MIDS W205

Lab #	9	Lab Title	Apache Storm Introduction
Related Module(s)	9	Goal	Get you started on Storm
Last Updated	10/24/15	Expected duration	90 – 120 minutes

Introduction

A storm application is designed as a "topology" represented as a direct acyclic graph (DAG) with *spouts* and *bolts* acting as graph vertices. Edges on the graph are named streams and direct data from one node to another. Together, the topology acts as a data transformation pipeline. At a superficial level the general topology structure is similar to a MapReduce job, with the main difference being that data is processed in real-time as opposed to in individual batches. Additionally, Storm topologies run indefinitely until killed, while a MapReduce job must eventually end.

Storm can be used with many different languages. To avoid introducing a new language we will be using Python for our implementations of Sprouts and Bolts. To get an example up running quickly (although installation of streamparse is somewhat cumbersome) we will be using streamparse.

If you run `sparse quickstart`, streamparse will quick-start a local Storm + Python project using the streamparse framework. The basic example will implement a simple word count against a stream of words. Going into that directory and executing `sparse run` will spin up a local Apache Storm cluster and execute your topology of Python code against the local cluster.

Here are the steps we will cover in this lab:

- A video tutorial that helps you validate the Storm installation on a UCB AMI
- Install streamparse and its prerequisites so that we can get a Storm example up and running quickly.
- Review of the definition and implementation of a simple word count application that is using Storm.
- Run the sample wordcount Storm application
- Create and explore a new application simulating a tweet analysis application.

Instructions, Resources and Prerequisites

In the table below you will find references to resources related to programs and components used and mention in subsequent sections.

Resource	What

http://storm.apache.org/documentation.html	Apache Storm Documentation
http://streamparse.readthedocs.org/en/latest/quickstart.html	Introduction to stream parse topology definitions.
https://streamparse.readthedocs.org/en/latest/api.html	Streamparse Documentation
http://www.pixelmonkey.org/2014/05/04/streamparse	Short description of streamparse.
https://drive.google.com/file/d/0B6706xGNaPPycWpIVU9YWUtKelU/view?usp=sharing	Instruction video referred to in this lab.
https://pip.pypa.io/en/stable/	Pip documentation page.
http://docs.python-guide.org/en/latest/dev/virtualenvs/	Description of virtualenv

Step 1: Environment and tool setup

Watch the following video tutorial that walks you through setting up your Apache Storm Streamparse environment. This will allow you to run a word count example Storm application.

https://drive.google.com/file/d/0B6706xGNaPPycWpIVU9YWUtKelU/view?usp=sharing

Here is a summary if the commands you need to run, and that are used in the video. All of the below commands are covered by the video but are included here for your reference to make the installation easier to follow.

Check the version of the installed storm.

```
$storm version
/usr/bin/storm: line 2: /usr/hdp/2.2.4.2-2/etc/default/hadoop: No such file or directory
0.9.3.2.2.4.2-2
```

Check the version of python. For this lab you will need python version 2.7x

```
$python --version
Python 2.6.6
```

Install required version of python.

```
$sudo yum install python27-devel –y
```

You can see that the python in your execution PATH is still 2.6.X by trying version again.

```
/usr/bin/python --version
Python 2.6.6
```

Rename current version to reflect its version.

```
$mv /usr/bin/python /usr/bin/python266
```

Create a symbolic link from the file in the PATH to the version you want to execute.

```
$ln -s /usr/bin/python2.7 /usr/bin/python
```

Check that the link indeed refers to the intended version of python.

```
$usr/bin/python --version
Python 2.7.3
```

You can check that the shell picks up the version of python you intended.

```
$python --version
Python 2.7.3
```

Install ez_setup.

```
$sudo curl -o ez_setup.py https://bootstrap.pypa.io/ez_setup.py

$sudo python ez_setup.py
```

Use ez_install to install pip. Pip is a package manager for python software.

```
$sudo /usr/bin/easy_install-2.7 pip
```

We then use pip to install virtualenv. Virtualenv is a tool to create and manage dependencies for distinct python environments. Streamparse uses virtualenv to manage all dependencies for individual Python Storm projects.

```
$sudo pip install virtualenv
```

Streamparse requires the build tool lein to resolve dependencies. So we need to install lein.

Note: in the video the installer fails to save to `/usr/bin` so we have to move the `lein` file there. If the command succeeds you will not need to the `mv lein /usr/bin` command.

```
$get --directory-prefix=/usr/bin/
https://raw.githubusercontent.com/technomancy/leiningen/stable/bin/lein
```

If you check the permissions on the lein file you will see it is not executable. This means the shell and operating system will not allow you to run it as a command.

```
$ls -l /usr/bin/lein
-rw-r--r-- 1 root root 12713 Oct 25 17:01 /usr/bin/lein
```

Use the following chmod command to turn on the exactable permission for all users.

```
$ chmod a+x  /usr/bin/lein
```

Check that is looks like what you expect.

```
$ls -l /usr/bin/lein
-rwxr-xr-x 1 root root 12713 Oct 25 17:01 /usr/bin/lein
```

First time you run lein it will install itself.

```
$sudo  /usr/bin/lein

$lein version
WARNING: You're currently running as root; probably by accident.
Press control-C to abort or Enter to continue as root.
Set LEIN_ROOT to disable this warning.

Leiningen 2.5.3 on Java 1.7.0_79 Java HotSpot(TM) 64-Bit Server
```

Installation of streamparse

```
$pip install streamparse
```

Now you have streamparse installed. It will greatly simplify create of python storm projects and help you get a simple example up and running quickly.

After watching the video and understanding the structure of topology definitions and the actual spout and bolt. To create and run the wordcount example use the following commands.

```
$sparse quickstart wordcount
$cd wordcount
$sparse run
```

Step 2: Implementation of a Tweet Word count Topology

In this step, your task is to use the following topology to create *one spout*, *two bolts* that parse the tweets, and one *bolt* that counts the number of a given word in a tweet stream.

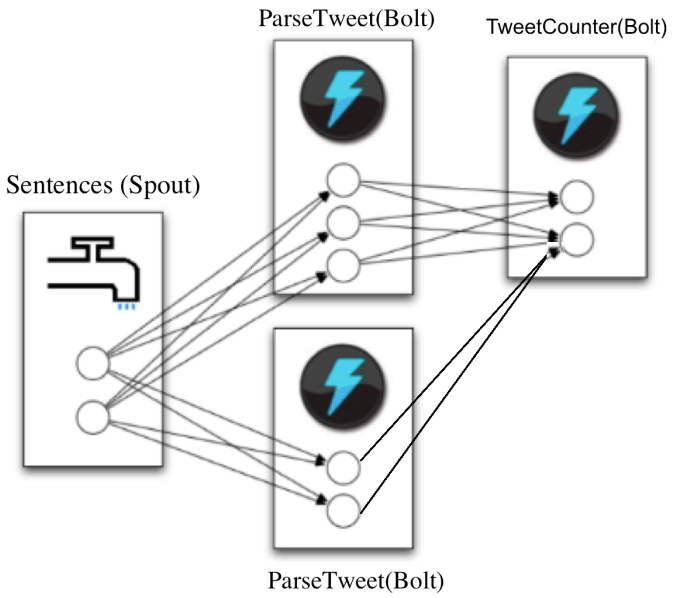

Figure 1: Task Topology

Create a project by running:

```
$sparse quickstart tweetcount
```

This command provides a basic `wordcount` topology example as seen in Step 1. You can modify this topology according to Figure 1 by modifying the file `wordcount.clj` in `tweetcount/topologies/`.

When constructing your topology it is important to remember that the topology is a function definition. This function must return an array with only two dictionaries and take one argument `options`. The first dictionary holds a named mapping of all the spouts that exist in the topology; the second holds a named mapping of all the bolts. Observe that the array is defined with square brackets, and each dictionary is a defined as a list with curly brackets. The options argument contains a mapping of topology settings.

You need to make sure that the topology and code are consistent with respect to the names for emitted tuples, dependencies etc. Be careful as each time to start a topology can take a long-time. If you have multiple bolts the structure is something along the following lines. Observe that when referring to the sprout or bolt, the first component is the file name the second the class name. The snippet below is only any outline and not a functional example.

```
(:use     [streamparse.specs])
  (:gen-class))

(defn tweetcount [options]
```

```
   [
    ;; spout configuration
      {"X-spout" (python-spout-spec
          options
          "spouts.<filename>.<classname>"
          [<emitted name>]
          ) }
    ;; bolt configuration 1
      {"Y-bolt"
             …
    ;; bolt configuration 2
      "Z-bolt"
            …
      }
  ]
)
```

Code base

Here are the code snippets that you can use for your spout and bolts. Remove all the `words.py` from your spouts directory and `wordcount.py` from your bolts folder in `tweetcount/src/`

Spout Name: Sentences(Spout)
Create a file called "`sentences.py`" using the following sample code. This is the spout code that will continuously generate tweet-like data.

```
from __future__ import absolute_import,print_function, unicode_literals
import itertools
from streamparse.spout import Spout
class Sentences(Spout):
    def initialize(self, stormconf, context):
        self.sentences = [
"She advised him to take a long holiday, so he i
mmediately quit work and took a trip around the world",
            "I was very glad to get a present from her",
            "He will be here in half an hour",
            "She saw him eating a sandwich",
        ]
        self.sentences = itertools.cycle(self.sentences)
    def next_tuple(self):
        sentence = next(self.sentences)
        self.emit([sentence])
    def ack(self, tup_id):
        pass  # if a tuple is processed properly, do nothing
    def fail(self, tup_id):
        pass  # if a tuple fails to process, do nothing
```

This Storm Spout has the following methods:

- initialize: "Initializes the storm spout and generates the data"

- next_tuple: “passes the events to bolts one by one”
- ack: “acknowledge the event delivery success”
- fail: “if event fails to deliver to bolts this method will be called”

Now you can put sentences.py into the /src/spouts/ directory.

Bolt 1 Name: ParseTweet(Bolt)
This bolt will capture the input coming from the Sentences spout, filter out specific formats and pass it to the next bolt of the topology, called tweetcount. Create a file called “parse.py” using the following sample code:

```
from __future__ import absolute_import, print_function,unicode_literals
import re
from streamparse.bolt import Bolt
def ascii_string(s):
  return all(ord(c) < 128 for c in s)
class ParseTweet(Bolt):
    def process(self, tup):
        tweet = tup.values[0]  # extract the tweet
        # Split the tweet into words
        words = tweet.split()
        valid_words = []
        for word in words:
            if word.startswith("#"): continue
            # Filter the user mentions
            if word.startswith("@"): continue
            # Filter out retweet tags
            if word.startswith("RT"): continue
            # Filter out the urls
            if word.startswith("http"): continue
            # Strip leading and lagging punctuations
            aword = word.strip("\"?><,'.:;)")
            # now check if the word contains only ascii
            if len(aword) > 0 and ascii_string(word):
                valid_words.append([aword])
        if not valid_words: return
        # Emit all the words
        self.emit_many(valid_words)
       # tuple acknowledgement is handled automatically.
```

ParseTweet(Bolt) will filter out input data that represents urls, user mentions, hash tags, etc. and will emit each word to the tweetcount bolt.

ParseTweet bolt methods:

- process: “actual programming logic is applied in this method”
- Tuple acknowledgement is handled automatically.

Bolt 2 Name: TweetCounter(Bolt)
This bolt will capture the input coming from the ParseTweet bolt, update the count of a given input word and print the result into log with the format “self.log('%s: %d' % (word, self.counts[word]))”. Create a file call “tweetcounter.py” using the following sample code.

```
from __future__ import absolute_import, print_function,
unicode_literals
from collections import Counter
from streamparse.bolt import Bolt

class TweetCounter(Bolt):
    def initialize(self, conf, ctx):
        self.counts = Counter()

    def process(self, tup):
        word = tup.values[0]
        # Increment the local count
        self.counts[word] += 1
        self.emit([word, self.counts[word]])
        # Log the count - just to see the topology running
        self.log('%s: %d' % (word, self.counts[word]))
```

TweetCounter bolt methods:

- initialize: “Initializes the bolt method with required variable initialization”
- process: “actual programming logic is applied in this method”
- Tuple acknowledgement is handled automatically.

Now you can put both parse.py and tweetcounter.py into your bolts/ directory.

Run the Storm Application

The final step is to run your application. You need to go inside tweetcount folder and run:

```
$cd wordcount
$sparse run
```

Submission

Submit a PDF that includes your topology file based on Figure 1 (wordcount.clj) and the screenshot of your running application that shows the stream of tweet counts on screen.