Batch data processing in Python

Erik Bernhardsson
erikbern@spotify.com
Btw I’m Erik Bernhardsson

I’m at Spotify in NYC

- Focusing mostly on music discovery and large scale machine learning
- Previously managed the “Analytics team” in Stockholm
Background

Why did we build Luigi?

We crunch a lot of data

- Billions of log messages (several TBs) every day
- Usage and backend stats, debug information
- What we want to do
  - AB-testing
  - Music recommendations
  - Monthly/daily/hourly reporting
  - Business metric dashboards
- We experiment a lot – need quick development cycles
We like Hadoop

Our second cluster (in 2009):
Our fifth cluster

Long story short :)

Thursday, July 25, 13
Running one job is easy

But what about running 1000s of job every day?

- Lots of long-running processes with dependencies
- Need monitoring
- Handle failures
- Go from experimentation to production easily
But also non-Hadoop stuff

- Most things are Python Map/Reduce jobs
- Also Pig, Hive
- SCP files from one host to another
- Train a machine learning model
- Put data in Cassandra
How not to do workflows

In the pre-Luigi world
Example: Artist Toplist

“Streams” is a list of (username, track, artist, timestamp) tuples

Streams → Artist Aggregation → Top 10 → Database
Don’t do this at home

Pre-Luigi example of artist toplists
OK, so chain the tasks

```python
aggregate_artists(input="streams", output="artist_streams")

top_artists(input="artist_streams", output="top_artists")

export_to_database(input="top_artists",
                   host="localhost",
                   database="toplists",
                   username="toplists",
                   table="top10")
```
Cron nicer, yay!

```bash
# m h dom mon dow user  command
1 * * * * spotify-analytics /usr/bin/run_toplists
```

-uuu:---F1 crontab  All L3  (Fundamental)-------------------
Errors will occur

That’s OK, but don’t leave broken data somewhere
(btw, Luigi gives you atomic file operations locally and in HDFS)

```python
try:
    aggregate_artists(input="streams",
                       output="artist_streams")
except:
    remove("artist_streams")
```
Don’t run things twice

The second step fails, you fix it, then you want to resume

```python
if not exists("artist_streams"):
    try:
        aggregate_artists(input="streams",
                           output="artist_streams")
    except:
        remove("artist_streams")
```
Parametrize tasks

To use data flows as command line tools

```python
import datetime
import argparse

parser = argparse.ArgumentParser()
parser.add_argument('--date', required=True)
args = parser.parse_args()
date = datetime.date.strptime(args.date, '%Y-%m-%d').date()

output_path = date.strftime("artist_streams/%Y-%m-%d")
input_path = date.strftime("streams/%Y-%m-%d")

if not exists(output_path):
    try:
        aggregate_artists(input=input_path,
                          output=output_path)
    except:
        remove(output_path)
raise
```
Put tasks in loops

You want to run the dataflow for a set of similar inputs

today = date.today()
date = today - timedelta(days=7)

while date < today:
    output_path = date.strftime("artist_streams/%Y-%m-%d")
    input_path = date.strftime("streams/%Y-%m-%d")
    if not exists(output_path):
        try:
            aggregate_artists(input=input_path,
                              output=output_path)
        except:
            remove(output_path)
            raise
    date += timedelta(days=1)

...
Plumbing sucks
Plumbing sucks...

Graph algorithms rock!
Who’s the world’s second most famous plumber?

Hint: he wears green
Introducing Luigi

A Python framework for data flow definition and execution
Luigi is “kind of like Makefile” in Python

On steroids and PCP
... with a toolbox of mainly Hadoop related stuff

Main features

- Simple dependency definitions
- Emphasis on Hadoop/HDFS integration
- Atomic file operations
- Data flow visualization
- Command line integration
Luigi Task

class MyTask(Task):
    def output(self):
        pass

    def requires(self):
        pass

    def run(self):
        pass
Luigi - Aggregate Artists

```python
class AggregateArtists(Task):
    def output(self):
        return HdfsTarget("data/artist_streams.tsv")

    def requires(self):
        return Streams()

    def run(self):
        with self.input().open('r') as in_file:
            ... # read stuff from in_file

        with self.output().open('w') as out_file:
            ... # write stuff to out_file

if __name__ == "__main__":
    luigi.run()
```
Luigi - Aggregate Artists

Run on the command line:
$ python dataflow.py AggregateArtists

DEBUG: Checking if AggregateArtists() is complete
INFO: Scheduled AggregateArtists()
DEBUG: Checking if Streams() is complete
INFO: Done scheduling tasks
DEBUG: Asking scheduler for work...
DEBUG: Pending tasks: 1
INFO: [pid 74375] Running AggregateArtists()
INFO: [pid 74375] Done AggregateArtists()
DEBUG: Asking scheduler for work...
INFO: Done
INFO: There are no more tasks to run at this time
Completing the top list

Top 10 artists - Wrapped arbitrary Python code

```python
class Top10Artists(luigi.Task):
    def requires(self):
        return AggregateArtists()

def output(self):
    return LocalTarget("data/top_artists.tsv")

def run(self):
    top10 = []
    with self.input().open('r') as in_file:
        ...
        # get the top 10 artists from in_file

    with self.output().open('w') as out_file:
        ...
        # write top10 to out_file
```
Database support

Basic functionality for exporting to Postgres. Cassandra support is in the works

```python
import luigi.postgres

class ArtistToplistToDatabase(luigi.postgres.CopyToTable):
    host = "localhost"
    database = "toplists"
    user = "luigi"
    password = "abc123"  # ;)
    table = "top10"

    columns = [("artist", "TEXT"),
               ("streams", "INT")]

    def requires(self):
        return Top10Artists()
```
DEBUG: Checking if ArtistToplistToDatabase() is complete
INFO: Scheduled ArtistToplistToDatabase()
DEBUG: Checking if Top10Artists() is complete
INFO: Scheduled Top10Artists()
DEBUG: Checking if AggregateArtists() is complete
INFO: Scheduled AggregateArtists()
DEBUG: Checking if Streams() is complete
INFO: Done scheduling tasks
DEBUG: Asking scheduler for work...
DEBUG: Pending tasks: 3
INFO: [pid 74811] Running AggregateArtists()
INFO: [pid 74811] Done AggregateArtists()
DEBUG: Asking scheduler for work...
DEBUG: Pending tasks: 2
INFO: [pid 74811] Running Top10Artists()
INFO: [pid 74811] Done Top10Artists()
DEBUG: Asking scheduler for work...
DEBUG: Pending tasks: 1
INFO: [pid 74811] Running ArtistToplistToDatabase()
INFO: Done writing, importing at 2013-03-13 15:41:09.407138
INFO: [pid 74811] Done ArtistToplistToDatabase()
DEBUG: Asking scheduler for work...
INFO: Done
INFO: There are no more tasks to run at this time
The results

Imagine how cool this would be with real data...

```

<table>
<thead>
<tr>
<th>artist</th>
<th>streams</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bar Fighters</td>
<td>245</td>
</tr>
<tr>
<td>Piano Foo Music</td>
<td>213</td>
</tr>
<tr>
<td>Just another artist</td>
<td>200</td>
</tr>
<tr>
<td>Hey</td>
<td>147</td>
</tr>
<tr>
<td>Yo</td>
<td>131</td>
</tr>
<tr>
<td>MyArtist2</td>
<td>125</td>
</tr>
<tr>
<td>MyArtist3</td>
<td>121</td>
</tr>
<tr>
<td>Is</td>
<td>113</td>
</tr>
<tr>
<td>Hey</td>
<td>80</td>
</tr>
<tr>
<td>Sup?</td>
<td>78</td>
</tr>
</tbody>
</table>

(10 rows)
```
Task Parameters

Class variables with some magic

```python
class AggregateArtists(Task):
    date = DateParameter()

    def output(self):
        return HdfsTarget("/data/artist_streams/%s" % (self.date,))
```

Tasks have implicit `__init__`

```
In [21]: aggregate_task = AggregateArtists(date(2013, 3, 5))
agegregate_task.date

Out[21]: datetime.date(2013, 3, 5)
```

Generates command line interface with typing and documentation

```bash
$ python dataflow.py AggregateArtists --date 2013-03-05
```
Task Parameters

Combined usage example

class Streams(luigi.ExternalTask):
    date = DateParameter()

    def output(self):
        path = "/streams/%s" % (self.date,)
        return HdfsTarget(path)

class AggregateArtists(Task):
    date_interval = DateIntervalParameter()

    def requires(self):
        for date in self.date_interval:
            yield Streams(date)

    def output(self):
        path = "/artist_streams/%s" % (self.date_interval,)
        return HdfsTarget(path)
Task templates and targets

... how to run anything, really

Luigi comes with a toolbox of abstract *Tasks* for...

- Running Hadoop MapReduce utilizing Hadoop Streaming or custom jar-files
- Running Hive and (soon) Pig queries
- Inserting data sets into Postgres

Writing new ones are as easy as defining an interface and implementing `run()`
Hadoop MapReduce

Built-in Hadoop Streaming Python framework

Features

- Tiny interface – just implement mapper and reducer
- Fetches error logs from Hadoop cluster and displays them to the user
- Class instance variables can be referenced in MapReduce code, which makes it easy to supply extra data in dictionaries etc. for map side joins
- Easy to send along Python modules that might not be installed on the cluster
- Support for counters, secondary sort, combiners, distributed cache, etc.
- Runs on CPython so you can use your favorite libs (numpy, pandas etc.)
Hadoop MapReduce

Built-in Hadoop Streaming Python framework

class AggregateArtists(luigi.hadoop.JobTask):
    def requires(self):
        return Streams()

    def output(self):
        return HdfsTarget("data/artist_streams.tsv")

    def mapper(self, line):
        timestamp, artist, track = line.split('\t')
        yield artist, 1

    def reducer(self, artist, streams):
        yield artist, sum(streams)
More features
Luigi’s “visualiser”
Dive into any task
Multiple workers

Basic multi-processing

$ python dataflow.py --workers 3 AggregateArtists --date_interval 2013-W08
Error notifications

Great for automated execution

Luigi: AggregateArtists() FAILED

Runtime error:
Traceback (most recent call last):
  File "/Users/freider/Code/spotify/luigi/luigi/worker.py", line 147, in _run_task
task.run()
  File "/Users/freider/Code/spotify/luigi/luigi/hadoop.py", line 428, in run
    self.job_runner().run_job(self)
  File "/Users/freider/Code/spotify/luigi/luigi/hadoop.py", line 397, in run_job
    job. run_reducer(reduce_input, reduce_output)
  File "/Users/freider/Code/spotify/luigi7/luigi/hadoop.py", line 564, in _run_reducer
    self.writer(outputs, stdout)
  File "/Users/freider/Code/spotify/luigi/luigi/hadoop.py", line 467, in writer
    for output in outputs:
    File "/Users/freider/Code/spotify/luigi/luigi/hadoop.py", line 549, in _reduce_input
      for output in reducer(key, (v[1] for v in values)):
  File "dataflow.py", line 24, in reducer
    yield artist, sum(streems)
NameError: global name 'streems' is not defined
Process Synchronization

Prevents two identical tasks from running simultaneously
Process Synchronization

...what happens
Process Synchronization

...what happens

Luigi worker 1

Luigi worker 2

A

B

C

A

F

C
Process Synchronization

...what happens
Large data flows

(Screenshot from web interface)
Things Luigi is not
Luigi is not trying to replace mrjob

Yes, you can run Python Hadoop jobs in Luigi. But the main focus is workflow management.
Luigi does not give you scalability

You still need to figure out how each task runs
Luigi does not help you transform the data

Mapreduce/Pig/Hive/etc are wonderful tools for doing this and Luigi is more than happy to delegate it to them.
Although Oozie is kind of annoying

<table>
<thead>
<tr>
<th>Only Hadoop</th>
<th>Oozie</th>
<th>Yes!</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horrible XML</td>
<td></td>
<td>Yes!</td>
</tr>
<tr>
<td>Easy</td>
<td></td>
<td>Yes!</td>
</tr>
<tr>
<td>Fun &amp; powerful</td>
<td></td>
<td>Yes!</td>
</tr>
</tbody>
</table>
“Oozie example”
Luigi does not have 999 features

- Instead, focus on ridiculously little boiler plate code
- General so you can build whatever on top of it
- As well as rapid experimentation cycle
- Once things work, trivial to put in production
What we use Luigi for

- Hadoop Streaming
- Java Hadoop MapReduce
- Hive
- Pig
- Train machine learning models
- Import/export data to/from Postgres
- Insert data into Cassandra
- scp/rsync/ftp data files and reports
- Dump and load databases

Others using it with Scala MapReduce and MRJob as well
Be one of the cool kids!
Luigi is open source

https://github.com/spotify/luigi

- Originated at Spotify
- Mainly built by me and Elias Freider
- Based on many years of experience with data processing
- Open source since September 2012
Future plans!

- Pig
- EC2
- Scalding
- Cassandra
Thank you!

For more information feel free to reach out at http://github.com/spotify/luigi

Oh, and we’re hiring – http://spotify.com/jobs

Erik Bernhardsson
erikbern@spotify.com