* directory
- You need to specify the complete path name.
- `pwd` evaluates to the current directory

Revised Simple Hadoop Streaming command on CDH Quickstart VM:

```
$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \
  -input data -output data2 \
  -mapper `pwd`/streaming_mapper.py -reducer `pwd`/streaming_reducer.py
```

You can specify more complex things

Sample command from tutorial:

```
$ hadoop jar hadoop-streaming-2.7.1.jar \
  -D mapreduce.map.output.key.field.separator=. \
  -D mapreduce.partition.keypartitioner.options=-k1,2 \
  -D mapreduce.fieldsel.data.field.separator=. \
  -D mapreduce.fieldsel.map.output.key.value.fields.spec=6,5,1-3:0- \
  -D mapreduce.fieldsel.reduce.output.key.value.fields.spec=0-2:5- \
  -D mapreduce.map.output.key.class=org.apache.hadoop.io.Text \
  -D mapreduce.job.reduces=12 \
  -input myInputDirs \
  -output myOutputDir \
  -mapper org.apache.hadoop.mapred.lib.FieldSelectionMapReduce \
  -reducer org.apache.hadoop.mapred.lib.FieldSelectionMapReduce \
  -partitioner org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner
```

Specifies:

- *mapper, reducer & partitioner*
- *Different field separators*
- *Output formats.*
- *etc.*

References

Good tutorials on Hadoop Streaming: streaming

- http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/

mrjob

mrjob is a python framework for Hadoop MapReduce

Advantages	Disadvantages
Allows you to write 100% Python	Hard to debug errors. (They frequently throw Java exceptions.)
Automatically moves data from filesystem → HDFS → filesystem/stdout	Can be confusing when data stored in HDFS. Can introduce performance penalties.
Map & Reduce with python data structures. Easily handle JSON. Write compressed data.	Relies on Python Streaming, so everything is sent over stdin/stdout. Creates opportunities for errors.
Many “hooks” for processing before & after steps. Easy to build multi-step pipelines.	Documentation is terse. Hard to debug.
Very flexible. Many ways to test & run jobs.	Potential for confusion.

mrjob depends on python iterators and generators to handle data that's larger than RAM.

Python supports iteration, e.g. in **for** loops:

- Lists, sets & dicts are iterables.

```
>>> for a in [1,2,3]:
...     print(a)
...
1
2
3
```

The "yield" statement lets you turn any function into a *generator,* which is also iterable:

```
#!/usr/bin/env python2

def get_one():
    yield 10
    yield 20
    yield 30
    yield 40

if __name__=="__main__":
    for x in get_one():
        print("get_more returned: %s" % (x))
```

Because of the "yield" statement, get_one() returns a generator.
A generator is a function that can be called multiple times.
Each time is is called, it continues off after the previous "yield."

```
$ python yield_demo.py
get_more returned: 10
get_more returned: 20
get_more returned: 30
get_more returned: 40
```

Big advantage of generators:
Process lists of any size without storing list in RAM.

"mrjob" uses "yield" to work with large files.

With mrjob:

- You write a class that implements mapper, reducer, etc.
- You run the program, which runs the MRJob routines…

mrjob takes care of:

- Splitting input data
- Getting your script to the remote note
- Copying files to HDFS & copying results back

```
#!/usr/bin/env python2
#
# Wordcount with mrjob


from mrjob.job import MRJob

class WordCount(MRJob):
    def mapper(self, _, line):
        for word in line.strip().split():
            yield word,1

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


if __name__=="__main__":
    WordCount.run()
```

mrjob help

```
$ python word_count.py --help
Usage: word_count.py [options] [input files]

Options:
  --help-emr            show EMR-related options
  --help-hadoop         show Hadoop-related options
  --help                show this message and exit
  --help-runner         show runner-related options

  Running specific parts of the job:
    --combiner          run a combiner
    --mapper            run a mapper
    --reducer           run a reducer
    --steps             print the mappers, combiners, and reducers that this
                        job defines
    --step-num=STEP_NUM
                        which step to execute (default is 0)

  Protocols:
    --strict-protocols  If something violates an input/output protocol then
                        raise an exception
    --no-strict-protocols
                        If something violates an input/output protocol then
                        increment a counter and continue
$
```

Three ways to run mrjob:

Runner:

- run jobs locally without Hadoop
  - —*Within a single Python Process*
  - —*With sub-processes and PIPE I/O*
- run jobs on local Hadoop Cluster — You need to install mrjob first and log into the master node.
- run jobs on ElasticMapReduce — mrjob starts up EMR and runs it.

General approach:

1. Run locally within a single python process and a reduced data set
2. Run locally with PIPE IO
3. Spin up a cluster, install mrjob, and try it out.
4. Have mrjob create and kill clusters for production.

Remember: anything stored in HDFS is lost when an EMR cluster shuts down!

- —*But things stored in S3 are preserved*

mrjob -r inline — runs the MapReduce as part of the local python interpreter

```
[cloudera@quickstart PS01]$ python mrjob_wordcount.py -r inline hamlet.txt
no configs found; falling back on auto-configuration
no configs found; falling back on auto-configuration
creating tmp directory /tmp/mrjob_wordcount.cloudera.20160123.225841.866456

PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by default. It's recommended you run your job with --strict-
protocols or set up mrjob.conf as described at https://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocols

writing to /tmp/mrjob_wordcount.cloudera.20160123.225841.866456/step-0-mapper_part-00000
Counters from step 1:
  (no counters found)
writing to /tmp/mrjob_wordcount.cloudera.20160123.225841.866456/step-0-mapper-sorted
> sort /tmp/mrjob_wordcount.cloudera.20160123.225841.866456/step-0-mapper_part-00000
writing to /tmp/mrjob_wordcount.cloudera.20160123.225841.866456/step-0-reducer_part-00000
Counters from step 1:
  (no counters found)
Moving /tmp/mrjob_wordcount.cloudera.20160123.225841.866456/step-0-reducer_part-00000 -> /tmp/mrjob_wordcount.cloudera.
20160123.225841.866456/output/part-00000
Streaming final output from /tmp/mrjob_wordcount.cloudera.20160123.225841.866456/output
"'tis"          1
"And"           1
"Arms"          1
"Arrows"        1
"Fortune,"      1
"Nobler"        1
"Or" 1
"Sea"           1
...
```

mrjob -r local — runs mrjob, but with mapper & reducer as subprocesses (à la Hadoop Streaming)

```
[cloudera@quickstart PS01]$ python mrjob_wordcount.py -r local hamlet.txt
no configs found; falling back on auto-configuration
no configs found; falling back on auto-configuration
creating tmp directory /tmp/mrjob_wordcount.cloudera.20160123.230124.769589
writing wrapper script to /tmp/mrjob_wordcount.cloudera.20160123.230124.769589/setup-wrapper.sh

PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by default. It's recommended you run your job with --strict-protocols
or set up mrjob.conf as described at https://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocols

writing to /tmp/mrjob_wordcount.cloudera.20160123.230124.769589/step-0-mapper_part-00000
> sh -ex setup-wrapper.sh /usr/bin/python mrjob_wordcount.py --step-num=0 --mapper /tmp/mrjob_wordcount.cloudera.
20160123.230124.769589/input_part-00000 > /tmp/mrjob_wordcount.cloudera.20160123.230124.769589/step-0-mapper_part-00000
STDERR: + __mrjob_PWD=/tmp/mrjob_wordcount.cloudera.20160123.230124.769589/job_local_dir/0/mapper/0
STDERR: + exec
STDERR: + /usr/bin/python -c 'import fcntl; fcntl.flock(9, fcntl.LOCK_EX)'
STDERR: + export PYTHONPATH=/tmp/mrjob_wordcount.cloudera.20160123.230124.769589/job_local_dir/0/mapper/0/mrjob.tar.gz:
STDERR: + PYTHONPATH=/tmp/mrjob_wordcount.cloudera.20160123.230124.769589/job_local_dir/0/mapper/0/mrjob.tar.gz:
STDERR: + exec
STDERR: + cd /tmp/mrjob_wordcount.cloudera.20160123.230124.769589/job_local_dir/0/mapper/0
STDERR: + /usr/bin/python mrjob_wordcount.py --step-num=0 --mapper /tmp/mrjob_wordcount.cloudera.20160123.230124.769589/
input_part-00000
Counters from step 1:
  (no counters found)
writing to /tmp/mrjob_wordcount.cloudera.20160123.230124.769589/step-0-mapper-sorted
> sort /tmp/mrjob_wordcount.cloudera.20160123.230124.769589/step-0-mapper_part-00000
writing to /tmp/mrjob_wordcount.cloudera.20160123.230124.769589/step-0-reducer_part-00000
> sh -ex setup-wrapper.sh /usr/bin/python mrjob_wordcount.py --step-num=0 --reducer /tmp/mrjob_wordcount.cloudera.
20160123.230124.769589/input_part-00000 > /tmp/mrjob_wordcount.cloudera.20160123.230124.769589/step-0-reducer_part-00000
writing to /tmp/mrjob_wordcount.cloudera.20160123.230124.769589/step-0-reducer_part-00001
> sh -ex setup-wrapper.sh /usr/bin/python mrjob_wordcount.py --step-num=0 --reducer /tmp/mrjob_wordcount.cloudera.
20160123.230124.769589/input_part-00001 > /tmp/mrjob_wordcount.cloudera.20160123.230124.769589/step-0-reducer_part-00001
```

-r local — continued

```
STDERR: + __mrjob_PWD=/tmp/mrjob_wordcount.cloudera.20160123.230124.769589/job_local_dir/0/reducer/0
STDERR: + exec
STDERR: + /usr/bin/python -c 'import fcntl; fcntl.flock(9, fcntl.LOCK_EX)'
STDERR: + export PYTHONPATH=/tmp/mrjob_wordcount.cloudera.20160123.230124.769589/job_local_dir/0/reducer/0/mrjob.tar.gz:
STDERR: + PYTHONPATH=/tmp/mrjob_wordcount.cloudera.20160123.230124.769589/job_local_dir/0/reducer/0/mrjob.tar.gz:
STDERR: + exec
STDERR: + cd /tmp/mrjob_wordcount.cloudera.20160123.230124.769589/job_local_dir/0/reducer/0
STDERR: + /usr/bin/python mrjob_wordcount.py --step-num=0 --reducer /tmp/mrjob_wordcount.cloudera.20160123.230124.769589/
input_part-00000
STDERR: + __mrjob_PWD=/tmp/mrjob_wordcount.cloudera.20160123.230124.769589/job_local_dir/0/reducer/1
STDERR: + exec
STDERR: + /usr/bin/python -c 'import fcntl; fcntl.flock(9, fcntl.LOCK_EX)'
STDERR: + export PYTHONPATH=/tmp/mrjob_wordcount.cloudera.20160123.230124.769589/job_local_dir/0/reducer/1/mrjob.tar.gz:
STDERR: + PYTHONPATH=/tmp/mrjob_wordcount.cloudera.20160123.230124.769589/job_local_dir/0/reducer/1/mrjob.tar.gz:
STDERR: + exec
STDERR: + cd /tmp/mrjob_wordcount.cloudera.20160123.230124.769589/job_local_dir/0/reducer/1
STDERR: + /usr/bin/python mrjob_wordcount.py --step-num=0 --reducer /tmp/mrjob_wordcount.cloudera.20160123.230124.769589/
input_part-00001
Counters from step 1:
  (no counters found)
Moving /tmp/mrjob_wordcount.cloudera.20160123.230124.769589/step-0-reducer_part-00000 -> /tmp/mrjob_wordcount.cloudera.
20160123.230124.769589/output/part-00000
Moving /tmp/mrjob_wordcount.cloudera.20160123.230124.769589/step-0-reducer_part-00001 -> /tmp/mrjob_wordcount.cloudera.
20160123.230124.769589/output/part-00001
Streaming final output from /tmp/mrjob_wordcount.cloudera.20160123.230124.769589/output
"'tis"          1
"And"           1
"Arms"          1
"Arrows"        1
"Fortune,"      1
"Nobler"        1
```

mrjob -r hadoop — submits the job to Hadoop
Note: Needs a way to find Hadoop!

```
[cloudera@quickstart PS01]$ export HADOOP_HOME=/usr/lib/hadoop-mapreduce
[cloudera@quickstart PS01]$ python mrjob_wordcount.py -r hadoop --hadoop-bin /usr/bin/hadoop hamlet.txt
no configs found; falling back on auto-configuration
no configs found; falling back on auto-configuration
creating tmp directory /tmp/mrjob_wordcount.cloudera.20160123.230637.192023
writing wrapper script to /tmp/mrjob_wordcount.cloudera.20160123.230637.192023/setup-wrapper.sh
Using Hadoop version 2.6.0
Copying local files into hdfs:///user/cloudera/tmp/mrjob/mrjob_wordcount.cloudera.20160123.230637.192023/files/

PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by default. It's recommended you run your job with --strict-
protocols or set up mrjob.conf as described at https://pythonhosted.org/mrjob/whats-new.html#ready-for-strict-protocols

HADOOP: packageJobJar: [] [/usr/jars/hadoop-streaming-2.6.0-cdh5.5.0.jar] /tmp/streamjob4166488844051930017.jar tmpDir=null
HADOOP: Connecting to ResourceManager at /0.0.0.0:8032
HADOOP: Connecting to ResourceManager at /0.0.0.0:8032
HADOOP: Total input paths to process : 1
HADOOP: number of splits:2
HADOOP: Submitting tokens for job: job_1453576193365_0007
HADOOP: Submitted application application_1453576193365_0007
HADOOP: The url to track the job: http://quickstart.cloudera:8088/proxy/application_1453576193365_0007/
HADOOP: Running job: job_1453576193365_0007
HADOOP: Job job_1453576193365_0007 running in uber mode : false
HADOOP:  map 0% reduce 0%
HADOOP:  map 100% reduce 0%
HADOOP:  map 100% reduce 100%
HADOOP: Job job_1453576193365_0007 completed successfully
```

Cont...

```
HADOOP: Counters: 49
HADOOP:         File System Counters
HADOOP:                 FILE: Number of bytes read=482
HADOOP:                 FILE: Number of bytes written=351615
HADOOP:                 FILE: Number of read operations=0
HADOOP:                 FILE: Number of large read operations=0
HADOOP:                 FILE: Number of write operations=0
HADOOP:                HDFS: Number of bytes read=671
HADOOP:                 HDFS: Number of bytes written=339
HADOOP:                 HDFS: Number of read operations=9
HADOOP:                 HDFS: Number of large read operations=0
HADOOP:                 HDFS: Number of write operations=2
HADOOP:         Job Counters
HADOOP:                 Launched map tasks=2
HADOOP:                 Launched reduce tasks=1
HADOOP:                 Data-local map tasks=2
HADOOP:                 Total time spent by all maps in occupied slots (ms)=32332
HADOOP:                 Total time spent by all reduces in occupied slots (ms)=9372
HADOOP:                 Total time spent by all map tasks (ms)=32332
HADOOP:                 Total time spent by all reduce tasks (ms)=9372
HADOOP:                 Total vcore-seconds taken by all map tasks=32332
HADOOP:                 Total vcore-seconds taken by all reduce tasks=9372
HADOOP:                 Total megabyte-seconds taken by all map tasks=33107968
HADOOP:                 Total megabyte-seconds taken by all reduce tasks=9596928
```

Cont.

```
HADOOP:         Map-Reduce Framework
HADOOP:                 Map input records=1
HADOOP:                 Map output records=43
HADOOP:                 Map output bytes=390
HADOOP:                 Map output materialized bytes=488
HADOOP:                 Input split bytes=344
HADOOP:                 Combine input records=0
HADOOP:                 Combine output records=0
HADOOP:                 Reduce input groups=36
HADOOP:                 Reduce shuffle bytes=488
HADOOP:                 Reduce input records=43
HADOOP:                 Reduce output records=36
HADOOP:                 Spilled Records=86
HADOOP:                 Shuffled Maps =2
HADOOP:                 Failed Shuffles=0
HADOOP:                 Merged Map outputs=2
HADOOP:                 GC time elapsed (ms)=564
HADOOP:                 CPU time spent (ms)=3050
HADOOP:                 Physical memory (bytes) snapshot=563351552
HADOOP:                 Virtual memory (bytes) snapshot=4514258944
HADOOP:                 Total committed heap usage (bytes)=391520256
HADOOP:         Shuffle Errors
HADOOP:                 BAD_ID=0
HADOOP:                 CONNECTION=0
HADOOP:                 IO_ERROR=0
HADOOP:                 WRONG_LENGTH=0
HADOOP:                 WRONG_MAP=0
HADOOP:                 WRONG_REDUCE=0
```

Cont...

```
HADOOP:         File Input Format Counters
HADOOP:                   Bytes Read=327
HADOOP:         File Output Format Counters
HADOOP:                   Bytes Written=339
HADOOP: Output directory: hdfs:///user/cloudera/tmp/mrjob/mrjob_wordcount.cloudera.20160123.230637.192023/output
Counters from step 1:
  (no counters found)
Streaming final output from hdfs:///user/cloudera/tmp/mrjob/mrjob_wordcount.cloudera.20160123.230637.192023/output
"'tis"          1
"And"           1
"Arms"          1
"Arrows"        1
"Fortune,"      1
"Nobler"        1
"Or" 1
"Sea"           1
"Slings"        1
"The"           1
"To" 1
"Whether"       1
"a"  1
"against"       1
"and"           1
"be,"           2
"by" 1
"die,"          1
"end"           1
"in" 1
```

Approaches for debugging mrjob scripts

Work with small data sets first!

Build your script one step at a time.

Try -r inline, then -r local, then -r hadoop.

Use counters to keep track of how many times each section is processed.

Simple MapReduce tricks with mrjob

Compute multiple things at once!

You control the <k,v> pairs that are sent from the mapper to the reducer.

MapReduce sorts the keys and the values for your reducer.

- Easy way to compute multiple stats — compute different stats at the same time.
- Example, to count number of chars, words, lines:

```
from mrjob.job import MRJob

class MRWordFrequencyCount(MRJob):

    def mapper(self, _, line):
        yield "chars", len(line)
        yield "words", len(line.split())
        yield "lines", 1
    def reducer(self, key, values):
        yield key, sum(values)

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

Data:

```
This is a test
Just another test
```

Intermediate looks like:

```
<chars, 15>
<words, 4>
<lines, 1>
<chars, 18>
<words, 3>
<lines, 1>
```

Reducer Gets:

```
<chars, (15, 18)>
<lines, (1, 1)>
<words, (3, 4)>
```

Final output looks like:

```
chars  33
lines  2
words  7
```

Manipulate & filter in the mapper

Output from simple wordcount:

```
"'tis"          1
"And"           1
"Arms"          1
"Arrows"        1
"Fortune,"      1
"Nobler"        1
"Or" 1
"Sea"           1
"Slings"        1
"The"           1
"To" 1
"Whether"       1
"a"  1
"against"       1
"and"           1
"be,"           2
```

New mapper function:

```
class WordCount(MRJob):
    def mapper(self, _, line):
        for word in line.strip().split():
            word = filter(str.isalpha,word.lower())
            yield word,1

...
```

New Output from simple wordcount:

```
"a"  1
"against"       1
"and"           2
"arms"          1
"arrows"        1
"be" 2
"by" 1
"die"           1
"end"           1
"fortune"       1
"in" 1
"is" 1
"mind"          1
"nobler"        1
"not"           1
"of" 2
"opposing"      1
"or" 2
"outrageous"    1
"question"      1
```

MRJob.mapper_final() — a function that runs when the mapper is finished. Easy way to create top-10 list

Modify word count to print the 10 most popular words:

```
#!/usr/bin/env python2

from mrjob.job import MRJob
import heapq

TOPN = 10
class WordCount(MRJob):
    def mapper(self, _, line):
        for word in line.strip().split():
            word = filter(str.isalpha,word.lower())
            yield word,1

    def reducer_init(self):
        print("reducer_init")
        self.heap = []

    def reducer(self, key, values):
        heapq.heappush(self.heap,(sum(values),key))
        if len(self.heap) > TOPN:
            heapq.heappop(self.heap)
        print(self.heap)

    def reducer_final(self):
        for v in sorted(self.heap, reverse=True):
            yield v

if __name__=="__main__":
    WordCount.run()
```

```
6       "to"
3       "the"
2       "or"
2       "of"
2       "be"
2       "and"
1       "whether"
1       "troubles"
1       "tis"
1       "them"
```

The range of MRJob functions

- http://mrjob.readthedocs.org/en/latest/job.html

—*MRJob.reducer_init()*
—*MRJob.reducer(key, values)*
—*MRJob.reducer_final()*

—*MRJob.combiner_init()*
—*MRJob.combiner(key, values)* — *like the reducer(), but runs on each node before shuffling*
—*MRJob.combiner_final()*

—*MRJob.mapper_init()*
—*MRJob.mapper(key, value)*
—*MRJob.mapper_final()*

You can add print statements to see what's happening*

```
    def reducer(self, key, values):
        heapq.heappush(self.heap,(sum(values),key))
        if len(self.heap) > TOPN:
            heapq.heappop(self.heap)
        print(self.heap)
```

```
[(1, 'a')]
[(1, 'a'), (1, 'against')]
[(1, 'a'), (1, 'against'), (2, 'and')]
[(1, 'a'), (1, 'against'), (2, 'and'), (1, 'arms')]
[(1, 'a'), (1, 'against'), (2, 'and'), (1, 'arms'), (1, 'arrows')]
[(1, 'a'), (1, 'against'), (2, 'and'), (1, 'arms'), (1, 'arrows'), (2, 'be')]
[(1, 'a'), (1, 'against'), (1, 'by'), (1, 'arms'), (1, 'arrows'), (2, 'be'), (2, 'and')]
[(1, 'a'), (1, 'against'), (1, 'by'), (1, 'arms'), (1, 'arrows'), (2, 'be'), (2, 'and'), (1, 'die')]
[(1, 'a'), (1, 'against'), (1, 'by'), (1, 'arms'), (1, 'arrows'), (2, 'be'), (2, 'and'), (1, 'die'), (1, 'end')]
[(1, 'a'), (1, 'against'), (1, 'by'), (1, 'arms'), (1, 'arrows'), (2, 'be'), (2, 'and'), (1, 'die'), (1, 'end'), (1, 'fortune')]
[(1, 'against'), (1, 'arms'), (1, 'by'), (1, 'die'), (1, 'arrows'), (2, 'be'), (2, 'and'), (1, 'in'), (1, 'end'), (1, 'fortune')]
[(1, 'arms'), (1, 'arrows'), (1, 'by'), (1, 'die'), (1, 'fortune'), (2, 'be'), (2, 'and'), (1, 'in'), (1, 'end'), (1, 'is')]
[(1, 'arrows'), (1, 'die'), (1, 'by'), (1, 'end'), (1, 'fortune'), (2, 'be'), (2, 'and'), (1, 'in'), (1, 'mind'), (1, 'is')]
[(1, 'by'), (1, 'die'), (1, 'nobler'), (1, 'end'), (1, 'fortune'), (2, 'be'), (2, 'and'), (1, 'in'), (1, 'mind'), (1, 'is')]
[(1, 'die'), (1, 'end'), (1, 'nobler'), (1, 'in'), (1, 'fortune'), (2, 'be'), (2, 'and'), (1, 'not'), (1, 'mind'), (1, 'is')]
[(1, 'end'), (1, 'fortune'), (1, 'nobler'), (1, 'in'), (1, 'is'), (2, 'be'), (2, 'and'), (1, 'not'), (1, 'mind'), (2, 'of')]
[(1, 'fortune'), (1, 'in'), (1, 'nobler'), (1, 'mind'), (1, 'is'), (2, 'be'), (2, 'and'), (1, 'not'), (1, 'opposing'), (2, 'of')]
[(1, 'in'), (1, 'is'), (1, 'nobler'), (1, 'mind'), (2, 'of'), (2, 'be'), (2, 'and'), (1, 'not'), (1, 'opposing'), (2, 'or')]
[(1, 'is'), (1, 'mind'), (1, 'nobler'), (1, 'not'), (1, 'outrageous'), (2, 'be'), (2, 'and'), (2, 'of'), (1, 'opposing'), (2, 'or')]
[(1, 'mind'), (1, 'not'), (1, 'nobler'), (1, 'opposing'), (1, 'outrageous'), (2, 'be'), (2, 'and'), (2, 'of'), (1, 'question'), (2, 'or')]
[(1, 'nobler'), (1, 'not'), (1, 'sea'), (1, 'opposing'), (1, 'outrageous'), (2, 'be'), (2, 'and'), (2, 'of'), (1, 'question'), (2, 'or')]
[(1, 'not'), (1, 'opposing'), (1, 'sea'), (1, 'question'), (1, 'outrageous'), (2, 'be'), (2, 'and'), (2, 'of'), (1, 'sleep'), (2, 'or')]
[(1, 'opposing'), (1, 'outrageous'), (1, 'sea'), (1, 'question'), (1, 'slings'), (2, 'be'), (2, 'and'), (2, 'of'), (1, 'sleep'), (2,
'or')]
[(1, 'outrageous'), (1, 'question'), (1, 'sea'), (1, 'sleep'), (1, 'slings'), (2, 'be'), (2, 'and'), (2, 'of'), (1, 'suffer'), (2, 'or')]
[(1, 'question'), (1, 'sleep'), (1, 'sea'), (1, 'suffer'), (1, 'slings'), (2, 'be'), (2, 'and'), (2, 'of'), (1, 'take'), (2, 'or')]
[(1, 'sea'), (1, 'sleep'), (1, 'that'), (1, 'suffer'), (1, 'slings'), (2, 'be'), (2, 'and'), (2, 'of'), (1, 'take'), (2, 'or')]
[(1, 'sleep'), (1, 'slings'), (1, 'that'), (1, 'suffer'), (2, 'or'), (2, 'be'), (2, 'and'), (2, 'of'), (1, 'take'), (3, 'the')]
[(1, 'slings'), (1, 'suffer'), (1, 'that'), (1, 'take'), (1, 'them'), (2, 'be'), (2, 'and'), (2, 'of'), (2, 'or'), (3, 'the')]
[(1, 'suffer'), (1, 'take'), (1, 'that'), (1, 'tis'), (1, 'them'), (2, 'be'), (2, 'and'), (2, 'of'), (2, 'or'), (3, 'the')]
[(1, 'take'), (1, 'them'), (1, 'that'), (1, 'tis'), (3, 'the'), (2, 'be'), (2, 'and'), (2, 'of'), (2, 'or'), (6, 'to')]
[(1, 'that'), (1, 'them'), (2, 'and'), (1, 'tis'), (1, 'troubles'), (2, 'be'), (3, 'the'), (2, 'of'), (2, 'or'), (6, 'to')]
[(1, 'them'), (1, 'tis'), (2, 'and'), (1, 'whether'), (1, 'troubles'), (2, 'be'), (3, 'the'), (2, 'of'), (2, 'or'), (6, 'to')]
```

***provided that you are running locally**

We ... make this much more efficient...

Improved:

```
#!/usr/bin/env python2

from mrjob.job import MRJob
import heapq

TOPN = 10
class WordCount(MRJob):
    def mapper(self, _, line):
        for word in line.strip().split():
            word = filter(str.isalpha,word.lower())
            yield word,1

   def reducer(self, key, values):
       for v in sorted(heapq.nlargest(TO...es),reverse=True):
           yield key,v

if __name__=="__main__":
    WordCount.run()
```

Old:

```
#!/usr/bin/env python2

from mrjob.job im...ob
import heapq

TOPN
...dCount(MRJob):
    def mapper(self, _, line):
        for word in line.strip().split():
            word = filter(str.isalpha,word.lower())
            yield word,1

    def reducer_init(self):
        print("reducer_init")
        self.heap = []

    ...educer(self, key, values):
        ...heappush(self.heap,(sum(values),key))
        if ...lf.heap) > TOPN:
            ...appop(self.heap)
        print(se...)

    def reducer_final(se...
        for v in sorted(sel... reverse=True):
            yield v

if __name__=="__main__":
    WordCount.run()
```

Counters:
You can increment them anywhere; they are reported when the job ends.

Typical uses:

- Count improperly formatted input lines
- Count missing values
- Global statistics

Code:

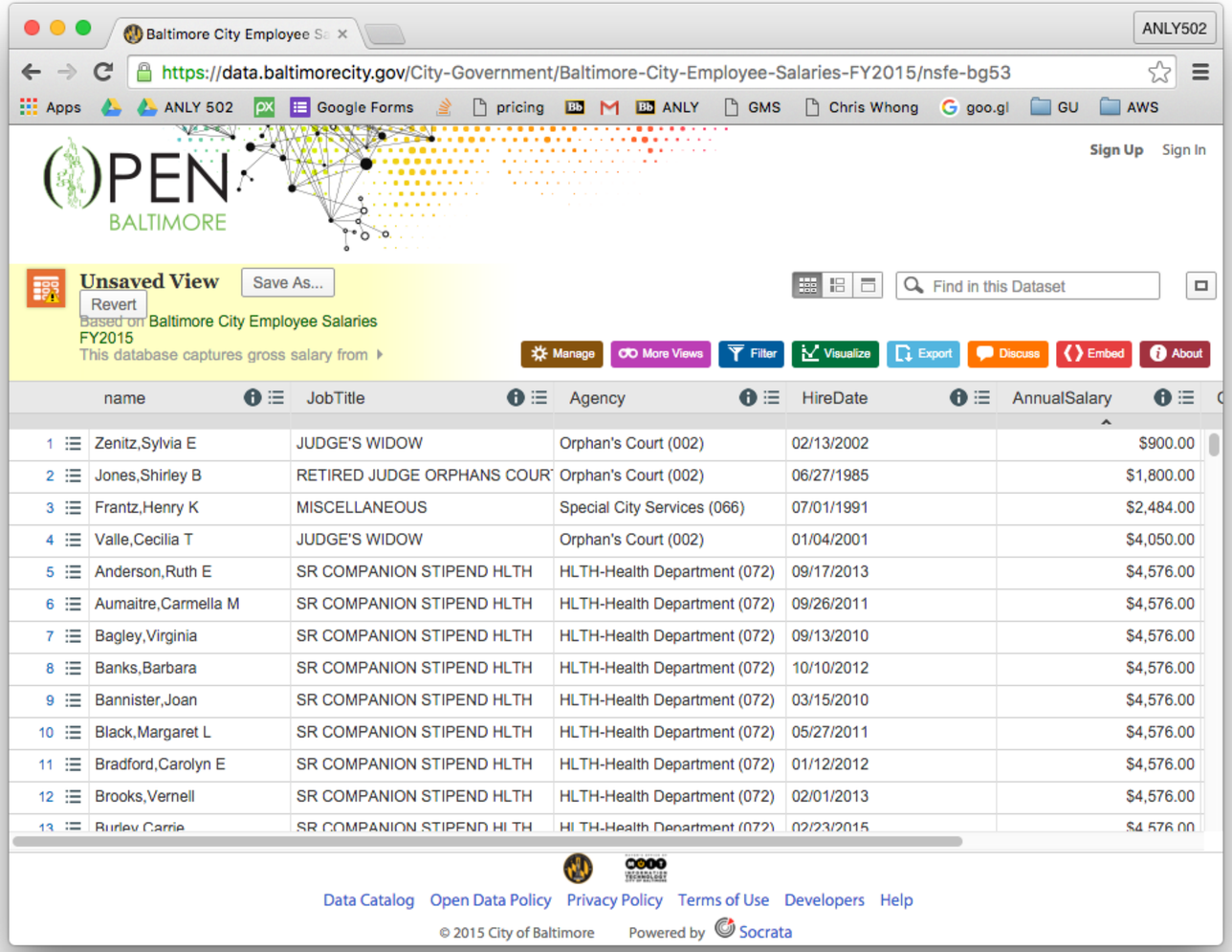
```
self.increment_counter( GROUP ,  COUNTER_NAME,  Amount)
```

—*e.g.*

```
self.increment_counter("warn", "missing gross pay", 1)
```

Example: Statistics for Baltimore City Employee Salaries

https://data.baltimorecity.gov/City-Government/Baltimore-City-Employee-Salaries-FY2015/nsfe-bg53



Examples courtesy of Donald Miner

Find the 5 highest values

Input format:

```
 Name,JobTitle,AgencyID,Agency,HireDate,AnnualSalary,GrossPay
"Aaron,Keontae E",AIDE BLUE CHIP,W02200,Youth Summer  ,06/10/2013,$11310.00,$873.63
"Aaron,Patricia G",Facilities/Office Services II,A03031,OED-Employment Dev ,10/24/1979,$53428.00,$52868.38
```

Code:

```
from mrjob.job import MRJob
import heapq, csv

cols = 'Name,JobTitle,AgencyID,Agency,HireDate,AnnualSalary,GrossPay'.split(",")

class salarymax(MRJob):
    def mapper_init(self):
        self.increment_counter("warn", "missing salary", 0)
        self.increment_counter("warn", "missing gross", 0)

    def mapper(self, _, line):
        if line[0]!=' ':
            row = dict(zip(cols, [ a.strip() for a in csv.reader([line]).next()]))
            try:
                yield "salary", (float(row["AnnualSalary"][1:]), line)
            except ValueError:
                self.increment_counter("warn", "missing salary", 1)
            try:
                yield "gross", (float(row["GrossPay"][1:]), line)
            except ValueError:
                self.increment_counter("warn", "missing gross", 1)

    def reducer(self, key, values):
        for p in heapq.nlargest(5,values):
            yield key, p

if __name__=="__main__":
    salarymax.run()
```

Find the 5 highest values — Output

Input format:

```
 Name,JobTitle,AgencyID,Agency,HireDate,AnnualSalary,GrossPay
"Aaron,Keontae E",AIDE BLUE CHIP,W02200,Youth Summer  ,06/10/2013,$11310.00,$873.63
"Aaron,Patricia G",Facilities/Office Services II,A03031,OED-Employment Dev ,10/24/1979,$53428.00,$52868.38
```

Output:

```
[Dance ~/gits/ANLY502/L02 22:06:54](master) $ python bces_demo1.py Baltimore_City_Employee_Salaries_FY2014.csv
...
Counters from step 1:
  warn:
    missing gross: 3223
    missing salary: 0
...
"gross"         [238772.04, "\"Bernstein,Gregg L\",STATE'S ATTORNEY,A29001,States Attorneys Office ,
01/03/2011,$238772.00,$238772.04"]
"gross"         [193653.69, "\"Batts,Anthony W\",EXECUTIVE LEVEL III,A99390,Police Department ,09/25/2012,$193800.00,$193653.69"]
"gross"         [188328.5, "\"Black,Harry E\",EXECUTIVE LEVEL III,A23001,FIN-Admin & Budgets ,01/30/2012,$190000.00,$188328.50"]
"gross"         [185741.81, "\"Charles,Ronnie E\",EXECUTIVE LEVEL III,A83001,HR-Human Resources ,07/05/2012,$200000.00,$185741.81"]
"gross"         [176141.33, "\"Nalewajko Jr,Stephen C\",POLICE LIEUTENANT EID,A99264,Police Department ,
08/21/1981,$95087.00,$176141.33"]

"salary"        [238772.0, "\"Bernstein,Gregg L\",STATE'S ATTORNEY,A29001,States Attorneys Office ,
01/03/2011,$238772.00,$238772.04"]
"salary"        [200000.0, "\"Charles,Ronnie E\",EXECUTIVE LEVEL III,A83001,HR-Human Resources ,07/05/2012,$200000.00,$185741.81"]
"salary"        [193800.0, "\"Batts,Anthony W\",EXECUTIVE LEVEL III,A99390,Police Department ,09/25/2012,$193800.00,$193653.69"]
"salary"        [190000.0, "\"Black,Harry E\",EXECUTIVE LEVEL III,A23001,FIN-Admin & Budgets ,01/30/2012,$190000.00,$188328.50"]
"salary"        [187200.0, "\"Swift,Michael\",CONTRACT SERV SPEC II,A02003,City Council ,05/19/2008,$187200.00,$3510.00"]
```

Find the average Annual Salary per job

Input format:

```
 Name,JobTitle,AgencyID,Agency,HireDate,AnnualSalary,GrossPay
"Aaron,Keontae E",AIDE BLUE CHIP,W02200,Youth Summer  ,06/10/2013,$11310.00,$873.63
"Aaron,Patricia G",Facilities/Office Services II,A03031,OED-Employment Dev ,10/24/1979,$53428.00,$52868.38
```

Mapper: <input line> → <Job Title, Annual Salary>

```
def mapper(self, _, line):
    row = dict(zip(cols, [ a.strip() for a in csv.reader([line]).next()]))
    self.increment_counter("depts",row["Agency"], 1)
    yield row["JobTitle"], int(float(row[“AnnualSalary"][1:]))
```

Reducer: <Job Title, [Annual Salaries] > → <Job Title, Average Salary>

- Can’t do this:

```
def reducer(self, key, values):
    yield key, sum(values)/len(values)
```

“values” is a generator, not a list!

- Do this instead:

```
def reducer(self, key, values):
    count = 0
    total = 0
    for value in values:
        count += 1
        total += value
    yield key, total / count
```

Show the average salaries:

The program:

```
# Compute the average salary per job title
# Then print the top 10 job titles.
#

from mrjob.job import MRJob
import csv
cols = 'Name,JobTitle,AgencyID,Agency,HireDate,AnnualSalary,GrossPay'.split(",")

class SalaryAvg(MRJob):
    def mapper(self, _, line):
        if line[0]==' ': return
        row = dict(zip(cols, [ a.strip() for a in csv.reader([line]).next()]))
        self.increment_counter("depts",row["Agency"], 1)
        yield row["JobTitle"], int(float(row["AnnualSalary"][1:]))

    def reducer(self, key, values):
        count = 0
        total = 0
        for value in values:
            count += 1
            total += value
        yield key, total / count

if __name__=="__main__":
    SalaryAvg.run()
```

Show the average salaries:

The output:

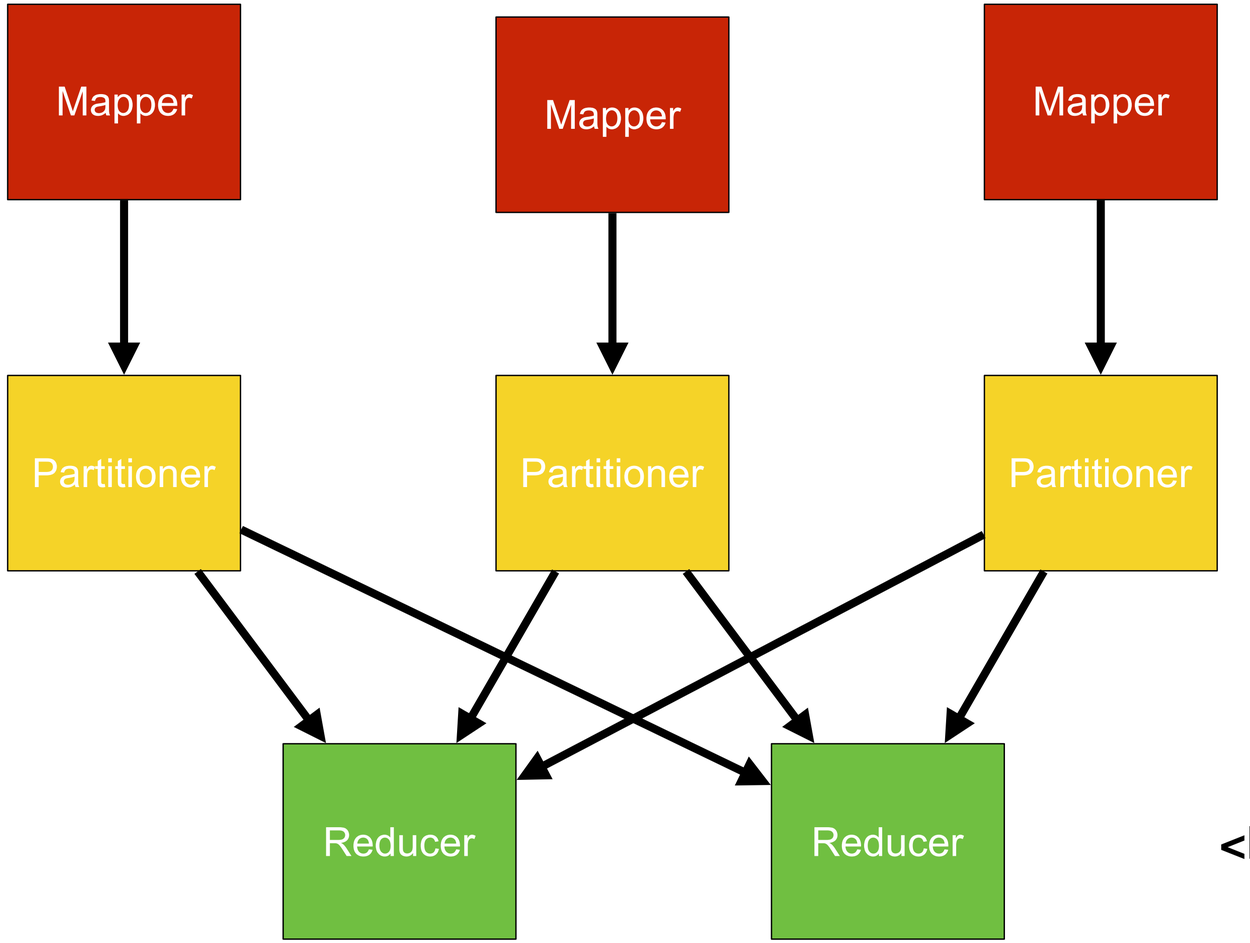
```
Counters from step 1:
  depts:
    COMP-Audits: 40
    COMP-Communication Ser: 30
    COMP-Comptroller's O: 12
    COMP-Real Estate: 8
    Circuit Court: 140
    City Council: 93
    Civil Rights & Wage Enfor: 12
    Council Services: 6
...

"911 LEAD OPERATOR"        47886
"911 OPERATOR SUPERVISOR"  54797
"911 OPERATOR"  42835
"ACCOUNT EXECUTIVE"        42960
"ACCOUNTANT I"  46316
"ACCOUNTANT II" 51253
"ACCOUNTANT SUPV"          62014
"ACCOUNTANT TRAINEE"       38572
"ACCOUNTING ASSISTANT II LIBRAR"     33215
"ACCOUNTING ASST I"        28393
"ACCOUNTING ASST II"       34376
"ACCOUNTING ASST III"      43248
"ACCOUNTING MANAGER"       72500
"ACCOUNTING OPERATIONS OFFICER"      77000
```

Hadoop Extra

Combiners, Filters, and more.

A closer look at map reduce

After the “map” runs, the Partitioner sends each <k,v> pair to the correct node.



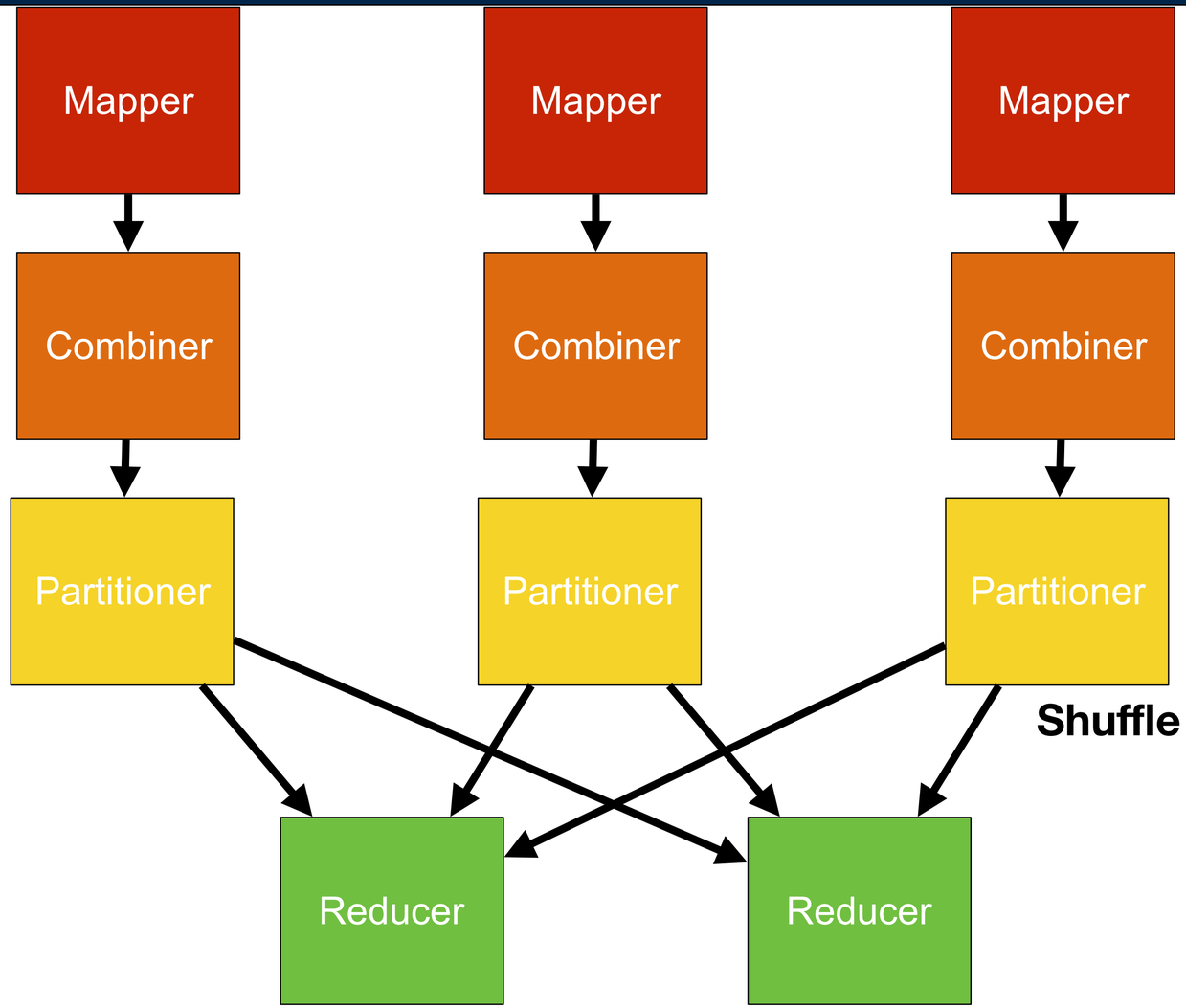
<k,v> → <k’, v’>

Shuffle

<k’,[v’,v’,v’...]> → <k”, v”>

Use a combiner for efficiency

For many operations, it's possible to perform an initial reduction at the node before data are sent across the network.



<k,v> → <k', v'>

<k',[v',v',v'...]> → <k", v">

<k",[v",v",v"...]> → <k"', v"'>

Combiner & Reducer: Same function signature!

The combiner can significantly reduce network traffic

Combiners perform an initial reduction

- Only works for some mathematical operations.
- The input & output <key,value> types must match the mapper.
- Combiners must be commutative and associative.
  - *e.g. f(a, b) == f(b, a) && f(a, f(b, c)) == f(f(a, b), c)*

This works for SUM:

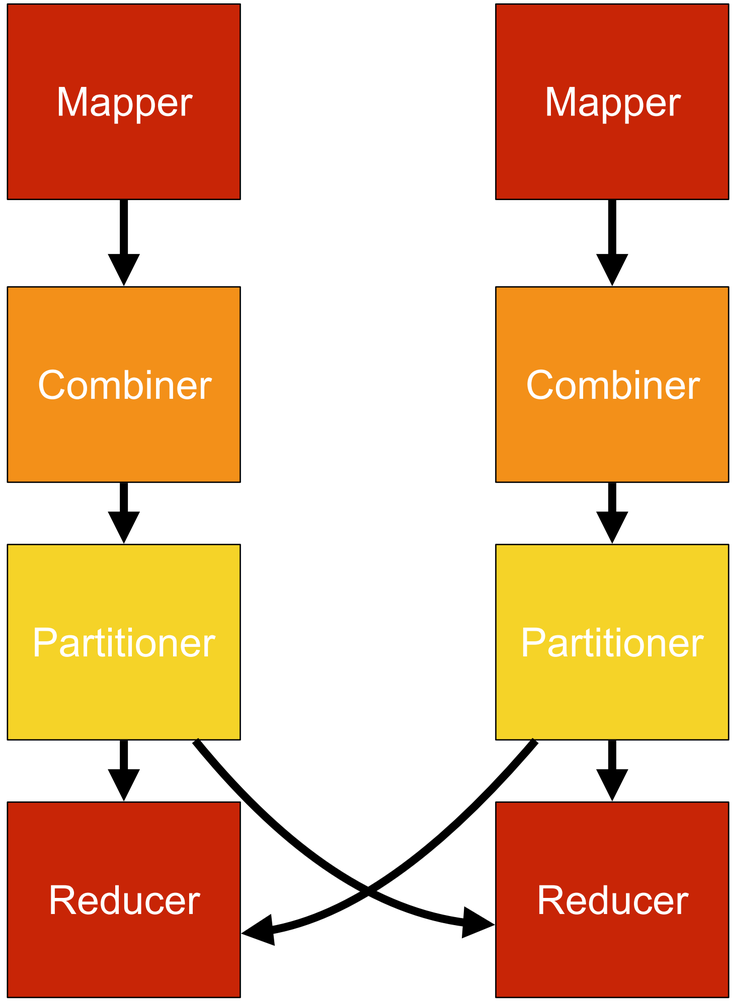
- (a+b+c+d) = (a+b) + (c+d) = (a+b+c) + d

Works for MIN/MAX/Top10

- max(a,b,c,d) = max(max(a,b) + max(c,d))

If you don't specify a combiner:

- The job runs the same
- The combiner becomes (k,v) => (k,v)



Combiners are easy to implement with mrjob.

In the mrjob example, they are the same:

```
class MRWordFreqCount(MRJob):

    def mapper(self, _, line):
        for word in WORD_RE.findall(line):
            yield word.lower(), 1

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

    def reducer(self, word, counts):
        yield word, sum(counts)
```

alternative, cool kids style:

```
class MRWordFreqCount(MRJob):

    def mapper(self, _, line):
        for word in WORD_RE.findall(line):
            yield word.lower(), 1

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

    reducer=combiner
```

Many functions can be made to work with combiners by refactoring the function.

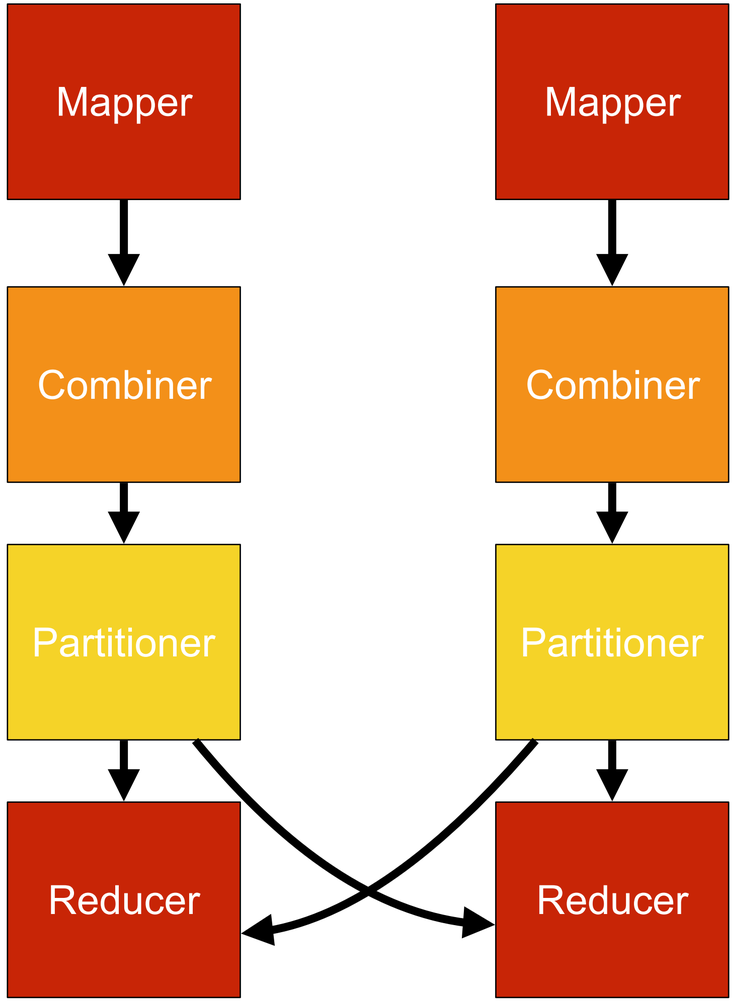
Making average work with combiner

The obvious “average” does not work for a combiner:

- (a+b+c+d)/4 ≠ ((a+b+c)/3 + d) / 2
- You can’t compute the average at the combiner and then again at the mapper.

Non-obvious pattern:

- <k,v> — make the v a tuple — (Σx, |x|)
- Mapper: outputs <category, value>
- Combiner: outputs <category, (Σ values, # values) >
- Reducer: outputs <category, average> (average = Σ(Σvalues) ÷ Σ(# values))



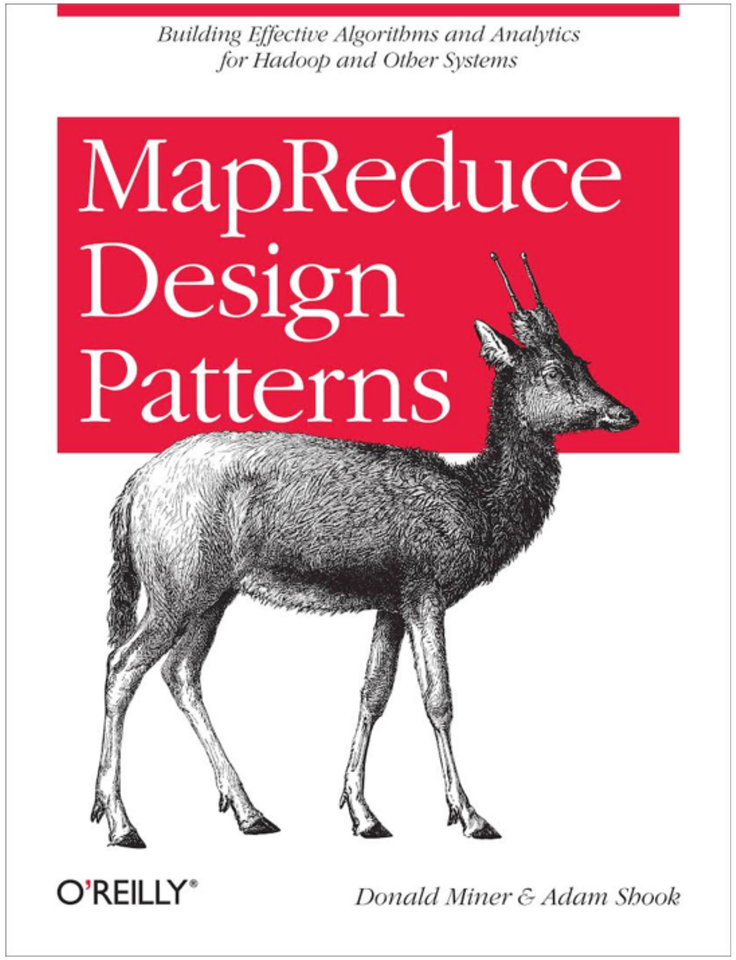
MapReduce Design Patterns

Be familiar with these patterns this week:

- Filtering in the mapper & reducer
- Finding min/max/average — at combiner & reducer
- Top-10 (Top-N)
- Counters (for bad input, etc.)
- Job chaining (review mrjob documentation)

Future design patterns:

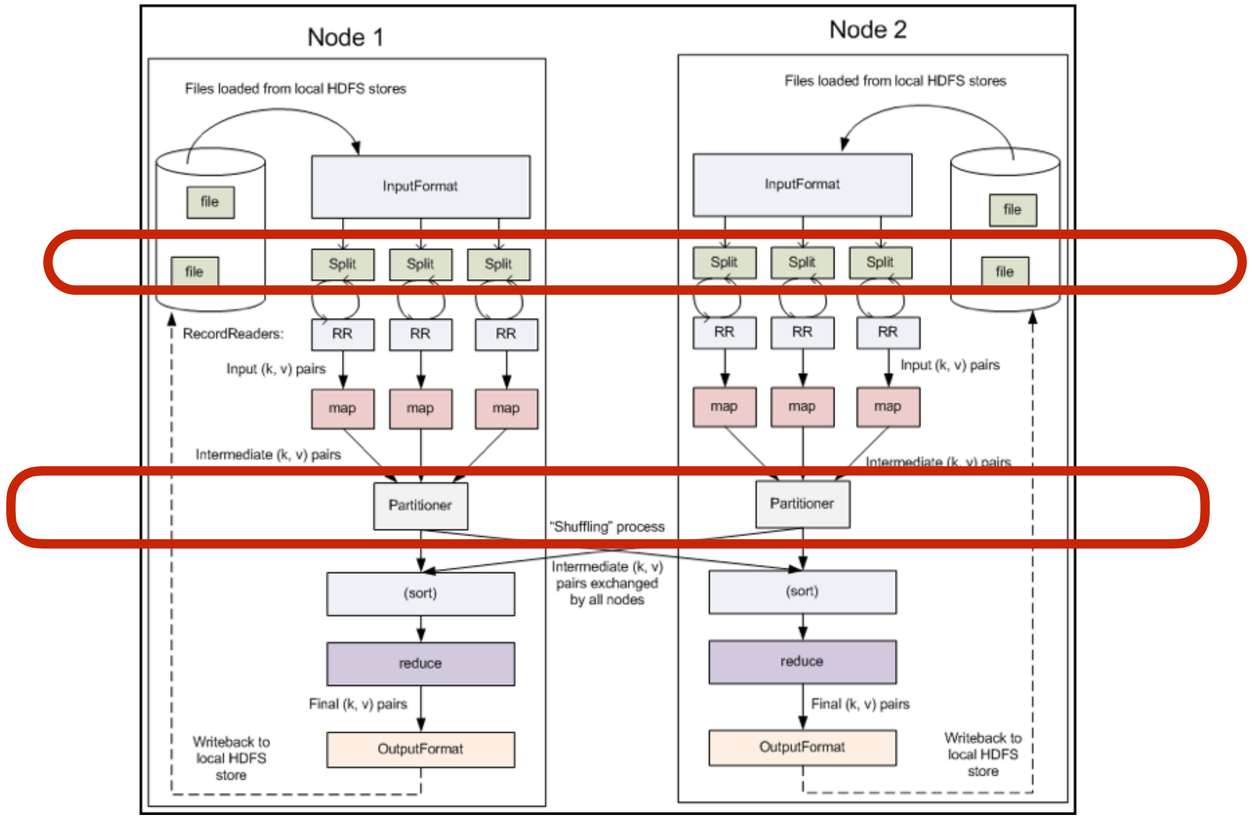
- Inner Joins (next week)
- Cartesian Product
- Inverted index (words, URLs, back pointers, etc)
- Bloom filter loaded from a distributed cache



Java-based MapReduce

(Hadoop also lets you define custom splitters and partitioners)

Useful for performance tuning.



(We won't be using custom splitting and partitioners in this course.)

Filtering — removing data from the stream.

Filtering — remove data from the pipeline early.

—*Removing data makes remaining work more efficient.*

mrjob supports “pre_filters,” but only for one-step jobs.

- MRJob’s filters are **shell commands.**
- Default filter is ‘cat’ — copies from stdin to stdout.
- Three filters.

```
MRJob.mapper_pre_filter()
MRJob.reducer_pre_filter()
MRJob.combiner_pre_filter()
```

- Filters run in a different process, before the Python program sees data.
- Filters don't run for "inline" runner.
- You may be better off filtering in your Python class

Example: remove “to” and “be” from wordcount program:

```
def mapper(self, _, line):
    for word in WORD_RE.findall(line):
        w=word.lower()
        if w not in ['to','be']:
            yield word.lower(), 1
```

Filter early:

- mapper first (if possible)
- else in the combiner
- else in the reducer

Controlling the number of Map and Reduce tasks

Map Tasks:

- You don't specify the number of Map Tasks.
  - —*You will have a minimum of one map task per input file / split / partition*
  - —*If you have too many input files, the Mapper tasks won't do enough.*

Reduce Tasks:

- Configurable and specified.
- Scale according to the amount of data.
  - —*If each reducer can handle 1 GB of data,*
    and you think the mappers will output 7 GB of data,
    then allocate 7 reducers.

HDFS File names

You can put metadata and other information in the filenames.

- Dates
- Sources

The file names are available to the **mapper()**, but not to the reducer()

- Why?

Multi-step jobs — when a single Map Reduce just won't do.

With traditional Hadoop MapReduce, you need to launch multiple jobs

— *In Java, you use org.apache.hadoop.mapred.jobcontrol.Job objects to represent each job.*

```
JobControl jbcntrl=new JobControl("jbcntrl");
jbcntrl.addJob(job1);
jbcntrl.addJob(job2);
job2.addDependingJob(job1);
jbcntrl.run();
```

- http://stackoverflow.com/questions/2499585/chaining-multiple-mapreduce-jobs-in-hadoop
- https://developer.yahoo.com/hadoop/tutorial/module4.html#chaining

— *You can use Oozie, the Hadoop Workflow Management System*

- XML-based configuration language
- https://issues.apache.org/jira/browse/HADOOP-5303

With mrjob, you can create a multi-step job directly in Python

Steps with mrjob

In your MRJob subclass:

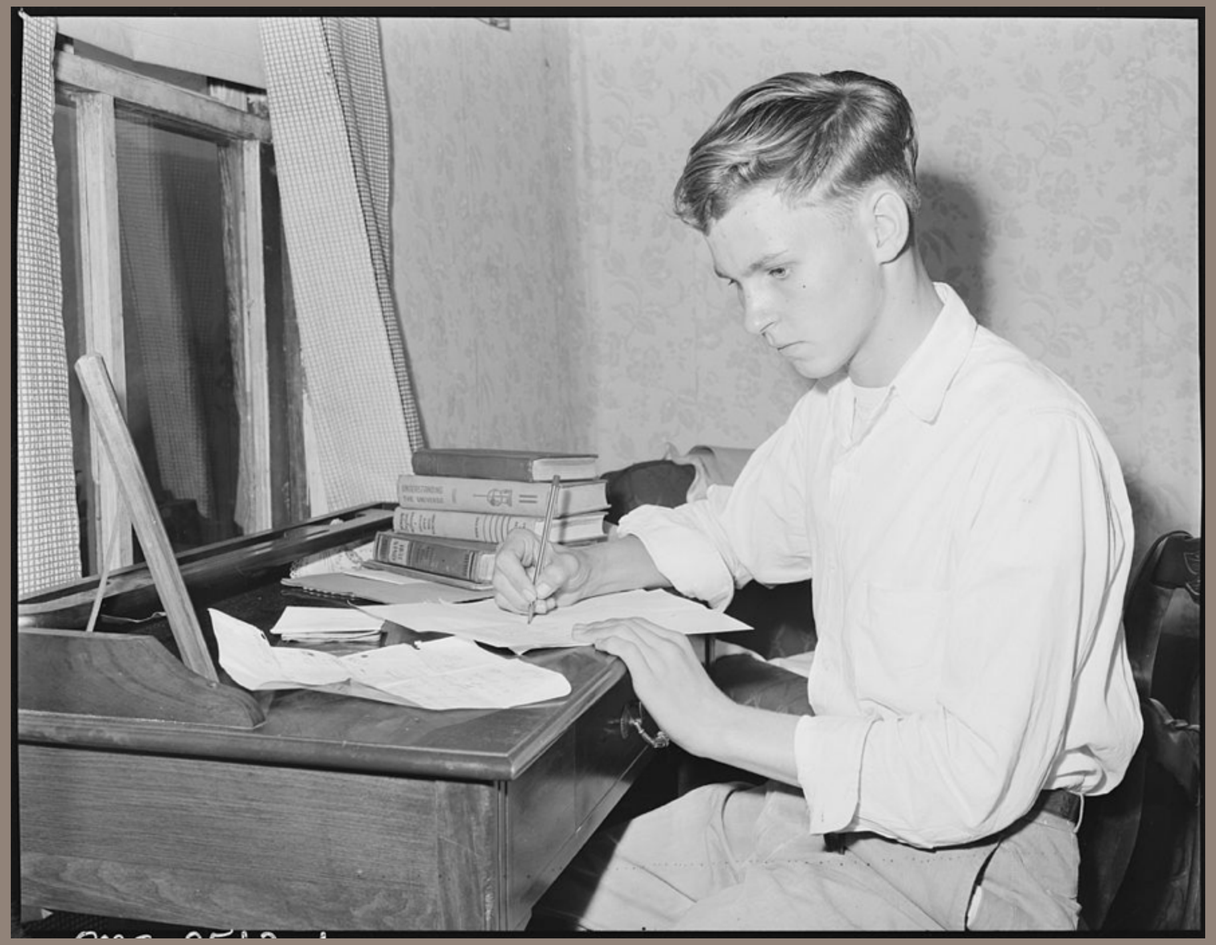
```
def steps(self):
    return [self.mr(mapper=self.transform_input,
                    reducer=self.consolidate_1),

            self.mr(reducer_init=self.log_mapper_init,
                    reducer=self.consolidate_2)]
```

MRJob automatically runs step2 after step1 is completed.

Note:

- Each step must completely finish before the next step starts

  - https://pythonhosted.org/mrjob/job.html#multi-step-jobs



http://bit.ly/louis_sergent_homework_1946

For Feb 1, 2016

Homework

Fri Jan 29 — PS01a Due

Mon Feb 1 — L03: Scaling from one computer to thousands

- Required Reading:
  - *DRAM Errors in the Wild: A Large-Scale Field Study, Schroeder et al, SIGMETRICS/Performance '09*
  - *Amazon EBS documentation, http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AmazonEBS.html*

Fri Feb 5 — PS02a Due Today!

Resources

Excellent blog post comparing different python frameworks for MR:

- http://blog.cloudera.com/blog/2013/01/a-guide-to-python-frameworks-for-hadoop/
- Slides: http://www.slideshare.net/slideshow/embed_code/key/9jAfDIRMoJiKPP
- Uses Google Books Ngram data as a demo, not wordcount!
  - See https://books.google.com/ngrams for more

Python iterators and generators:

- http://stackoverflow.com/questions/231767/what-does-the-yield-keyword-do-in-python
- https://www.jeffknupp.com/blog/2013/04/07/improve-your-python-yield-and-generators-explained/

What does the yield keyword do in Python?

4372

3096

What is the use of the `yield` keyword in Python? What does it do?

For example, I'm trying to understand this code[1]:

```
def node._get_child_candidates(self, distance, min_dist, max_dist):
    if self._leftchild and distance - max_dist < self._median:
        yield self._leftchild
    if self._rightchild and distance + max_dist >= self._median:
        yield self._rightchild
```

6821

+550

To understand what `yield` does, you must understand what *generators* are. And before generators come *iterables*.

Iterables

When you create a list, you can read its items one by one. Reading its items one by one is iteration:

```
>>> mylist = [1, 2, 3]
>>> for i in mylist:
...    print(i)
1
2
3
```

More Resources

Yahoo! Hadoop Tutorial

- https://developer.yahoo.com/hadoop/tutorial/

Apache Hadoop FAQ:

- https://wiki.apache.org/hadoop/FAQ

Hadoop-user mailing list archives:

- http://mail-archives.apache.org/mod_mbox/hadoop-user/

Frontiers in Massive Data Analysis (prepublication)

- http://www.nap.edu/catalog/18374/frontiers-in-massive-data-analysis (read online for free)

Revolution R Open

- http://www.revolutionanalytics.com/revolution-r-open

mrjob

- https://pythonhosted.org/mrjob/index.html
- http://stackoverflow.com/questions/tagged/mrjob
- https://github.com/Yelp/mrjob

Data Community DC
http://www.datacommunitydc.org/data-science-dc/