MIDS W205

|  |  |  |  |
| --- | --- | --- | --- |
| **Lab #** | 9 | **Lab Title** | Apache Storm Introduction |
| **Related Module(s)** | 9 | **Goal** | Get you started on Storm |
| **Last Updated** | 10/23/15 | **Expected duration** | 60 - 90 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.

## Here are the steps we will cover in this lab:

* A video tutorial that helps you install Storm on a UCB AMI and run a sample Storm application
* Implementation of a tweet word count application using Storm

# Instructions, Resources and Prerequisites

|  |  |
| --- | --- |
| **Resource** | **What** |
| http://storm.apache.org/documentation.html | Apache Storm Documentation |
| https://streamparse.readthedocs.org/en/latest/api.html | Stream Parse Documentation |

# Step 1: Setup Apache Storm Environment

Watch the following video tutorial that walks you through setting up your Apache Storm environment and running a word count example Storm application:

[https://drive.google.com/file/d/0B6706xGNaPPycWpIVU9YWUtKelU/view?usp=sharing](https://drive.google.com/file/d/0B6706xGNaPPycWpIVU9YWUtKelU/view?usp=sharing" \t "_blank)

# 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.

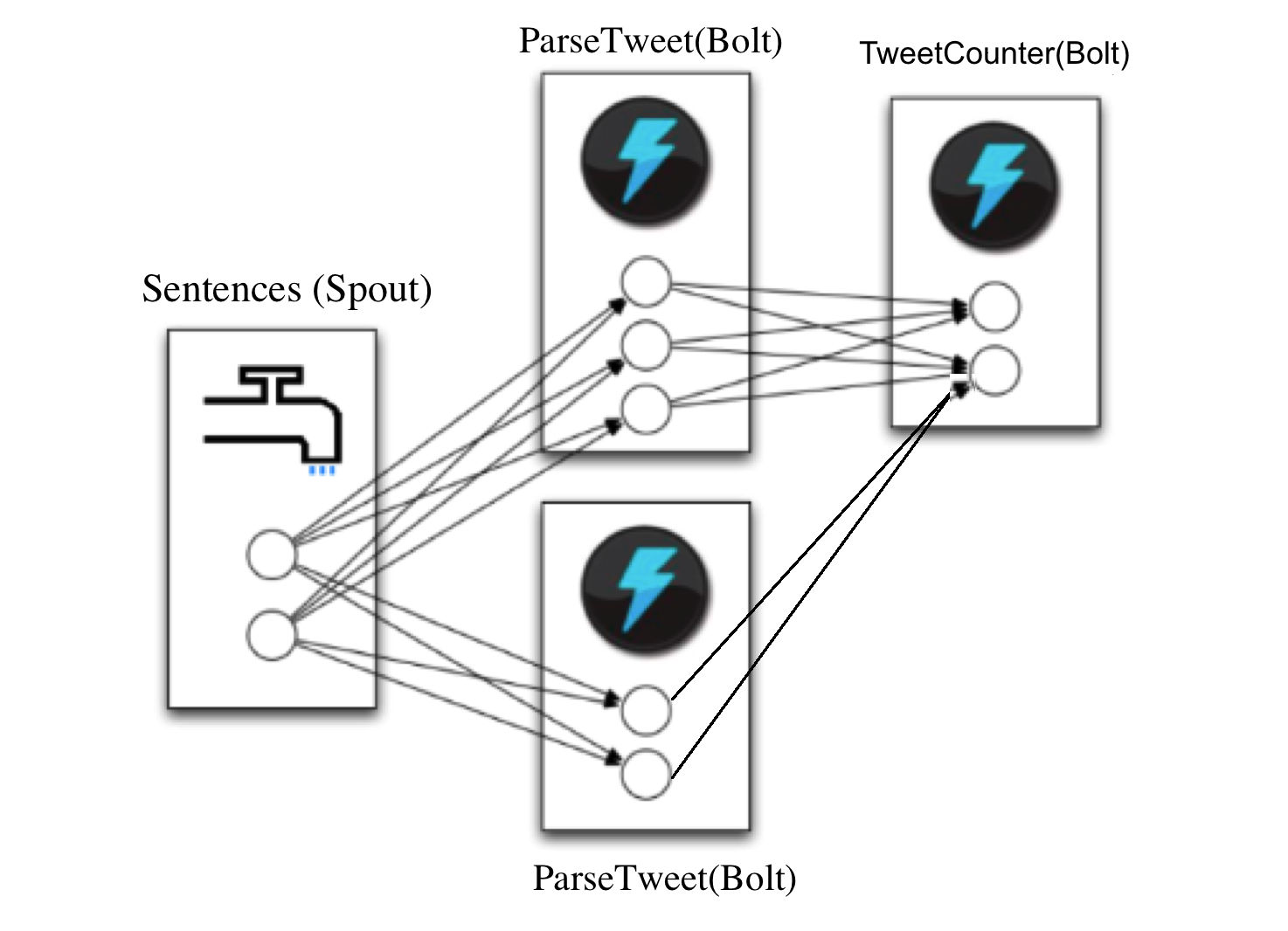


Figure 1: Task Topology

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. 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. The options argument contains a mapping of topology settings.

## 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, hashtags, 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:

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.