MIDS W205

Lab #	9 (ex 8)	Lab Title	Apache Storm -- Introduction
Related Module(s)	9	Goal	Get you started on Storm
Last Updated	9/27/15	Expected duration	60 minutes

Introduction

Storm application is designed as a "topology" in the shape of a direct acyclic graph (DAG) with spouts and bolts acting as the graph vertices. You must get familiar with a DAG first. 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 DAG must eventually end.

Here are the items we will cover in this lab:

1. Check version of Storm in your server (Server using the UCB AMI)
2. Get Github code for your lab
3. Code walkthrough of spout code and run
4. Code walkthrough of bolt and run

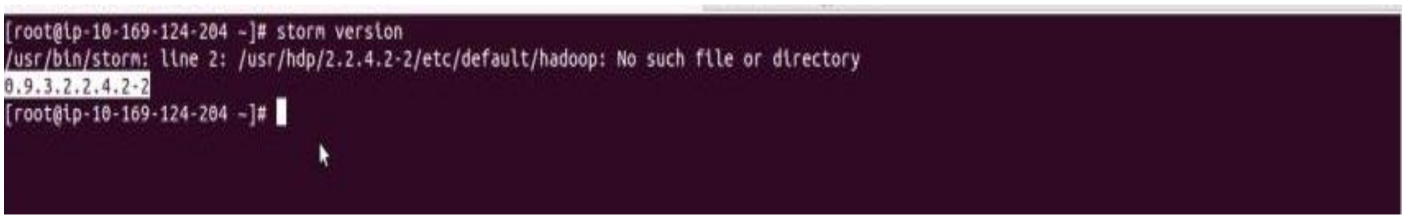
Instructions, resources and prerequisites

Resource	What
http://spark.apache.org/docs/latest/programming-guide.html	
https://spark.apache.org/docs/1.1.0/sql-programming-guide.html	
https://spark.apache.org/docs/latest/sql-programming-guide.html#running-the-spark-sql-cli	
https://spark.apache.org/docs/0.9.0/python-programming-guide.html	Python spark programming guide.
http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.SparkContext	Programming guide for the Spark Context object. Here you can find actions available on the Spark Contexts.

Step-1: Check version of Storm:

```
> storm version
0.9.3.2.2.4.2-2
```

Please check same in below screenshot



```
[root@ip-10-169-124-204 ~]# 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
[root@ip-10-169-124-204 ~]#
```

Step-2: Get Github code for your lab:

The location of the code is:
https://github.com/UC-Berkeley-I-School/data-science-w205/tree/master/exercise_2/wordcount

Step-3: Code walkthrough of spout code and run:

Spout Name: **Sentences** (Spout)

Create a file called "**Sentences.py**" with below code
This is the spout code where it will generate the data for wordcount sequentially.

```
      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 immediately 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 have the methods as below:

- **initialize** : “Initialize the storm spout and generates the data”
- **next_tuple**: “pass the events to bolts one by one”
- **ack** : “acknowledge the event delivery success”
- **fail** : “if event gets failed to deliver to bolts fail method will be called ”

You can refer to the code which you got from github as well. The goal is to understand the functions of each line of this code and real time data processing concepts at physical code level.

KARTHIK:PLEASE ADD YOUR SCREEN SHOTS and notes.

Step-4: Code walkthrough of bolt and run:

Bolt Name: **ParseTweet(**Bolt**)**
Create a file called “**parse.py**” with below code:

This bolt code will capture the input from **Sentences** spout and parse the given input data into different formats and pass to the next bolt called **wordcount** bolt.

```
      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.
```

above **ParseTweet(**Bolt**)** will filter the input data into urls,user mentions,hashtags and etc formats. and emits each word to **wordcount bolt and** tuple acknowledgement is handled automatically.

BOLT-2:

Bolt Name: **WordCounter**(Bolt)
Create a file call "**wordcounter.py**" with below code
This bolt code will capture the input from **ParseTweet** bolt and counts the given input word and prints the results into log with this format "**self.log('%s: %d' % (word, self.counts[word]))**"

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

      class WordCounter(Bolt):
          def initialize(self, conf, ctx):
              self.counts = Counter()
              self.redis = StrictRedis()

          def process(self, tup):
              word = tup.values[0]
              # Increment the word count in redis
              self.redis.zincrby("tweetwordcount", word)
              # 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]))
```

WordCount bolt methods:
initialize: "Initialize the bolt method with required variable initialization "
process: "actual programing logic will be applied in this method"
tuple acknowledgement is handled automatically.

How to Run Application:

KARTHIK:PLEASE ADD YOUR SCREEN SHOTS and notes.

Questions:
1. When will you use Apache Storm?
2. What is a bolt?
3. What is a spout?