

Getting Started with Hadoop with Amazon's Elastic MapReduce

Scott Hendrickson
scott@drskippy.net

<http://drskippy.net/projects/EMR-HadoopMeetup.pdf>

Boulder/Denver Hadoop Meetup

8 July 2010

Agenda

- 1 Amazon Web Services
- 2 Interlude: Solving problems with Map and Reduce
- 3 Running MapReduce on Amazon Elastic MapReduce
 - Example 1: Streaming Work Flow with AWS Management Console
 - Example 2 - Word count (Slightly more useful)
 - Example 3 - elastic-mapreduce command line tool
- 4 References and Notes

What is Amazon Web Services?

For first Hadoop project on AWS, use these services:

- Elastic Compute Cloud (EC2)
- Amazon Simple Storage Service (S3)
- Elastic MapReduce (EMR)

For future projects, AWS is much more:

- SimpleDB, Relational Database Services
- Simple Queue Service (SQS), Simple Notification Service (SNS)
- Alexa
- Mechanical Turk
- ...

Signing up for AWS

- 1 Create an AWS account - <http://aws.amazon.com/>
- 2 Sign up for EC2 cloud compute services - <http://aws.amazon.com/ec2/>
- 3 Set up Security Credentials (under menu **Account|Security Credentials**) - 3 kinds of credentials, you need to create an "Access Key"; use it to access S3 storage
- 4 Sign up for S3 storage services - <http://aws.amazon.com/s3/>
- 5 Sign up for EMR - <http://aws.amazon.com/elasticmapreduce/>

What are S3 buckets?

Streaming EMR projects use Simple Storage Service (S3) Buckets for data, code, logging and output.

Bucket “A bucket is a container for objects stored in Amazon S3. Every object is contained in a bucket.” Bucket names must be globally unique.

Object “Entities stored in Amazon S3. Objects consist of object data and metadata.” Metadata consists of key-value pairs. Object data is opaque.

Objects Keys “An object is uniquely identified within a bucket by a key (name) and a version ID.”

Accessing objects in S3 buckets

Want to:

- 1 Move data into and out of S3 buckets
- 2 Set access privileges

Tools:

- S3 console in your AWS control panel is adequate for managing S3 buckets and objects one at a time
- Other browser options: good for multiple file upload/download - Firefox S3 <https://addons.mozilla.org/en-US/firefox/addon/3247/> ; or minimal - S3 plug-in for Chrome <https://chrome.google.com/extensions/detail/appegcmaojledegaonmdaakfhjhchf>
- Programmatic options: Web Services (both SOAP-y and REST-ful): wget, curl, Python, Ruby, Java ...

S3 Bucket Example 1 - RESTful GET

Example - Image object

Bucket: bsi-test

Key: image.jpg

Object: JPEG structured data data from image.jpg

RESTful GET access, use URL:

<http://s3.amazonaws.com/bsi-test/image.jpg>

Example - Text file object

Bucket: bsi-test

Key: foobar

Object: text

RESTful GET access, use URL:

<http://s3.amazonaws.com/bsi-test/foobar>

S3 Bucket Example 2

Example - Python, Boto, Metadata

```
from boto.s3.connection import S3Connection
conn = S3Connection('key-id', 'secret-key')
bucket = conn.get_bucket('bsi-test')

k = bucket.get_key('image.jpg')
print "Value for key 'x-amz-meta-s3fox-modifiedtime' is:"
print k.get_metadata('s3fox-modifiedtime')
k.get_contents_to_filename('deleteme.jpg')

k = bucket.get_key('foobar')
print "Object value for key 'foobar' is:"
print k.get_contents_as_string()
print "Value for key 'x-amz-meta-example-key' is:"
print k.get_metadata('example-key')
```

S3 Bucket Example 2

Example - Python, Boto, Metadata - Output

```
scott@mowgli-ubuntu:~/Dropbox/hadoop$ ./botoExample.py
Value for key 'x-amz-meta-s3fox-modifiedtime' is:
1273869756000
Object value for key 'foobar' is:
This is a test of S3
Value for key 'x-amz-meta-example-key' is:
This is an example value.
```

What is Elastic Map Reduce?

Hadoop Hosted Hadoop framework running on EC2 and S3.

Job Flow Processing steps EMR “runs on a specified dataset using a set of Amazon EC2 instances.”

S3 Bucket(s) Input data, output, scripts, jars, logs.

Controlling Job Flows

Want to:

- 1 Configure jobs
- 2 Start jobs
- 3 Check status or stop jobs

Tools:

- AWS Management Console
`https://console.aws.amazon.com/elasticmapreduce/home`
- Command Line Tools
(requires Ruby [`sudo apt-get install ruby libopenssl-ruby`])
`http://developer.amazonwebservices.com/connect/entry.jspa?externalID=2264&categoryID=262`
- API calls defined by the service (REST-ful and SOAP-y)

EMR Example 1 - Running a simple Work Flow from the AWS Management Console

EMR Example 1

Hold up a minute...!

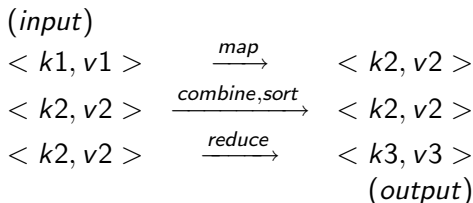
What problem are we solving?

Agenda

- 1 Amazon Web Services
- 2 Interlude: Solving problems with Map and Reduce
- 3 Running MapReduce on Amazon Elastic MapReduce
 - Example 1: Streaming Work Flow with AWS Management Console
 - Example 2 - Word count (Slightly more useful)
 - Example 3 - elastic-mapreduce command line tool
- 4 References and Notes

Central MapReduce Ideas

- Operate on key-value pairs
- Data scientist provides **map** and **reduce**



- (Optional: **Combine** provided in **map**, may significantly reduce bandwidth between workers)
- Efficient **Sort** provide by MapReduce library. Implies efficient **compare(k2_a, k2_b)**
- “Implicit” parallelization - splitting and distributing data, starting maps, reduces, collecting output

Key components of MapReduce framework

(wikipedia <http://en.wikipedia.org/wiki/MapReduce>)

The frozen part of the MapReduce framework is a large distributed sort.

The hot spots, which the application defines, are:

- 1 input reader
- 2 **Map** function
- 3 partition function
- 4 compare function
- 5 **Reduce** function
- 6 output writer

Google Tutorial View

- 1 MapReduce library shards the input files and starts up many copies on a cluster.
- 2 Master assigns work to workers. There are **map** and **reduce** tasks.
- 3 Workers assigned map tasks reads the contents input shard, parse key-value pairs and pass pairs to **map** function. Intermediate key-value pairs produced by the **map** function are buffered in memory.
- 4 Periodically, buffered pairs are written to disk, partitioned into regions. Locations of buffered pairs on the local disk are passed to the master.
- 5 When a **reduce** worker has read all intermediate data, it sorts by the intermediate keys. All occurrences a key are grouped together.
- 6 Reduce workers pass a key and the corresponding set of intermediate values to the **reduce** function.
- 7 Output of the **reduce** function is appended to a final output file for each reduce partition.

MapReduce Example 1 - Word Count - Data

(from Apache Hadoop tutorial)

Example: Word Count

file1:

```
Hello World Bye World
```

file2:

```
Hello Hadoop Goodbye Hadoop
```

MapReduce Example 1 - Word Count - Map

Example: Word Count

The first map emits:

```
< Hello, 1>  
< World, 1>  
< Bye, 1>  
< World, 1>
```

The second map emits:

```
< Hello, 1>  
< Hadoop, 1>  
< Goodbye, 1>  
< Hadoop, 1>
```

MapReduce Example 1 - Word Count - Sort and Combine

Example: Word Count

The output of the first map:

```
< Bye, 1>  
< Hello, 1>  
< World, 2>
```

The output of the second map:

```
< Goodbye, 1>  
< Hadoop, 2>  
< Hello, 1>
```

MapReduce Example 1 - Word Count - Sort and Reduce

Example: Word Count

The Reducer method sums up the values for each key.

The output of the job is:

```
< Bye, 1>
< Goodbye, 1>
< Hadoop, 2>
< Hello, 2>
< World, 2>
```

What problems is MapReduce good at solving?

Themes:

- Identify, transform, aggregate, filter, count, sort...
- Requirement of global knowledge of data is (a) “occasional” (vs. cost of map) (b) confined to ordinality
- Discovery tasks (vs. high repetition of similar transactional tasks, many reads)
- Unstructured data (vs. tabular, indexes!)
- Continuously updated data (indexing cost)
- Many, many, many machines (fault tolerance)

What problems is MapReduce good at solving?

Memes:

- MapReduce \Leftrightarrow SQL (read the comments too)
<http://www.data-miners.com/blog/2008/01/mapreduce-and-sql-aggregations.html>
- MapReduce vs. Message Passing Interface (MPI) “MPI is good for task parallelism and Hadoop is good for Data Parallelism.” finite differences, finite elements, particle-in-cell...
- MapReduce vs. column-oriented DBs tabular data, indexes (cantankerous old farts!) <http://databasecolumn.vertica.com/database-innovation/mapreduce-a-major-step-backwards/> and <http://databasecolumn.vertica.com/database-innovation/mapreduce-ii/>
- MapReduce vs. relational DBs http://scienceblogs.com/goodmath/2008/01/databases_are_hammers_mapreduc.php

Agenda

- 1 Amazon Web Services
- 2 Interlude: Solving problems with Map and Reduce
- 3 Running MapReduce on Amazon Elastic MapReduce
 - Example 1: Streaming Work Flow with AWS Management Console
 - Example 2 - Word count (Slightly more useful)
 - Example 3 - elastic-mapreduce command line tool
- 4 References and Notes

Example 1 - Add up integers

Data

3
4
-1
4
-3
1
1
...

Map

```
import sys
for line in sys.stdin:
    print '%s%s%d' % ("sum", '\t', int(line))
```

Example 1 - Add up integers

Reduce

```
import sys
sum_of_ints = 0
for line in sys.stdin:
    key, value = line.split('\t') # key is always the same
    try:
        sum_of_ints += int(value)
    except ValueError:
        pass
try:
    print "%s%s%d" % (key, '\t', sum_of_ints)
except NameError: # No items processed
    pass
```

Example 1 - Add up integers

Shell test

```
cat ./input/ints.txt | ./mapper.py > ./inter
cat ./input/ints1.txt | ./mapper.py >> ./inter
cat ./input/ints2.txt | ./mapper.py >> ./inter
cat ./input/ints3.txt | ./mapper.py >> ./inter
echo "Intermediate output:"
cat ./inter
cat ./inter | sort | \
    ./reducer.py > ./output/cmdLineOutput.txt
```

Example 1 - Add up integers

What was that comment earlier about an optional combiner?

Combiner in map

```
import sys
sum_of_ints = 0
for line in sys.stdin:
    sum_of_ints += int(line)
print '%s%s%d' % ("sum", '\t', sum_of_ints)
```

Example 1 - Add up integers

Combiner shell test

```
cat ./input/ints.txt | ./mapper_combine.py > ./inter
cat ./input/ints1.txt | ./mapper_combine.py >> ./inter
cat ./input/ints2.txt | ./mapper_combine.py >> ./inter
cat ./input/ints3.txt | ./mapper_combine.py >> ./inter
echo "Intermediate output:"
cat ./inter
cat ./inter | sort | \
    ./reducer.py > ./output/cmdLineCombOutput.txt
```

Example 1 - Add up integers - AWS Console

- 1 Upload **oneCount** directory with FFS3
- 2 Create a New Job Flow
Name: "oneCount"
Job Flow: Run own app
Job Type: Streaming
- 3 Input: bsi-test/oneCount/input
Output: bsi-test/oneCount/outputConsole (must not exist)
Mapper: bsi-test/oneCount/mapper.py
Reducer: bsi-test/oneCount/reducer.py
Extra Args: none

Example 1 - Add up integers - AWS Console

- ④ Instances: 4
 - Type: small
 - Keypair: No (Yes allows ssh to Hadoop master)
 - Log: yes
 - Log Location: bsi-test/oneCount/log
 - Hadoop Debug: no
- ⑤ No bootstrap actions
- ⑥ Start it, and wait. . .

Agenda

- 1 Amazon Web Services
- 2 Interlude: Solving problems with Map and Reduce
- 3 Running MapReduce on Amazon Elastic MapReduce
 - Example 1: Streaming Work Flow with AWS Management Console
 - Example 2 - Word count (Slightly more useful)
 - Example 3 - elastic-mapreduce command line tool
- 4 References and Notes

Example 2 - Word count

Map

```
def read_input(file):
    for line in file:
        yield line.split()

def main(separator='\t'):
    data = read_input(sys.stdin)
    for words in data:
        for word in words:
            lword = word.lower().strip(string.punctuation)
            print '%s%s%d' % (lword, separator, 1)
```

Example 2 - Word count

Reduce

```
def read_mapper_output(file, separator='\t'):
    for line in file:
        yield line.rstrip().split(separator, 1)

def main(separator='\t'):
    data = read_mapper_output(sys.stdin,
                              separator=separator)
    for current_word, group in groupby(data, itemgetter(0)):
        try:
            total_count = sum(int(count)
                               for current_word, count in group)
            print "%s%s%d" % (current_word,
                              separator, total_count)
        except ValueError:
            pass
```

Example 2 - Word count

Shell test

```
echo "foo foo quux labs foo bar quux" | ./mapper.py
echo "foo foo quux labs foo bar quux" | ./mapper.py \
    | sort | ./reducer.py
cat ./input/alice.txt | ./mapper.py \
    | sort | ./reducer.py > ./output/cmdLineOutput.txt
```

Example 2 - Word count - AWS Console

- 1 Upload **myWordCount** directory with FFS3
- 2 Create a New Job Flow
Name: "myWordCount"
Job Flow: Run own app
Job Type: Streaming
- 3 Input: bsi-test/myWordCount/input
Output: bsi-test/myWordCount/outputConsole (must not exist)
Mapper: bsi-test/myWordCount/mapper.py
Reducer: bsi-test/myWordCount/reducer.py
Extra Args: none

Example 2 - Word count - AWS Console

- ④ Instances: 4
 - Type: small
 - Keypair: No (Yes allows ssh to Hadoop master)
 - Log: yes
 - Log Location: bsi-test/myWordCount/log
 - Hadoop Debug: no
- ⑤ No bootstrap actions
- ⑥ Start it, and wait. . .

Agenda

- 1 Amazon Web Services
- 2 Interlude: Solving problems with Map and Reduce
- 3 Running MapReduce on Amazon Elastic MapReduce**
 - Example 1: Streaming Work Flow with AWS Management Console
 - Example 2 - Word count (Slightly more useful)
 - **Example 3 - elastic-mapreduce command line tool**
- 4 References and Notes

Example 3 - elastic-mapreduce command line tool

Word count (again, only better)

```
/usr/local/emr-ruby/elastic-mapreduce --create \  
  --stream \  
  --num-instances 2 \  
  --name from-elastic-mapreduce \  
  --input s3n://bsi-test/myWordCount/input \  
  --output s3n://bsi-test/myWordCount/outputRubyTool \  
  --mapper s3n://bsi-test/myWordCount/mapper.py \  
  --reducer s3n://bsi-test/myWordCount/reducer.py \  
  --log-uri s3n://bsi-test/myWordCount/log  
  
/usr/local/emr-ruby/elastic-mapreduce --list
```

Agenda

- 1 Amazon Web Services
- 2 Interlude: Solving problems with Map and Reduce
- 3 Running MapReduce on Amazon Elastic MapReduce
 - Example 1: Streaming Work Flow with AWS Management Console
 - Example 2 - Word count (Slightly more useful)
 - Example 3 - elastic-mapreduce command line tool
- 4 References and Notes

MapReduce Concepts Links

- Google MapReduce Tutorial: <http://code.google.com/edu/parallel/mapreduce-tutorial.html>
- Apache Hadoop tutorial: http://hadoop.apache.org/common/docs/current/mapred_tutorial.html
- Google Code University presentation on MapReduce: <http://code.google.com/edu/submissions/mapreduce/listing.html>
- MapReduce framework paper: <http://labs.google.com/papers/mapreduce-osdi04.pdf>

Amazon Web Services Links

- EMR Getting Started documentation:
`http://aws.amazon.com/documentation/elasticmapreduce/`
- Getting started with Amazon S3: `http://docs.amazonwebservices.com/AmazonS3/2006-03-01/gsg/`
- PIG on EMR: `http://s3.amazonaws.com/awsVideos/AmazonElasticMapReduce/ElasticMapReduce-PigTutorial.html`
- Boto Python library (multiple Amazon Services):
`http://code.google.com/p/boto/`

Machine Learning

- Linear speedup (with processor number) for “locally weighted linear regression (LWLR), k-means, logistic regression (LR), naive Bayes (NB), SVM, ICA, PCA, gaussian discriminant analysis (GDA), EM, and backpropagation (NN)”: <http://www.cs.stanford.edu/people/ang/papers/nips06-mapreducemulticore.pdf>
- Mahout framework: <http://mahout.apache.org/>

Examples Links

- Wordcount example/tutorial: http://www.michael-noll.com/wiki/Writing_An_Hadoop_MapReduce_Program_In_Python
- CouchDB and MapReduce (interesting examples of MR implementations for common problems)
http://wiki.apache.org/couchdb/View_Snippets
- This presentation:
<http://drskippy.net/projects/EMR-HadoopMeetup.pdf> or presentation source, example files etc.:
<http://drskippy.net/projects/EMR-HadoopMeetup.zip>