

Python MapReduce Programming with Pydoop

Simone Leo

Distributed Computing – CRS4

<http://www.crs4.it>

EuroPython 2011



Acknowledgments

- Part of the MapReduce tutorial is based upon: J. Zhao, J. Pjesivac-Grbovic, “MapReduce: The programming model and practice”, *SIGMETRICS'09 Tutorial*, 2009.

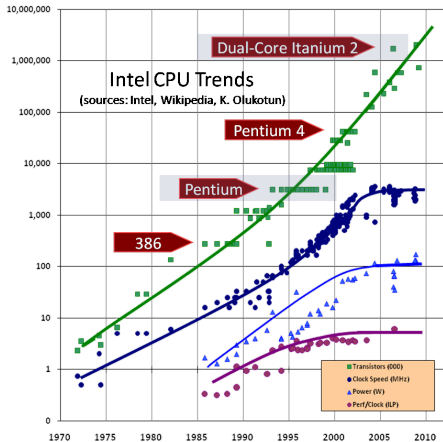
<http://research.google.com/pubs/pub36249.html>

- “The Free Lunch is Over” is a well-known article by Herb Sutter, available online at

<http://www.gotw.ca/publications/concurrency-ddj.htm>

- Pygments rules!

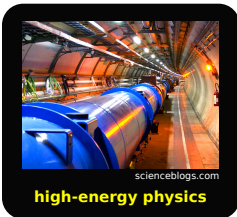
Intro: 1. The Free Lunch is Over



www.gotw.ca

- CPU clock speed reached saturation around 2004
 - Multi-core architectures
 - Everyone must go parallel
- Moore's law reinterpreted
 - number of *cores* per chip doubles every 2 y
 - clock speed remains fixed or decreases
 - must rethink the design of our software

Intro: 2. The Data Deluge



- **data-intensive applications**
 - 1 high-throughput sequencer: several TB/week
- Hadoop to the rescue!

Intro: 3. Python and Hadoop

- Hadoop: a DC framework for data-intensive applications
 - Open source Java implementation of Google's MapReduce and GFS
- Pydoop: API for writing Hadoop programs in Python
 - Architecture
 - Comparison with other solutions
 - Usage
 - Performance

Outline

- 1 MapReduce and Hadoop
 - The MapReduce Programming Model
 - Hadoop: Open Source MapReduce
- 2 Hadoop Crash Course
- 3 Pydoop: a Python MapReduce and HDFS API for Hadoop
 - Motivation
 - Architecture
 - Usage

Outline

- 1 **MapReduce and Hadoop**
 - The MapReduce Programming Model
 - Hadoop: Open Source MapReduce
- 2 Hadoop Crash Course
- 3 Pydoop: a Python MapReduce and HDFS API for Hadoop
 - Motivation
 - Architecture
 - Usage

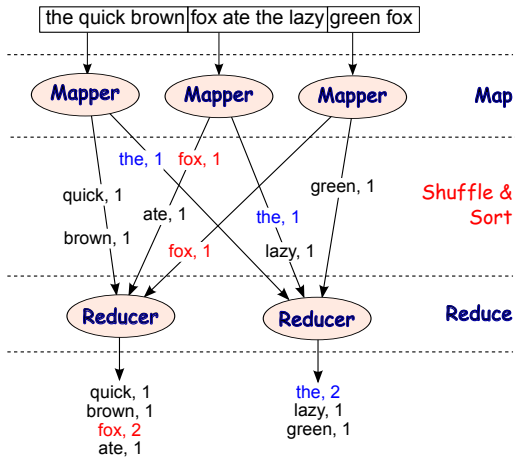
Outline

- 1 **MapReduce and Hadoop**
 - The MapReduce Programming Model
 - Hadoop: Open Source MapReduce
- 2 Hadoop Crash Course
- 3 Pydoop: a Python MapReduce and HDFS API for Hadoop
 - Motivation
 - Architecture
 - Usage

What is MapReduce?

- A programming model for large-scale distributed data processing
 - Inspired by `map` and `reduce` in functional programming
 - *Map*: map a set of input key/value pairs to a set of intermediate key/value pairs
 - *Reduce*: apply a function to all values associated to the same intermediate key; emit output key/value pairs
- An implementation of a system to execute such programs
 - Fault-tolerant (as long as the master stays alive)
 - Hides internals from users
 - Scales very well with dataset size

MapReduce's Hello World: Wordcount



Wordcount: Pseudocode

```
map(String key, String value):  
    // key: does not matter in this case  
    // value: a subset of input words  
    for each word w in value:  
        Emit(w, "1");  
  
reduce(String key, Iterator values):  
    // key: a word  
    // values: all values associated to key  
    int wordcount = 0;  
    for each v in values:  
        wordcount += ParseInt(v);  
    Emit(key, AsString(wordcount));
```



Mock Implementation – `mockmr.py`

```
from itertools import groupby
from operator import itemgetter

def _pick_last(it):
    for t in it:
        yield t[-1]

def mapreduce(data, mapf, redf):
    buf = []
    for line in data.splitlines():
        for ik, iv in mapf("foo", line):
            buf.append((ik, iv))
    buf.sort()
    for ik, values in groupby(buf, itemgetter(0)):
        for ok, ov in redf(ik, _pick_last(values)):
            print ok, ov
```

Mock Implementation – `mockwc.py`

```
from mockmr import mapreduce

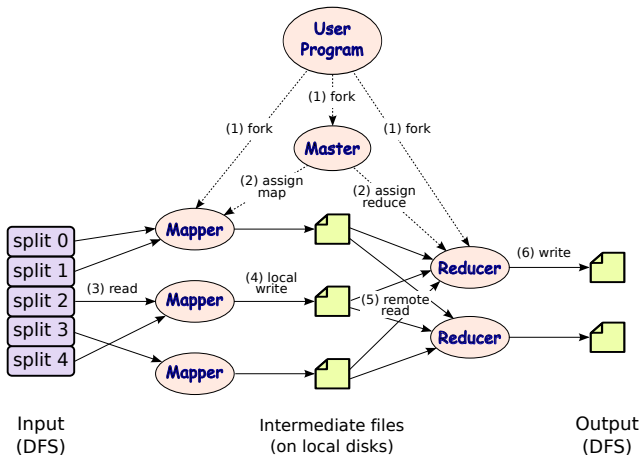
DATA = """the quick brown
fox ate the
lazy green fox
"""

def map_(k, v):
    for w in v.split():
        yield w, 1

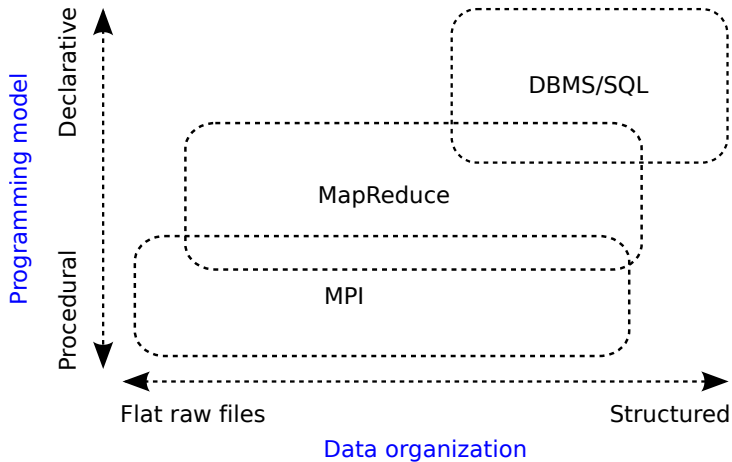
def reduce_(k, values):
    yield k, sum(v for v in values)

if __name__ == "__main__":
    mapreduce(DATA, map_, reduce_)
```

MapReduce: Execution Model



MapReduce vs Alternatives – 1



MapReduce vs Alternatives – 2

	MPI	MapReduce	DBMS/SQL
programming model	message passing	map/reduce	declarative
data organization	no assumption	files split into blocks	organized structures
data type	any	(k,v) string/protobuf	tables with rich types
execution model	independent nodes	map/shuffle/reduce	transaction
communication	high	low	high
granularity	fine	coarse	fine
usability	steep learning curve	simple concept	runtime: hard to debug
key selling point	run any application	huge datasets	interactive querying

- There is no one-size-fits-all solution
- Choose according to your problem's characteristics

adapted from Zhao et al., MapReduce: The programming model and practice, 2009 – see acknowledgments



MapReduce Implementations / Similar Frameworks

- Google MapReduce (C++, Java, Python)
 - Based on proprietary infrastructures (MapReduce, GFS, ...) and some open source libraries
- Hadoop (Java)
 - Open source, top-level Apache project
 - GFS → HDFS
 - Used by Yahoo, Facebook, eBay, Amazon, Twitter ...
- DryadLINQ (C# + LINQ)
 - Not MR, DAG model: vertices=programs, edges=channels
 - Proprietary (Microsoft); academic release available
- The “small ones”
 - Starfish (Ruby), Octopy (Python), Disco (Python + Erlang)



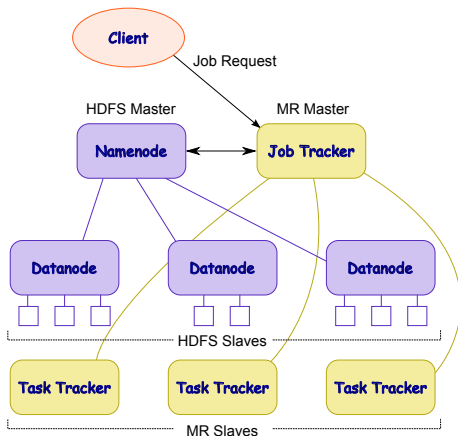
Outline

- 1 **MapReduce and Hadoop**
 - The MapReduce Programming Model
 - Hadoop: Open Source MapReduce
- 2 Hadoop Crash Course
- 3 Pydoop: a Python MapReduce and HDFS API for Hadoop
 - Motivation
 - Architecture
 - Usage

Hadoop: Overview

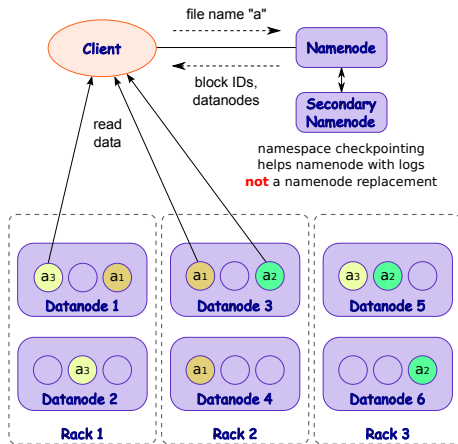
- Scalable
 - Thousands of nodes
 - Petabytes of data over 10M files
 - Single file: Gigabytes to Terabytes
- Economical
 - Open source
 - COTS Hardware (but master nodes should be reliable)
- Well-suited to bag-of-tasks applications (many bio apps)
 - Files are split into blocks and distributed across nodes
 - High-throughput access to huge datasets
 - WORM storage model

Hadoop: Architecture



- Client sends Job request to Job Tracker
- Job Tracker queries Namenode about physical data block locations
- Input stream is split among the desired number of map tasks
- Map tasks are scheduled closest to where data reside

Hadoop Distributed File System (HDFS)



- Each block is replicated n times (3 by default)
- One replica on the same rack, the others on different racks
- *You* have to provide network topology

Wordcount: (part of) Java Code

```

public static class TokenizerMapper
  extends Mapper<Object, Text, Text, IntWritable> {
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();
  public void map(Object key, Text value, Context context
    ) throws IOException, InterruptedException {
    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens()) {word.set(itr.nextToken());
      context.write(word, one);}
  }
}

public static class IntSumReducer
  extends Reducer<Text, IntWritable, Text, IntWritable> {
  private IntWritable result = new IntWritable();
  public void reduce(Text key, Iterable<IntWritable> values,
    Context context
    ) throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {sum += val.get();}
    result.set(sum);
    context.write(key, result);
  }
}

```

Other Optional MapReduce Components

- Combiner (local Reducer)
- RecordReader
 - Translates the byte-oriented view of input files into the record-oriented view required by the Mapper
 - Directly accesses HDFS files
 - Processing unit: InputSplit (filename, offset, length)
- Partitioner
 - Decides which Reducer receives which key
 - Typically uses a hash function of the key
- RecordWriter
 - Writes key/value pairs output by the Reducer
 - Directly accesses HDFS files

Outline

- 1 MapReduce and Hadoop
 - The MapReduce Programming Model
 - Hadoop: Open Source MapReduce
- 2 Hadoop Crash Course
- 3 Pydoop: a Python MapReduce and HDFS API for Hadoop
 - Motivation
 - Architecture
 - Usage

Hadoop on your Laptop in 10 Minutes

- Download from

```
www.apache.org/dyn/closer.cgi/hadoop/core
```

- Unpack to /opt, then set a few vars:

```
export HADOOP_HOME=/opt/hadoop-0.20.2
export PATH=$HADOOP_HOME/bin:${PATH}
```

- Setup passphraseless ssh:

```
ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
cat ~/.ssh/id_dsa.pub >>~/.ssh/authorized_keys
```

- in \$HADOOP_HOME/conf/hadoop-env.sh, set JAVA_HOME to the appropriate value for your machine



Additional Tweaking – Use as Non-Root

- **Assumption: user is in the user group**

```
# mkdir /var/tmp/hdfs /var/log/hadoop
# chown :users /var/tmp/hdfs /var/log/hadoop
# chmod 770 /var/tmp/hdfs /var/log/hadoop
```

- **Edit \$HADOOP_HOME/conf/hadoop-env.sh:**

```
export HADOOP_LOG_DIR=/var/log/hadoop
```

- **Edit \$HADOOP_HOME/conf/hdfs-site.xml:**

```
<property>
  <name>dfs.name.dir</name>
  <value>/var/tmp/hdfs/nn</value>
</property>
<property>
  <name>dfs.data.dir</name>
  <value>/var/tmp/hdfs/data</value>
</property>
```

Additional Tweaking – MapReduce

- **Edit** \$HADOOP_HOME/conf/mapred-site.xml:

```
<property>
  <name>mapred.system.dir</name>
  <value>/var/tmp/hdfs/system</value>
</property>
<property>
  <name>mapred.local.dir</name>
  <value>/var/tmp/hdfs/tmp</value>
</property>
<property>
  <name>mapred.tasktracker.map.tasks.maximum</name>
  <value>2</value>
</property>
<property>
  <name>mapred.tasktracker.reduce.tasks.maximum</name>
  <value>2</value>
</property>
<property>
  <name>mapred.child.java.opts</name>
  <value>-Xmx512m</value>
</property>
```



Start your Pseudo-Cluster

- **Namenode format is required only on first use**

```
hadoop namenode -format
```

```
start-all.sh
```

```
firefox http://localhost:50070 &
```

```
firefox http://localhost:50030 &
```

- **localhost:50070: HDFS web interface**
- **localhost:50030: MapReduce web interface**

Web Interface – HDFS

NameNode 'neuron.crs4.it:9000'

Started: Mon Jul 19 12:26:22 CEST 2010
Version: 0.20.2, r911707
Compiled: Fri Feb 19 08:07:34 UTC 2010 by chrisdo
Upgrades: There are no upgrades in progress.

[Browse the filesystem](#)
[Namenode Logs](#)

Cluster Summary

81 files and directories, 52 blocks = 133 total. Heap Size is 53.19 MB / 888.94 MB (5%)

Configured Capacity	:	19.69 GB
DFS Used	:	2.13 MB
Non DFS Used	:	12.87 GB
DFS Remaining	:	6.81 GB
DFS Used%	:	0.01 %
DFS Remaining%	:	34.61 %
Live Nodes	:	1
Dead Nodes	:	0



Web Interface – MapReduce

localhost Hadoop Map/Reduce Administration

State: RUNNING

Started: Mon Jul 19 12:26:22 CEST 2010

Version: 0.20.2, r911707

Compiled: Fri Feb 19 08:07:34 UTC 2010 by chrisdo

Identifier: 201007191226

Cluster Summary (Heap Size is 53.19 MB/888.94 MB)

Maps	Reduces	Total Submissions	Nodes	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node
0	0	0	1	2	2	4.00

Scheduling Information

Queue Name	Scheduling Information
default	N/A

Filter (Jobid, Priority, User, Name)

Example: 'user:smith 3200' will filter by 'smith' only in the user field and '3200' in all fields

Running Jobs



Run the Java Word Count Example

- Wait until HDFS is ready for work

```
hadoop dfsadmin -safemode wait
```

- Copy input data to HDFS

```
wget http://www.gutenberg.org/cache/epub/11/pg11.txt  
hadoop fs -put pg11.txt alice.txt
```

- Run Word Count

```
hadoop jar $HADOOP_HOME/*examples*.jar wordcount alice.txt output
```

- Copy output back to local fs

```
hadoop fs -get output{,}  
sort -rn -k2 output/part-r-00000 | head -n 3  
the 1664  
and 780  
to 773  
ls output/_logs/history  
localhost_1307814843760_job_201106111954_0001_conf.xml  
localhost_1307814843760_job_201106111954_0001_simleo_word+count
```

Cool! I Want to Develop my own MR Application!

- The easiest path for beginners is Hadoop Streaming
 - Java package included in Hadoop
 - Use any executable as the mapper or reducer
 - Read key-value pairs from standard input
 - Write them to standard output
 - Text protocol: records are serialized as `k\tv\n`
- Usage:

```
hadoop jar \  
  $HADOOP_HOME/contrib/streaming/*streaming*.jar \  
  -input myInputDirs \  
  -output myOutputDir \  
  -mapper my_mapper \  
  -reducer my_reducer \  
  -file my_mapper \  
  -file my_reducer \  
  -jobconf mapred.map.tasks=2 \  
  -jobconf mapred.reduce.tasks=2
```



WC with Streaming and Python Scripts – Mapper

```
#!/usr/bin/env python
import sys
for line in sys.stdin:
    for word in line.split():
        print "%s\t1" % word
```

WC with Streaming and Python Scripts – Reducer

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

def serialize(key, value):
    return "%s\t%d" % (key, value)

def deserialize(line):
    key, value = line.split("\t", 1)
    return key, int(value)

def main():
    prev_key, out_value = None, 0
    for line in sys.stdin:
        key, value = deserialize(line)
        if key != prev_key:
            if prev_key is not None:
                print serialize(prev_key, out_value)
                out_value = 0
            prev_key = key
            out_value += value
        print serialize(key, out_value)

if __name__ == "__main__": main()
```

Outline

- 1 MapReduce and Hadoop
 - The MapReduce Programming Model
 - Hadoop: Open Source MapReduce
- 2 Hadoop Crash Course
- 3 **Pydoop: a Python MapReduce and HDFS API for Hadoop**
 - Motivation
 - Architecture
 - Usage

Outline

- 1 MapReduce and Hadoop
 - The MapReduce Programming Model
 - Hadoop: Open Source MapReduce
- 2 Hadoop Crash Course
- 3 **Pydoop: a Python MapReduce and HDFS API for Hadoop**
 - **Motivation**
 - Architecture
 - Usage

MapReduce Development with Hadoop

- Java: native
- C/C++: APIs for both MR and HDFS are supported by Hadoop Pipes and included in the Hadoop distribution
- Python: several solutions, but do they meet all of the requirements of nontrivial apps?
 - Reuse existing modules, including C/C++ extensions
 - NumPy / SciPy for numerical computation
 - Specialized components (RecordReader/Writer, Partitioner)
 - HDFS access

Python MR: Hadoop-Integrated Solutions

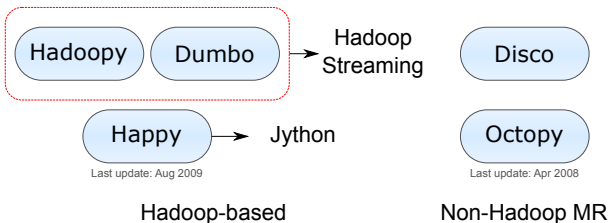
Hadoop Streaming

- awkward programming style
- can only write mapper and reducer scripts (no RecordReader, etc.)
- no HDFS
- can only process text data streams (lifted in 0.21+)

Jython

- incomplete standard library
- most third-party packages are only compatible with CPython
- cannot use C/C++ extensions
- typically one or more releases behind CPython

Python MR: Third Party Solutions



- Hadoop-based: same limitations as Streaming/Jython, except for ease of use
- Other implementations: not as mature/widespread

Python MR: Our Solution

Pydoop – <http://pydoop.sourceforge.net>

- Access to most MR components, including RecordReader, RecordWriter and Partitioner
- Get configuration, set counters and report status
- Programming model similar to the Java one: you define classes, the MapReduce framework instantiates them and calls their methods
- CPython → use any module
- HDFS API



Summary of Features

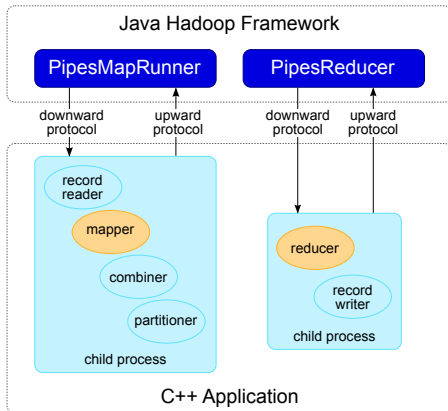
	Streaming	Jython	Pydoop
C/C++ Ext	Yes	No	Yes
Standard Lib	Full	Partial	Full
MR API	No*	Full	Partial
Java-like FW	No	Yes	Yes
HDFS	No	Yes	Yes

(*) you can only write the map and reduce parts as executable scripts.

Outline

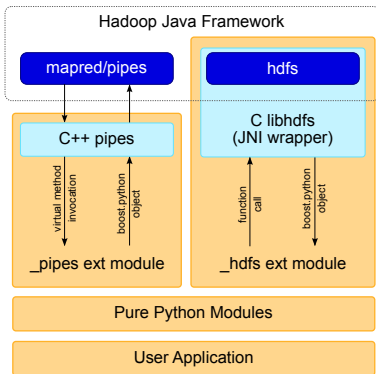
- 1 MapReduce and Hadoop
 - The MapReduce Programming Model
 - Hadoop: Open Source MapReduce
- 2 Hadoop Crash Course
- 3 **Pydoop: a Python MapReduce and HDFS API for Hadoop**
 - Motivation
 - **Architecture**
 - Usage

Hadoop Pipes



- App: separate process
- Communication with Java framework via persistent sockets
- The C++ app provides a factory used by the framework to create MR components

Integration of Pydoop with the C/C++ API



- Integration with Pipes (C++):
 - Method calls flow from the framework through the C++ and the Pydoop API, ultimately reaching user-defined methods
 - Results are wrapped by Boost and returned to the framework
- Integration with libhdfs (C):
 - Function calls initiated by Pydoop
 - Results wrapped and returned as Python objects to the app

Outline

- 1 MapReduce and Hadoop
 - The MapReduce Programming Model
 - Hadoop: Open Source MapReduce
- 2 Hadoop Crash Course
- 3 **Pydoop: a Python MapReduce and HDFS API for Hadoop**
 - Motivation
 - Architecture
 - **Usage**

Python Wordcount, Full Program Code

```
#!/usr/bin/env python
import pydoop.pipes as pp

class Mapper(pp.Mapper):
    def map(self, context):
        words = context.getInputValue().split()
        for w in words:
            context.emit(w, "1")



class Reducer(pp.Reducer):
    def reduce(self, context):
        s = 0
        while context.nextValue():
            s += int(context.getInputValue())
        context.emit(context.getInputKey(), str(s))

if __name__ == "__main__":
    pp.runTask(pp.Factory(Mapper, Reducer))
```

Status Reports and Counters

```
class Mapper(pp.Mapper):  
  
    def __init__(self, context):  
        super(Mapper, self).__init__(context)  
        context.setStatus("initializing")  
        self.inputWords = context.getCounter("WORDCOUNT", "INPUT_WORDS")  
  
    def map(self, context):  
        words = context.getInputValue().split()  
        for w in words:  
            context.emit(w, "1")  
        context.incrementCounter(self.inputWords, len(words))
```

Status Reports and Counters: Web UI

Task	Complete	Status	Start Time	Finish Time
task_201105051838_0001_m_000000	100.00% 	initializing	5-May-2011 18:53:35	5-May-2011 18:53:47
task_201105051838_0001_m_000001	100.00% 	initializing	5-May-2011 18:53:35	5-May-2011 18:53:47

	Counter	Map	Reduce	Total
WORDCOUNT	OUTPUT_WORDS	7,318	6,014	13,332
	INPUT_WORDS	29,459	0	29,459
Job Counters	Launched reduce tasks	0	0	2
	Launched map tasks	0	0	2
	Data-local map tasks	0	0	2
FileSystemCounters	FILE_BYTES_READ	0	84,210	84,210
	FILE_BYTES_WRITTEN	84,334	84,210	168,544
	Reduce input groups	0	6,014	6,014
	Combine output records	0	0	0
	Map input records	2	0	2
		0		

Optional Components: Record Reader

```

import struct, pydoop.hdfs as hdfs

class Reader(pp.RecordReader):
    def __init__(self, context):
        super(Reader, self).__init__(context)
        self.isplit = pp.InputSplit(context.getInputSplit())
        self.file = hdfs.open(self.isplit.filename)
        self.file.seek(self.isplit.offset)
        self.bytes_read = 0
        if self.isplit.offset > 0:
            discarded = self.file.readline() # read by prev. split reader
            self.bytes_read += len(discarded)
    def next(self): # return: (have_a_record, key, value)
        if self.bytes_read > self.isplit.length: # end of input split
            return (False, "", "")
        key = struct.pack(">q", self.isplit.offset+self.bytes_read)
        value = self.file.readline()
        if value == "": # end of file
            return (False, "", "")
        self.bytes_read += len(value)
        return (True, key, value)
    def getProgress(self):
        return min(float(self.bytes_read)/self.isplit.length, 1.0)

```



Optional Components: Record Writer, Partitioner

```
import pydoop.utils as pu

class Writer(pp.RecordWriter):
    def __init__(self, context):
        super(Writer, self).__init__(context)
        jc = context.getJobConf()
        pu.jc_configure_int(self, jc, "mapred.task.partition", "part")
        pu.jc_configure(self, jc, "mapred.work.output.dir", "outdir")
        pu.jc_configure(self, jc, "mapred.textoutputformat.separator",
                       "sep", "\t")
        self.outfn = "%s/part-%05d" % (self.outdir, self.part)
        self.file = hdfs.open(self.outfn, "w")
    def emit(self, key, value):
        self.file.write("%s%s%s\n" % (key, self.sep, value))

class Partitioner(pp.Partitioner):
    def partition(self, key, numofReduces):
        return (hash(key) & sys.maxint) % numofReduces
```

The HDFS Module

```
>>> import pydoop.hdfs as hdfs
>>> f = hdfs.open('alice.txt')
>>> f.fs.host
'localhost'
>>> f.fs.port
9000
>>> f.name
'hdfs://localhost:9000/user/simleo/alice.txt'
>>> print f.read(50)
Project Gutenberg's Alice's Adventures in Wonderla
>>> print f.readline()
nd, by Lewis Carroll
>>> f.close()
```

HDFS Usage by Block Size

```
import collections, pydoop.hdfs as hdfs

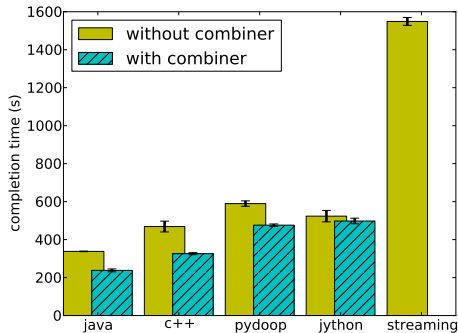
def treewalker(fs, root_info):
    yield root_info
    if root_info["kind"] == "directory":
        for info in fs.list_directory(root_info["name"]):
            for item in treewalker(fs, info):
                yield item

def usage_by_bs(fs, root):
    usage = collections.Counter()
    root_info = fs.get_path_info(root)
    for info in treewalker(fs, root_info):
        if info["kind"] == "file":
            usage[info["block_size"]] += info["size"]
    return usage

def main():
    fs = hdfs.hdfs("default", 0)
    root = "%s/%s" % (fs.working_directory(), "tree_test")
    for bs, tot_size in usage_by_bs(fs, root).iteritems():
        print "%.1f\t%d" % (bs/float(2**20), tot_size)
    fs.close()
```

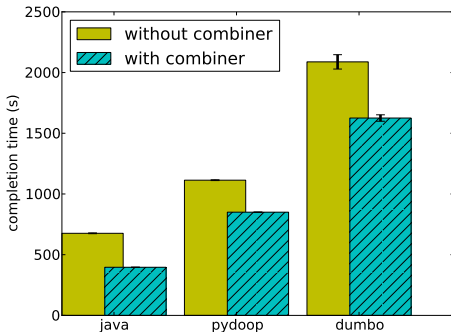


Comparison: vs Jython and Text-Only Streaming



- 48 nodes, 2 1.8 GHz dual core Opterons, 4 GB RAM
- App: Wordcount on 20 GB of random English text
 - Dataset: uniform sampling from a spell checker list
 - Java/C++ included for reference

Comparison: vs Dumbo (Binary Streaming)

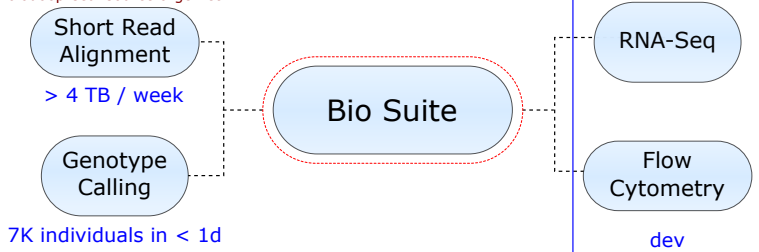


- 24 nodes, 2 1.8 GHz dual core Opterons, 4 GB RAM
- App: Wordcount on 20 GB of random English text
 - Dataset: uniform sampling from a spell checker list
 - Java included for reference

Pydoop at CRS4

- Core: computational biology applications for analyzing data generated by our Sequencing and Genotyping Platform
- the vast majority of the code is written in Python

biodoop-seal.sourceforge.net



Summary

- MapReduce is a big deal :)
 - Strengths: large datasets, scalability, ease of use
 - Weaknesses: overhead, lower raw performance
- MapReduce vs more traditional models
 - MR: low communication, coarse-grained, data-intensive
 - Threads/MPI: high communication, fine-grained, CPU-intensive
 - As with any set of tools, choose according to your problem
- Solid open source implementation available (Hadoop)
- Full-fledged Python/HDFS API available (Pydoop)

For Further Reading I



H. Sutter,

The Free Lunch is Over: a Fundamental Turn Toward Concurrency in Software

Dr. Dobbs Journal 30(3), 2005.



J. Dean and S. Ghemawat,

MapReduce: Simplified Data Processing on Large Clusters
in *OSDI 2004: Sixth Symposium on Operating System Design and Implementation*, 2004.





<http://hadoop.apache.org>



<http://pydoop.sourceforge.net>

For Further Reading II

-  S. Leo and G. Zanetti,
Pydoop: a Python MapReduce and HDFS API for Hadoop
In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC 2010)*, pages 819–825, 2010.
-  S. Leo, F. Santoni, and G. Zanetti,
Biodoop: Bioinformatics on Hadoop
In *The 38th International Conference on Parallel Processing Workshops (ICPPW 2009)*, pages 415–422, 2009.