LARGE SCALE DATA PROCESSING PROJECT

(CSE 3025)

DR R. BHARGAVI

WINTER SEMESTER 2016-2017

Real-Time Analytics with Apache Storm using data provided by Twitter

KASHISH MIGLANI (15BCE1003)
OSHO AGYEYA (15BCE1326)

ABSTRACT

PLATFORM: APACHE STORM (VERSION-1.0.3)

This project is aimed at the following targets:

- To learn Apache Storm, fundamental concepts involved in real time analytics as well as basic Storm terminology.
- To understand and implement Storm-Twitter connectivity.
- To use the above connectivity in order to obtain top-n trending worldwide hash tags.
- To retrieve tweets related to the above top-n trending worldwide hash tags.

DESIGN

Storm Overview:

- **Storm** is a platform for easily analyzing realtime streams of data as they arrive , so we can react to data as it happens.

- It is a free and open source distributed realtime computation system.

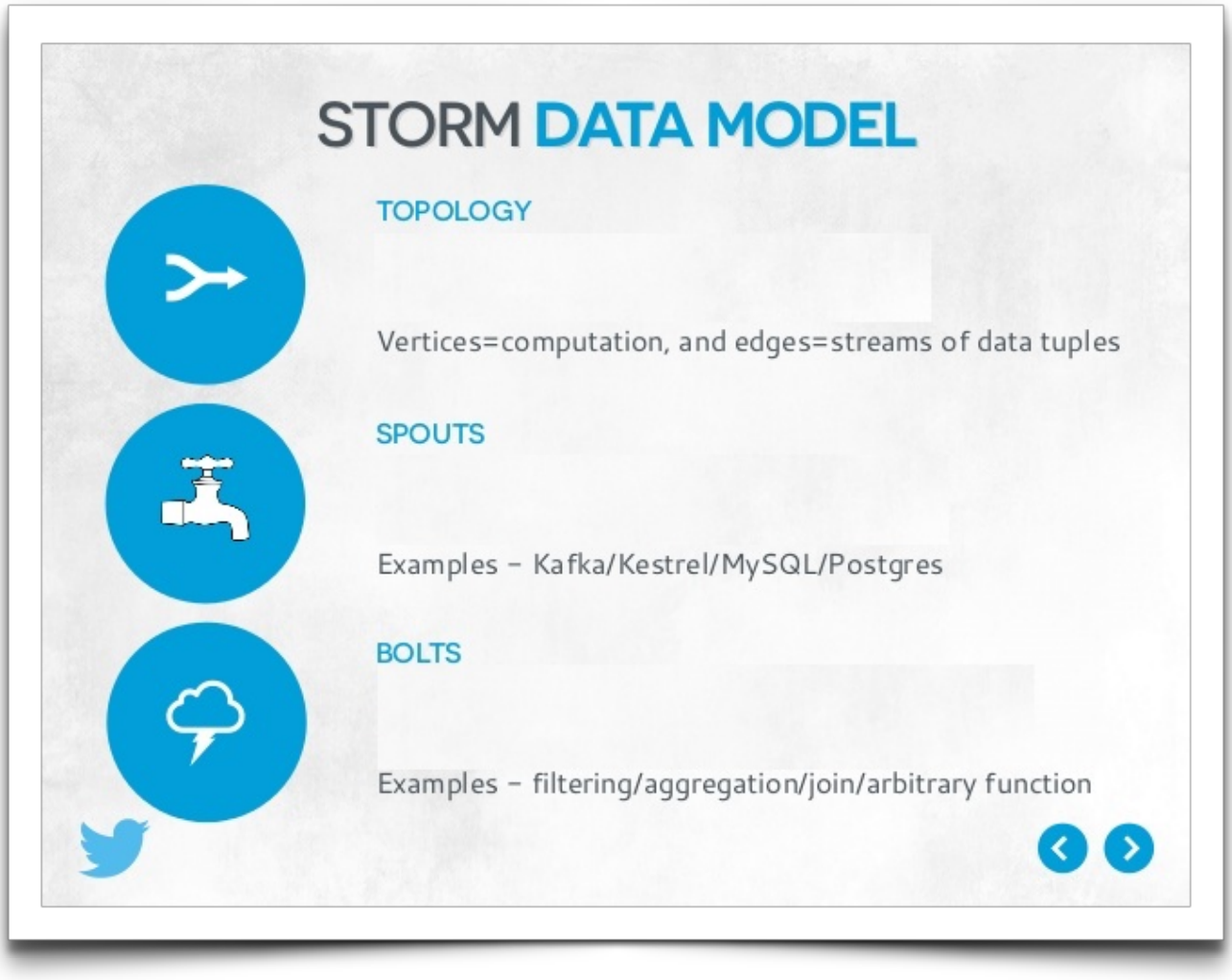
Why are we not using Hadoop for this purpose ?

Hadoop and Storm are totally complimentary .

- Storm is used for Real Time Computation , it's really fast and reactive.

- Hadoop is used of for Big Batch Processing.

STORM DATA MODEL



It has three major parts

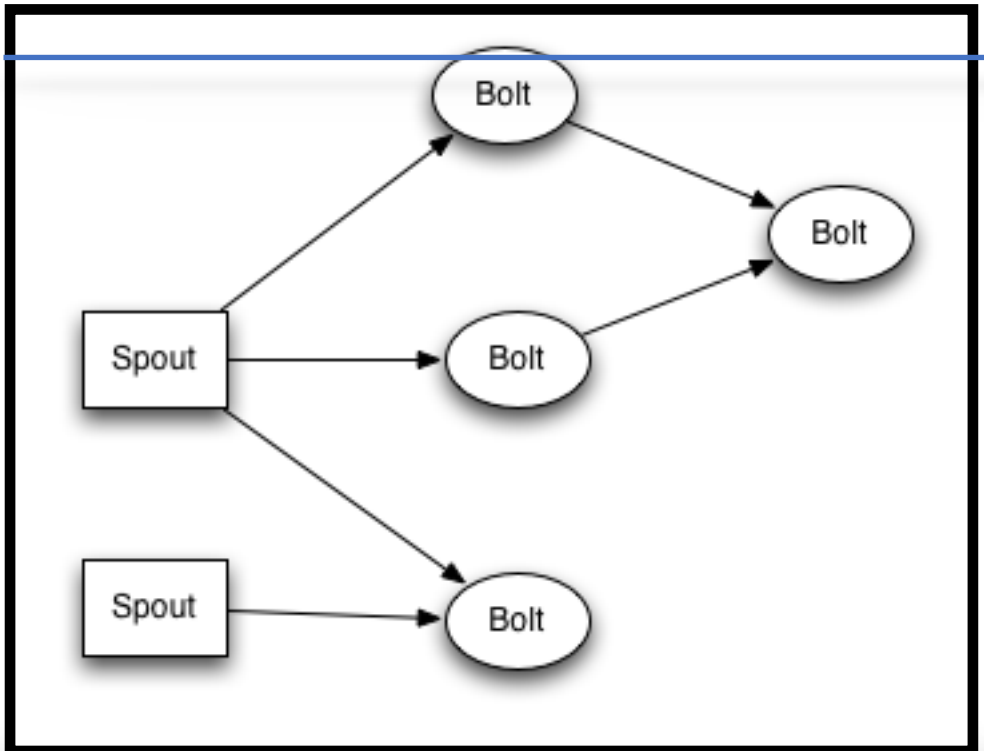
1. **Spouts :** They are the sources of the data for the entire topology.

---

2. **Bolts** :They are the units of computation of the data.

---

3. **Tuple** - They are the immutable ordered list of elements .



4. **Topology** : It consists of Directed acyclic graph with number of vertices equal to the computation and

edges equals to the streams of data.

GROUPING

When we connect a bolt to spout or a bolt to bolt we have different types of grouping mentioned below :

- **Shuffle Grouping** : In this tuple are randomly distributed to next stage bolt instances.

- **Fields Grouping** : Tuples will be grouped by a single or many tuple values.

- **All Grouping** : In this tuples are replicated to stage bolt instances.

- **Global Grouping** : In this all the tuples are sent to a single next stage bolt instance.

Word Count Topology .

- We first created tweet spout which will get the tweet .
- After this we created Parse Tweet Bolt which will process over the fetched tweet and will nbreak the tweets into words.
- Finally all the words will be sent to the Word Count Bolt which will keep track of all the words passes till a particular moment.

Note:

For visualisation we using javascript library called D3.

UNDERSTANDING STORM'S FUNCTIONALITY

PART -I

1. *First we installed virtual box and vagrant for out Operating System.*

Following are some *Ubuntu -Storm* commands.

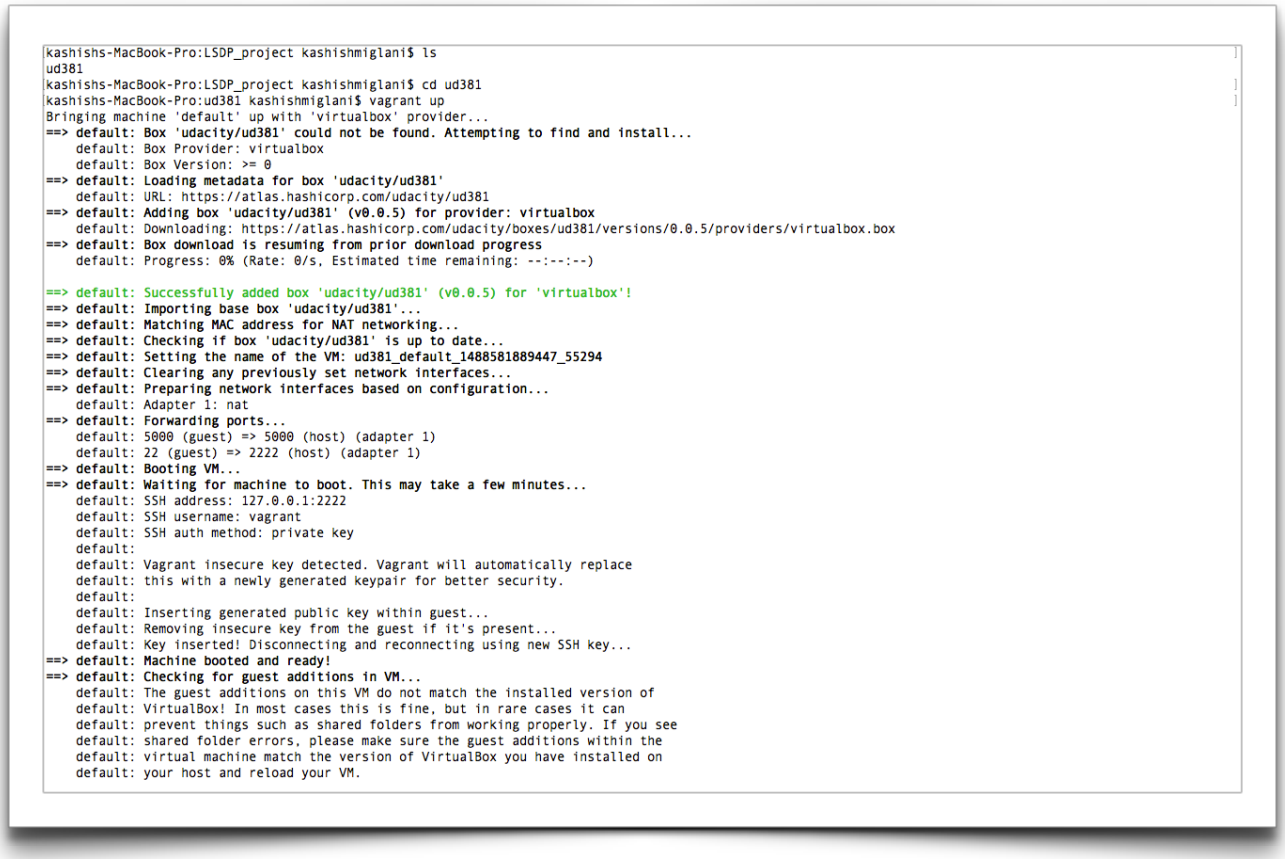
- **vagrant ssh** : When this command is run is command command prompt / git -bash , we'll automatically log into the downloaded Virtual Box.

- **storm version** : This will tell the current storm version installed in the machine.

- **cd** : it helps in changing the directory.

- **ls**: it helps in listing all the files in the Present Working Directory.

- **logout** : This will help us to logout go the virtual box.
- **tree :** This command will list all the compiled classes that exist in the path.



```
kashishs-MacBook-Pro:LSDP_project kashishmiglani$ ls
ud381
kashishs-MacBook-Pro:LSDP_project kashishmiglani$ cd ud381
kashishs-MacBook-Pro:ud381 kashishmiglani$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Box 'udacity/ud381' could not be found. Attempting to find and install...
    default: Box Provider: virtualbox
    default: Box Version: >= 0
==> default: Loading metadata for box 'udacity/ud381'
    default: URL: https://atlas.hashicorp.com/udacity/ud381
==> default: Adding box 'udacity/ud381' (v0.0.5) for provider: virtualbox
    default: Downloading: https://atlas.hashicorp.com/udacity/boxes/ud381/versions/0.0.5/providers/virtualbox.box
==> default: Box download is resuming from prior download progress
    default: Progress: 0% (Rate: 0/s, Estimated time remaining: --:--:--)

==> default: Successfully added box 'udacity/ud381' (v0.0.5) for 'virtualbox'!
==> default: Importing base box 'udacity/ud381'...
==> default: Matching MAC address for NAT networking...
==> default: Checking if box 'udacity/ud381' is up to date...
==> default: Setting the name of the VM: ud381_default_1488581889447_55294
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
==> default: Forwarding ports...
    default: 5000 (guest) => 5000 (host) (adapter 1)
    default: 22 (guest) => 2222 (host) (adapter 1)
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
    default: SSH username: vagrant
    default: SSH auth method: private key
    default:
    default: Vagrant insecure key detected. Vagrant will automatically replace
    default: this with a newly generated keypair for better security.
    default:
    default: Inserting generated public key within guest...
    default: Removing insecure key from the guest if it's present...
    default: Key inserted! Disconnecting and reconnecting using new SSH key...
==> default: Machine booted and ready!
==> default: Checking for guest additions in VM...
    default: The guest additions on this VM do not match the installed version of
    default: VirtualBox! In most cases this is fine, but in rare cases it can
    default: prevent things such as shared folders from working properly. If you see
    default: shared folder errors, please make sure the guest additions within the
    default: virtual machine match the version of VirtualBox you have installed on
    default: your host and reload your VM.
```

2. *After this we went through the **Maven build command tool** :*

    (I) This tool helps us to compile and package our storm programme.

Maven Commands

- mvn package - This command will perform all commands in the life cycle up to package including clean and compile.

3. *Finally we learnt how to submit the current topology designed.* we used the following command to submit out storm topology

storm jar<jar_file_name><file_to_be_submitted>

After the execution of this command we have to wait for at least 3-4 seconds in order to view the final output on the browser(Localhost).

---

4. *Then we connected storm to flask and d3 with the help of* ***Redis .***

   REDIS : It is an open source , in - memory data structure store , used as a database, cache and message broker. It supports data structures such as strings , hashes , lists , sets etc.

---

5. *After this we modified the project model xml file to manage project dependencies.*

- import lettuce module (to link java to redis into the POM.xml)

- This dependency must be located in the <dependencies> section.

```
<dependency>
  <groupId>com.lambdaworks</groupId>
  <artifactId>lettuce</artifactId>
  <version>2.3.3</version>
</dependency>
```

---

6. Then we compiled the topology using maven package.

---

7. After successful compilation , we will import the redis client and the redis connection in the reporter exclamation file.

a. import com.lambdaworks.redis.RedisClient
b. import
com.lambdaworks.redis.RedisConnection

---

8. Then we instantiated the redis connection using
a. RedisConnection<String,String> redis;
b. RedisClient client=new
RedisClient("localhost",6379);
c. redis=client.connect();

---

9. At last, we'll use
redis.publish("WordCountTopology",exclamatedWord.toString()+"|"+Long.toString(count)) ,to print the words separated by their count

---

10. Finally we compile again and then submitted out New Topology I.e. ReporterExclamationTopology.

OUTPUT

PART-II

- Now we'll be adding a spout to the topology to capture the data.

1. We'll import the RandomSentenceSpout in reporter exclamation topology
2. We have a RandomSentenceSpout class in which spout has been added in the beginning.
3. Then we compiled the code.
4. Now updating the topology builder
    - we added the random sentence spout using parallelism 10 and id- (rand-sentence).
    - now we added a new bolt and connected that to spout with id- (rand-=sentence) using shuffle grouping connection (having id (exclaim1))

- Adding one more new bolt which is connected to the Bolt with id- (exclaim1) using a shuffle grouping connection.

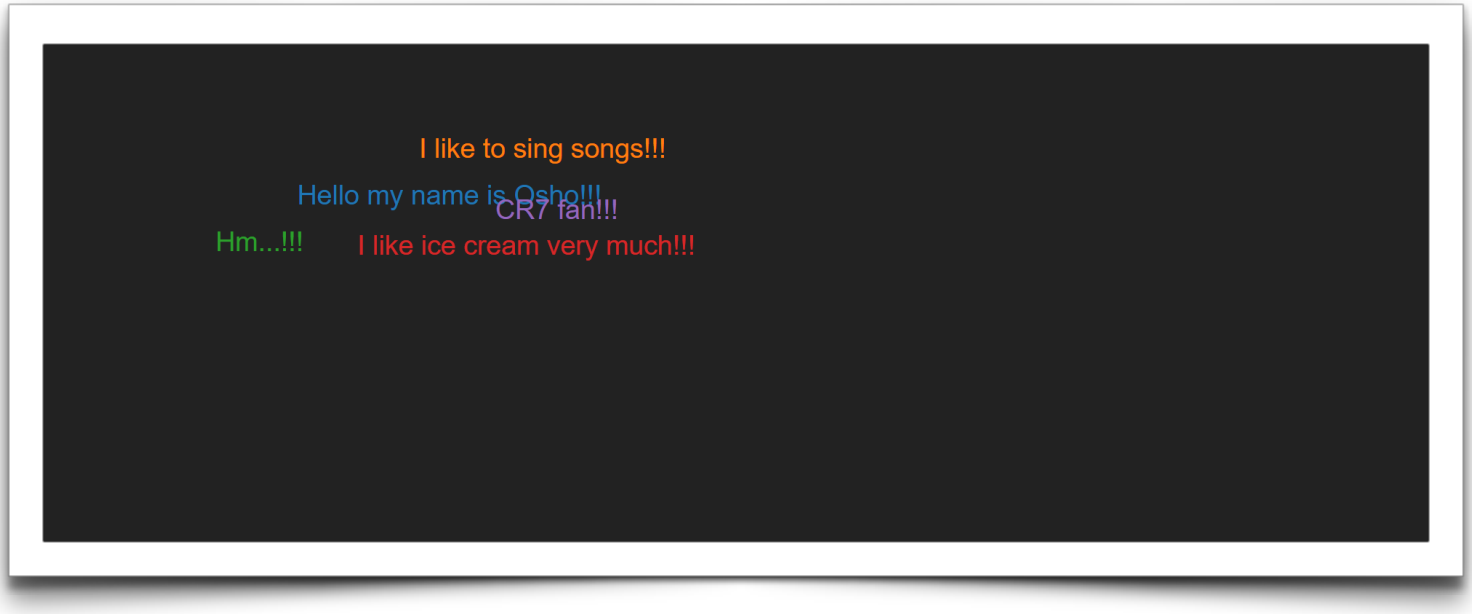
So in this way we connected A spout to a bolt and a bolt to another bolt using SHUFFLE grouping connection.

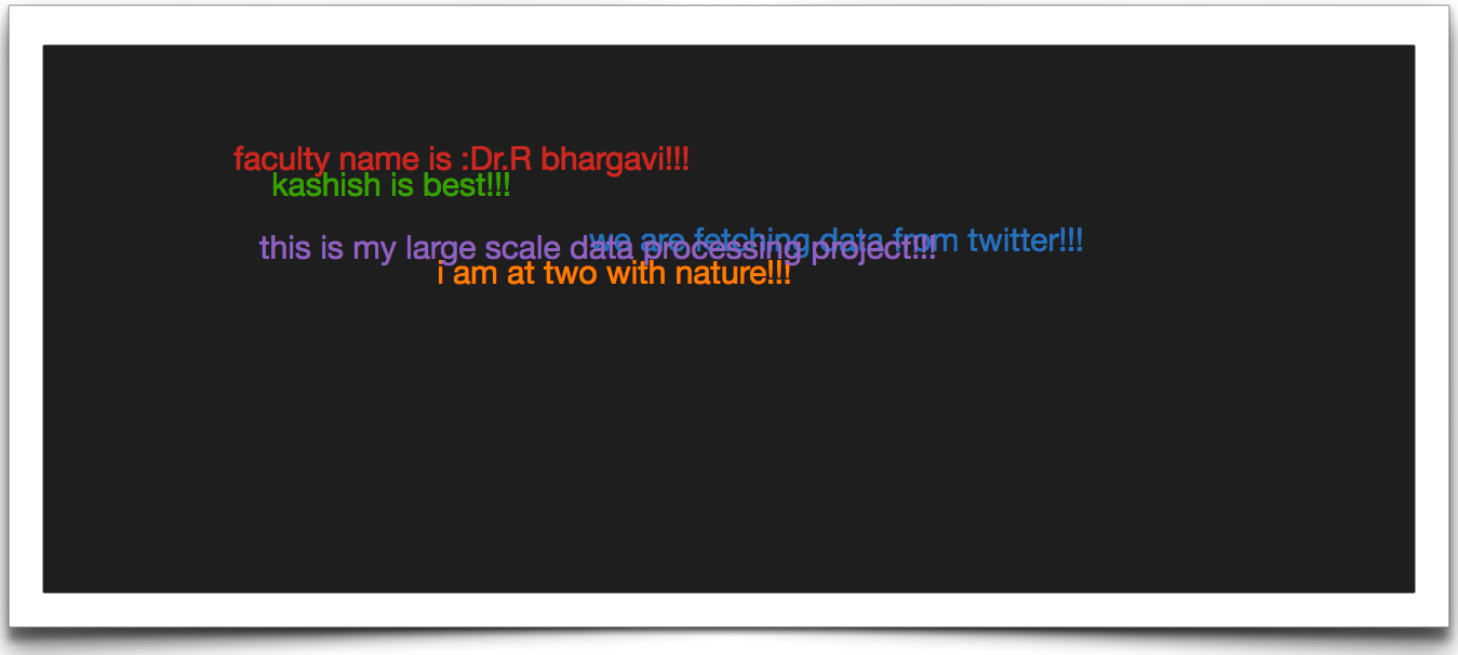
Major Commands for connection of the spouts and bolts

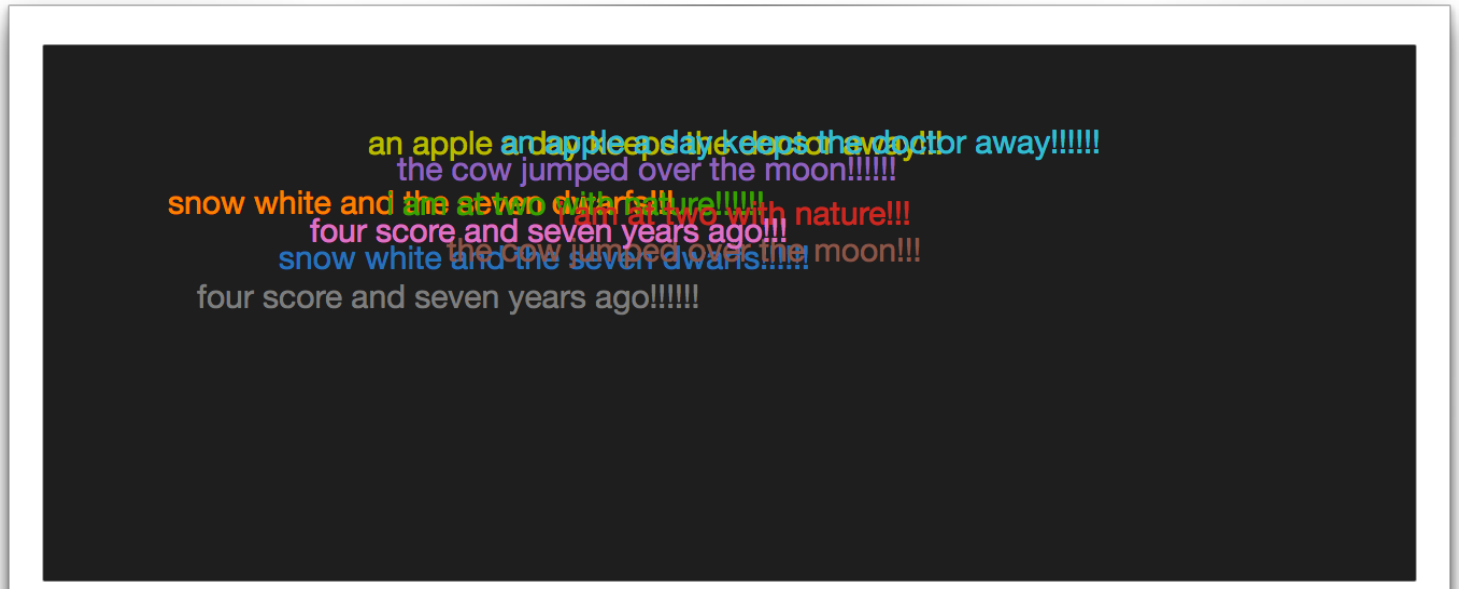
1. builder.setSpout("rand-sentence", new RandomSentenceSpout(), 10);
2. builder.setBolt("exclaim1", new ExclamationBolt(), 3).shuffleGrouping("rand-sentence");
3. builder.setBolt("exclaim2", new ExclamationBolt(), 2).shuffleGrouping("exclaim1");

FINALLY WE COMPILE AND SUBMIT THE CODE .

OUTPUT :


I like to sing songs!!!
Hello my name is Osho!!!
CR7 fan!!!
Hm...!!!
I like ice cream very much!!!


faculty name is :Dr.R bhargavi!!!
kashish is best!!!
this is my large scale data processing project!!!
we are fetching data from twitter!!!
i am at two with nature!!!


an apple a day keeps the doctor away!!!!!!
the cow jumped over the moon!!!!!!
snow white and the seven dwarfs!!!!!!
four score and seven years ago!!!
snow white and the seven dwarfs!!!!!!
four score and seven years ago!!!!!!

Part- III

(PERSONALISING THE TOPOLOGY)

1. Firstly , All the sentences in the output screen are being fetched form RandomSentenceSpout.

2. In the beginning we'll add up the connections.

    a. rand-sentence <–> exclaim1

    b. rand-sentence <–> exclaim2

    c. rand-sentence <–> exclaim1 (and) rand-sentence <–> exclaim2.

3. Then we compiled and once we got the success at the compilation time , we submitted the programme .

---

4. After that we implemented **CountBolt** methods **execute** and **declareOutputFields** to complete the Word Count Topology.

---

5. Count bolt method generally helps us in counting the number times a particular word occurred.

```
@Override
    public void execute(Tuple tuple)
    {
      String word = tuple.getString(0);
      if(countMap.containsKey(word)){

countMap.put(word,countMap.get(word)+1);
      }else{
        countMap.put(word,1);
```

```
        }
        collector.emit(new Values(word,
countMap.get(word)));
    }
```

–-This code takes in a tuple, extracts the word.

–Then it checks in the already created map that wether the word is present or not and increases the value per count.

–Finally, after countMap is updated, word along with count is emitted to the output collector.

–Then we'll be setting the output fields.

outputFieldsDeclarer.declare(new Fields("word","count"));

compile the programme now.

NOTE: The number of times a word appeared will be directly proportional to the size of that word on the visualising window.

CREATING A NEW TOPOLOGIES

Counting Words

1.Now we created a new topology

```
TopologyBuilder builder = new TopologyBuilder();
```

2.Creating a word spout

```
builder.setSpout("word-spout", new WordSpout(), 5);
```

3.Attaching the count bolt using fields grouping - parallelism of 15.

- builder.setBolt("count-bolt", new CountBolt(), 15).fieldsGrouping("word-spout", new Fields("word"));

4.Attaching the report bolt using global grouping - parallelism of 1.

- builder.setBolt("report-bolt",new ReportBolt(), 1).globalGrouping("count-bolt");

5.Publishing the word count to redis using word as the key

- redis.publish("WordCountTopology", word + "|" + Long.toString(count));

- here '|' takes the string published by Redis and splits according to '|' and loads "word" and "count".

6.FINALLY compile and run the programme.

OUTPUT:

Counting Sentences

---

1)Importing RandomSentenceSpout.

---

2)Creating a new topology for counting the word change the word-pout and connect Random sentence with id "sentence-spout".

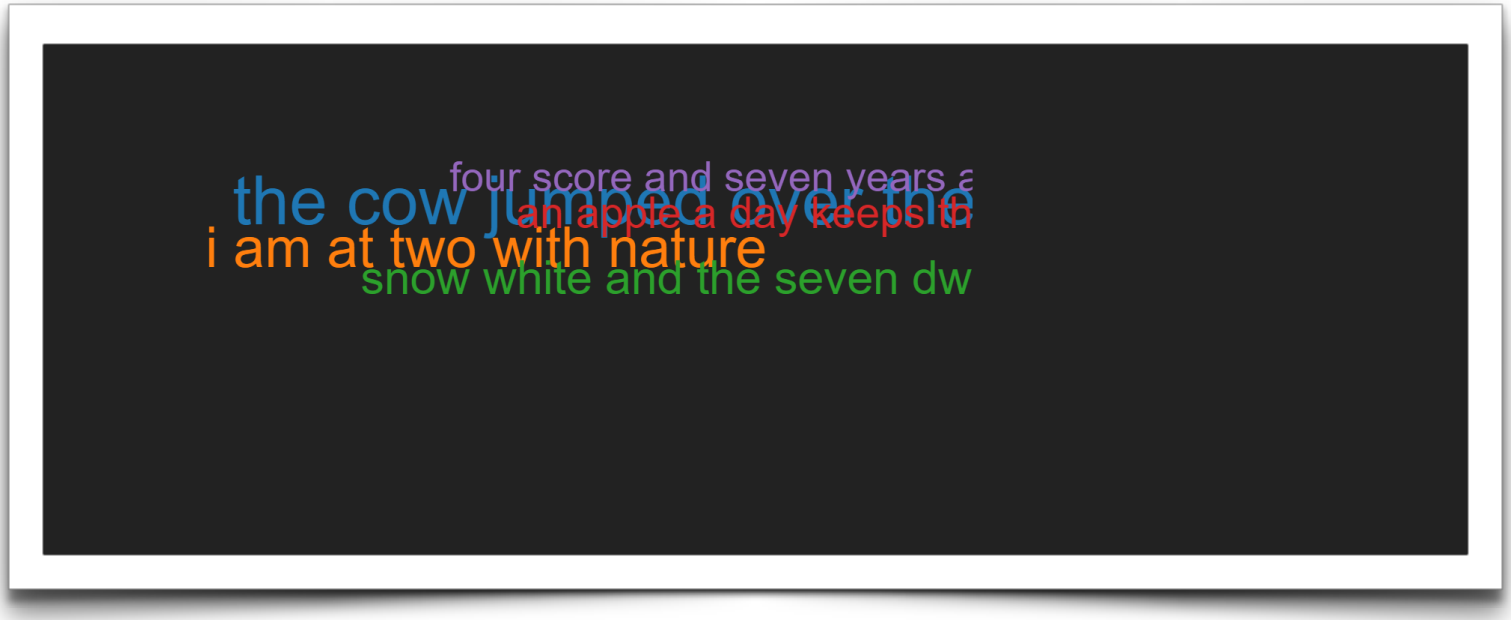
---

3)Connecting count bolt to the report bolt.

---

4)Now compile and run

OUTPUT:



Counting Words in Sentences

1) Importing RandomSentenceSpout.

2) builder.setSpout(“sentence-spout”, new RandomSentenceSpout(), 1);

3) builder.setBolt("split-sentence-bolt", new SplitSentenceBolt(), 15).shuffleGrouping(“sentence-spout");

4) builder.setBolt("count-bolt", new CountBolt(), 15).fieldsGrouping("split-sentence-bolt", new Fields(“sentence-word”));

5) builder.setBolt("report-bolt", new ReportBolt(), 1).globalGrouping(“count-bolt");

6) Now compile and run.

OUTPUT

FINALLY - OBTAINING TWITTER CREDENTIALS

Step 1) Creating a Twitter account.

Step 2) While logged in, navigate to: **https://apps.twitter.com/**

Step 3) Click "Create New App"

Step 4) In Name, enter the **name of your app, suitable description and website.**

Step 5) Click on "Generate my Access Access Token and Token Secret." then we copied these two secret keys.

Step 6) Then we saved the obtained authentication details. They were of the form:

```
    “[Our customer key]",
    "[Our secret key]",
    "[Our access token]",
    "[Our access secret]"
```

Step 7) In the file tweet topology.java , we entered the keys which we obtained Respectively

"MLWnUQpNnx296SZX9OlaAkgDP",

"1OtqpCWwRWRcF06EUNRyuB2LeyBMDDyvLl833qdlhlhoEby3u3",

"840911560616759296-4OKFD4VJKrcmtMM9rqeRp8ZHuA7SNvi",

“yrwSfgOLlrjT8rKUtem46hlMNZrMQclQZtsogqkMPVlbF"

Step 8) After this we created spout snd bolts and connected them by suitable grouping method.

Step 9) Setting the tweet spout

builder.setSpout(“tweet-spout”,tweetSpout,1);

Step10) then we attached the parse-tweet-bolt with the count bolt with a parallelism of 15 using fields Grouping.

builder.setBolt("count-bolt",new CountBolt(),fieldsGrouping(“parse-tweet-bolt”,new Fields("tweet-word"));

Step 11) Attaching the report bolt to count bolt with a parallelism of 1 using Global Grouping.

builder.setBolt("report-bolt",new ReportBolt(),1).globalGrouping(“count-bolt”);

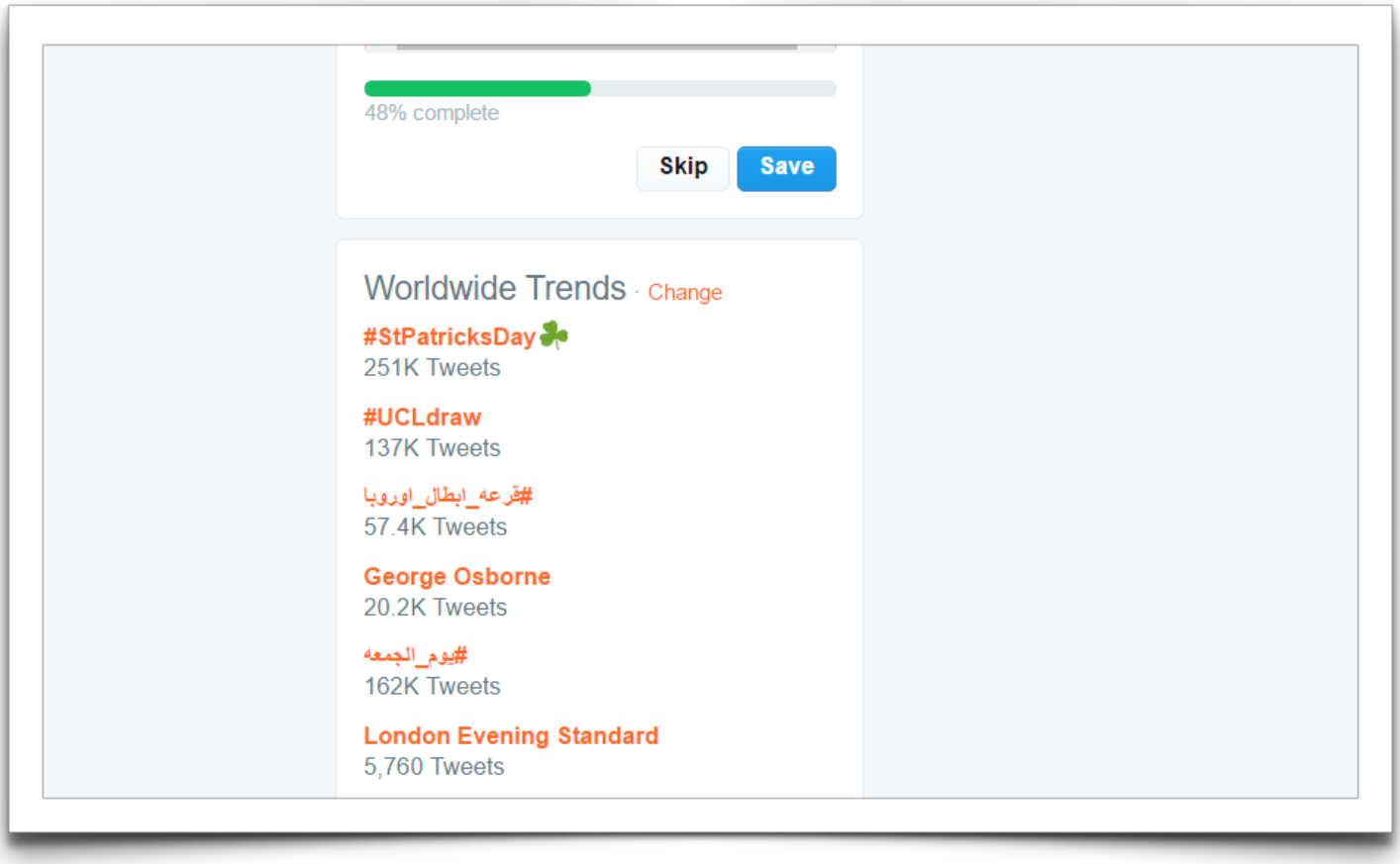
Step 12)Finally COMPILE and SUBMIT

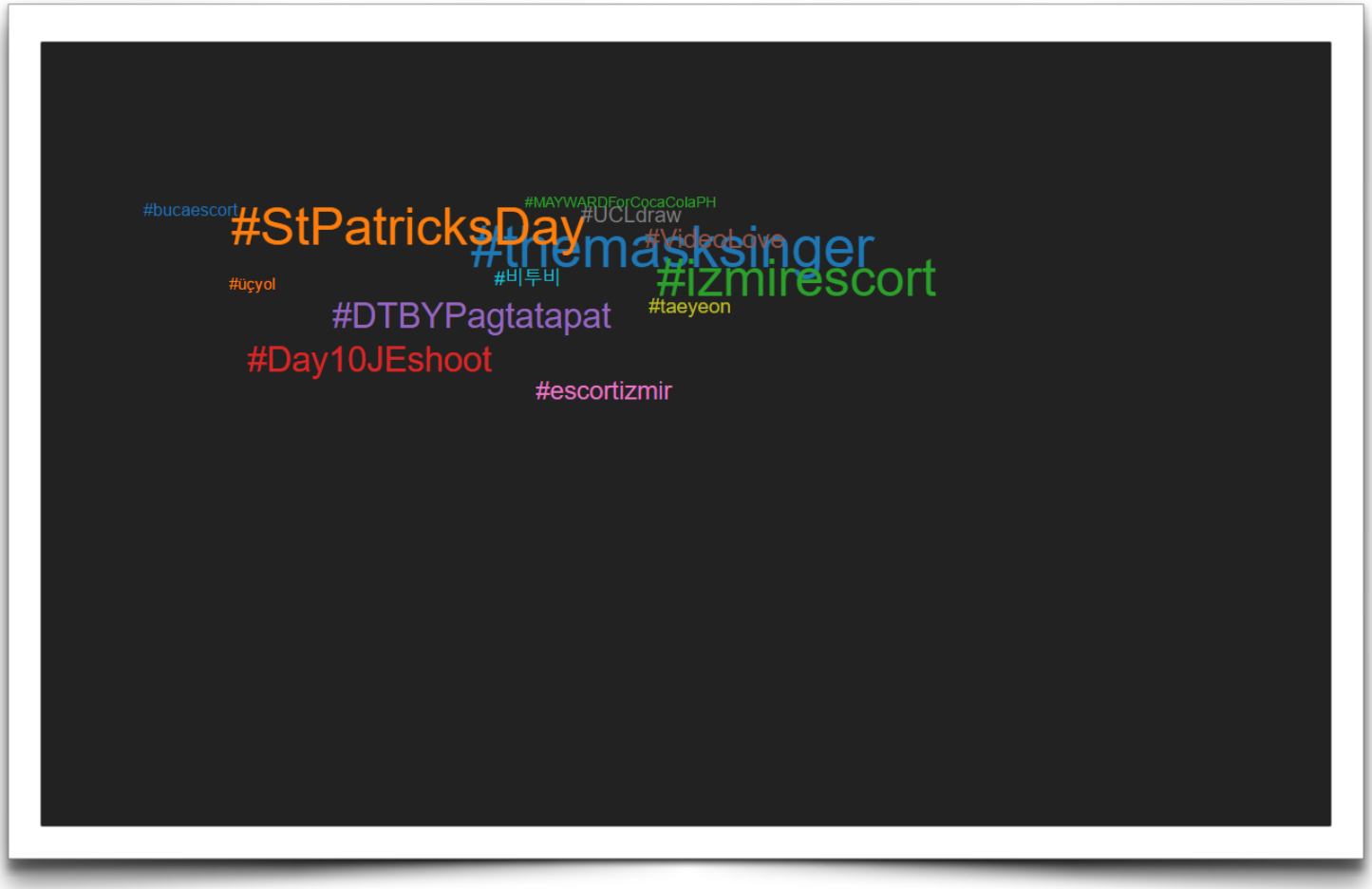
NOTE : We also implemented the **RollingCountBolt**

This bolt will help in faster counting of the fetched words by using sliding window based counting.For eg : If the window length s set to an equivalent of five minutes and the emit frequency to one minute, then the bolt will output the latest five-minute sliding window every minute.

-builder.setBolt("rolling-count-bolt",new RollingCountBolt(30,10), 15).fieldsGrouping("parse-tweet-bolt",new Fields("tweet-word"));

OUTPUT:


48% complete
Skip
Save
Worldwide Trends · Change
#StPatricksDay
251K Tweets
#UCLdraw
137K Tweets
#قرعه_ابطال_اوروبا
57.4K Tweets
George Osborne
20.2K Tweets
#يوم_الجمعه
162K Tweets
London Evening Standard
5,760 Tweets


#bucaescort
#StPatricksDay
#MAYWARDForCocaColaPH
#UCLdraw
#videoLove
#theMaskSinger
#izmirescort
#üçyol
#비투비
#taeyeon
#DTBYPagtatapat
#Day10JEshoot
#escortizmir

ALL MAIN CODES:

AbstractRankerBolt.java

```
package storm;

import backtype.storm.Config;

import backtype.storm.topology.BasicOutputCollector;

import backtype.storm.topology.OutputFieldsDeclarer;

import backtype.storm.topology.base.BaseBasicBolt;

import backtype.storm.tuple.Fields;

import backtype.storm.tuple.Tuple;

import backtype.storm.tuple.Values;

import org.apache.log4j.Logger;


import storm.tools.Rankings;

import storm.tools.TupleHelpers;
```

```
import java.util.HashMap;

import java.util.Map;



/**

 * This abstract bolt provides the basic behavior of
bolts that rank objects according to their count.

 * <p/>

 * It uses a template method design pattern for
{@link AbstractRankerBolt#execute(Tuple,
BasicOutputCollector)} to allow

 * actual bolt implementations to specify how
incoming tuples are processed, i.e. how the objects
embedded within those

 * tuples are retrieved and counted.

 */

public abstract class AbstractRankerBolt extends
BaseBasicBolt {



  private static final long serialVersionUID =
493164019850153020 2L;

  private static final int
DEFAULT_EMIT_FREQUENCY_IN_SECONDS = 2;

  private static final int DEFAULT_COUNT = 10;
```

```
  private final int emitFrequencyInSeconds;

  private final int count;

  private final Rankings rankings;


  public AbstractRankerBolt() {

    this(DEFAULT_COUNT,
DEFAULT_EMIT_FREQUENCY_IN_SECONDS);

  }


  public AbstractRankerBolt(int topN) {

    this(topN, DEFAULT_EMIT_FREQUENCY_IN_SECONDS);

  }


  public AbstractRankerBolt(int topN, int
emitFrequencyInSeconds) {

    if (topN < 1) {

      throw new IllegalArgumentException("topN must
be >= 1 (you requested " + topN + ")");

    }

    if (emitFrequencyInSeconds < 1) {
```

```
      throw new IllegalArgumentException(
          "The emit frequency must be >= 1 seconds
(you requested " + emitFrequencyInSeconds + "
seconds)");
    }
    count = topN;
    this.emitFrequencyInSeconds =
emitFrequencyInSeconds;
    rankings = new Rankings(count);
  }

  protected Rankings getRankings() {
    return rankings;
  }

  /**
   * This method functions as a template method
(design pattern).
   */
  @Override
  public final void execute(Tuple tuple,
BasicOutputCollector collector) {
```

```
    if (TupleHelpers.isTickTuple(tuple)) {
      getLogger().debug("Received tick tuple,
triggering emit of current rankings");
      emitRankings(collector);
    }
    else {
      updateRankingsWithTuple(tuple);
    }
  }

  abstract void updateRankingsWithTuple(Tuple tuple);

  private void emitRankings(BasicOutputCollector
collector) {
    collector.emit(new Values(rankings.copy()));
    getLogger().debug("Rankings: " + rankings);
  }

  @Override
  public void
declareOutputFields(OutputFieldsDeclarer declarer) {
```

```
    declarer.declare(new Fields("rankings"));
  }

  @Override
  public Map<String, Object>
getComponentConfiguration() {
    Map<String, Object> conf = new HashMap<String,
Object>();
    conf.put(Config.TOPOLOGY_TICK_TUPLE_FREQ_SECS,
emitFrequencyInSeconds);
    return conf;
  }

  abstract Logger getLogger();
}
```

CountBolt.java

```
package storm;

import backtype.storm.Config;

import backtype.storm.LocalCluster;

import backtype.storm.StormSubmitter;

import backtype.storm.spout.SpoutOutputCollector;

import backtype.storm.task.OutputCollector;

import backtype.storm.task.TopologyContext;

import backtype.storm.testing.TestWordSpout;

import backtype.storm.topology.OutputFieldsDeclarer;

import backtype.storm.topology.TopologyBuilder;

import backtype.storm.topology.base.BaseRichSpout;

import backtype.storm.topology.base.BaseRichBolt;

import backtype.storm.tuple.Fields;

import backtype.storm.tuple.Tuple;

import backtype.storm.tuple.Values;

import backtype.storm.utils.Utils;
```

```
import java.util.HashMap;

import java.util.Map;


/**

 * A bolt that counts the words that it receives

 */

public class CountBolt extends BaseRichBolt

{

  // To output tuples from this bolt to the next
stage bolts, if any

  private OutputCollector collector;


  // Map to store the count of the words

  private Map<String, Long> countMap;


  @Override

  public void prepare(

      Map                     map,

      TopologyContext         topologyContext,

      OutputCollector         outputCollector)
```

```
  {

    // save the collector for emitting tuples
    collector = outputCollector;

    // create and initialize the map
    countMap = new HashMap<String, Long>();
  }

  @Override
  public void execute(Tuple tuple)
  {
    // get the word from the 1st column of incoming
tuple
    String word = tuple.getString(0);

    // check if the word is present in the map
    if (countMap.get(word) == null) {

      // not present, add the word with a count of 1
```

```
      countMap.put(word, 1L);
    } else {

      // already there, hence get the count
      Long val = countMap.get(word);

      // increment the count and save it to the map
      countMap.put(word, ++val);
    }

    // emit the word and count
    collector.emit(new Values(word,
countMap.get(word)));
  }

  @Override
  public void
declareOutputFields(OutputFieldsDeclarer
outputFieldsDeclarer)
  {
    // tell storm the schema of the output tuple for
this spout
```

```
    // tuple consists of a two columns called 'word'
and 'count'



    // declare the first column 'word', second column
'count'

    outputFieldsDeclarer.declare(new
Fields("word","count"));

  }

}
```

IntermediateRankingsBolt.java

```
package storm;

import backtype.storm.tuple.Tuple;

import org.apache.log4j.Logger;

import udacity.storm.tools.Rankable;

import udacity.storm.tools.RankableObjectWithFields;


/**

 * This bolt ranks incoming objects by their count.

 * <p/>

 * It assumes the input tuples to adhere to the
following format: (object, object_count,
additionalField1,

 * additionalField2, ..., additionalFieldN).

 */

public final class IntermediateRankingsBolt extends
AbstractRankerBolt {


  private static final long serialVersionUID =
-136980053025663740 9L;
```

```
  private static final Logger LOG =
Logger.getLogger(IntermediateRankingsBolt.class);

  public IntermediateRankingsBolt() {
    super();
  }

  public IntermediateRankingsBolt(int topN) {
    super(topN);
  }

  public IntermediateRankingsBolt(int topN, int
emitFrequencyInSeconds) {
    super(topN, emitFrequencyInSeconds);
  }

  @Override
  void updateRankingsWithTuple(Tuple tuple) {
    Rankable rankable =
RankableObjectWithFields.from(tuple);
    super.getRankings().updateWith(rankable);
```

```
  }

  @Override
  Logger getLogger() {
    return LOG;
  }
}
```

ParsetweetBolt.java

```
package storm;

import backtype.storm.Config;

import backtype.storm.LocalCluster;

import backtype.storm.StormSubmitter;

import backtype.storm.spout.SpoutOutputCollector;

import backtype.storm.task.OutputCollector;

import backtype.storm.task.TopologyContext;

import backtype.storm.testing.TestWordSpout;

import backtype.storm.topology.OutputFieldsDeclarer;

import backtype.storm.topology.TopologyBuilder;

import backtype.storm.topology.base.BaseRichSpout;

import backtype.storm.topology.base.BaseRichBolt;

import backtype.storm.tuple.Fields;

import backtype.storm.tuple.Tuple;

import backtype.storm.tuple.Values;

import backtype.storm.utils.Utils;
```

```
import java.util.Map;

import java.util.Arrays;


/**

 * A bolt that parses the tweet into words

 */

public class ParseTweetBolt extends BaseRichBolt

{

  // To output tuples from this bolt to the count
bolt

  OutputCollector collector;



  private String[] skipWords = {"rt", "to",
"me","la","on","that","que",


"followers","watch","know","not","have","like","I'm",
"new","good","do",


"more","es","te","followers","Followers","las","you",
"and","de","my","is",
```

```
"en","una","in","for","this","go","en","all","no","do
n't","up","are",

"http","http:","https","https:","http://","https://",
"with","just","your",

"para","want","your","you're","really","video","it's"
,"when","they","their","much",

"would","what","them","todo","FOLLOW","retweet","RETW
EET","even","right","like",

"bien","Like","will","Will","pero","Pero","can't","we
re","Can't","Were","TWITTER",

"make","take","This","from","about","como","esta","fo
llows","followed"};


  @Override
  public void prepare(
      Map                     map,
      TopologyContext         topologyContext,
      OutputCollector         outputCollector)
  {
```

```
    // save the output collector for emitting tuples
    collector = outputCollector;
  }

  @Override
  public void execute(Tuple tuple)
  {
    // get the 1st column 'tweet' from tuple
    String tweet = tuple.getString(0);

    // provide the delimiters for splitting the tweet
    String delims = "[ .,?!]+";

    // now split the tweet into tokens
    String[] tokens = tweet.split(delims);

    // for each token/word, emit it
    for (String token: tokens) {
      //emit only words greater than length 3 and not
stopword list
```

```
      if(token.length() > 3 && !
Arrays.asList(skipWords).contains(token)){

        if(token.startsWith("#")){

          collector.emit(new Values(token));

        }

      }

    }

  }


  @Override

  public void
declareOutputFields(OutputFieldsDeclarer declarer)

  {

    // tell storm the schema of the output tuple for
this spout

    // tuple consists of a single column called
'tweet-word'

    declarer.declare(new Fields("tweet-word"));

  }


}
```

ReportBolt.java

```
package storm;

import backtype.storm.Config;

import backtype.storm.LocalCluster;

import backtype.storm.StormSubmitter;

import backtype.storm.spout.SpoutOutputCollector;

import backtype.storm.task.OutputCollector;

import backtype.storm.task.TopologyContext;

import backtype.storm.testing.TestWordSpout;

import backtype.storm.topology.OutputFieldsDeclarer;

import backtype.storm.topology.TopologyBuilder;

import backtype.storm.topology.base.BaseRichSpout;

import backtype.storm.topology.base.BaseRichBolt;

import backtype.storm.tuple.Fields;

import backtype.storm.tuple.Tuple;

import backtype.storm.tuple.Values;

import backtype.storm.utils.Utils;



import java.util.Map;
```

```
import com.lambdaworks.redis.RedisClient;
import com.lambdaworks.redis.RedisConnection;

import udacity.storm.tools.*;
import udacity.storm.tools.Rankings;
import com.google.common.collect.ImmutableList;
import com.google.common.collect.Lists;

/**
 * A bolt that prints the word and count to redis
 */
public class ReportBolt extends BaseRichBolt
{
  // place holder to keep the connection to redis
  transient RedisConnection<String,String> redis;

  @Override
  public void prepare(
      Map                     map,
```

```
      TopologyContext          topologyContext,

      OutputCollector          outputCollector)

  {

    // instantiate a redis connection

    RedisClient client = new RedisClient("localhost",
6379);



    // initiate the actual connection

    redis = client.connect();

  }



  @Override

  public void execute(Tuple tuple)

  {

    Rankings rankableList = (Rankings)
tuple.getValue(0);



    for (Rankable r: rankableList.getRankings()){

      String word = r.getObject().toString();

      Long count = r.getCount();
```

```
      redis.publish("WordCountTopology", word + "|" +
Long.toString(count));

    }



    // access the first column 'word'

  }



  public void
declareOutputFields(OutputFieldsDeclarer declarer)

  {

    // nothing to add - since it is the final bolt

  }

}
```

RollingCountBolt.java

```
package storm;

import backtype.storm.Config;

import backtype.storm.task.OutputCollector;

import backtype.storm.task.TopologyContext;

import backtype.storm.topology.OutputFieldsDeclarer;

import backtype.storm.topology.base.BaseRichBolt;

import backtype.storm.tuple.Fields;

import backtype.storm.tuple.Tuple;

import backtype.storm.tuple.Values;

import org.apache.log4j.Logger;


import storm.tools.*;


import java.util.HashMap;

import java.util.Map;

import java.util.Map.Entry;


/**
```

```
 * This bolt performs rolling counts of incoming objects, i.e. sliding window based counting.

 * <p/>

 * The bolt is configured by two parameters, the length of the sliding window in seconds (which influences the output

 * data of the bolt, i.e. how it will count objects) and the emit frequency in seconds (which influences how often the

 * bolt will output the latest window counts). For instance, if the window length is set to an equivalent of five

 * minutes and the emit frequency to one minute, then the bolt will output the latest five-minute sliding window every

 * minute.

 * <p/>

 * The bolt emits a rolling count tuple per object, consisting of the object itself, its latest rolling count, and the

 * actual duration of the sliding window. The latter is included in case the expected sliding window length (as

 * configured by the user) is different from the actual length, e.g. due to high system load. Note that the actual
```

```
 * window length is tracked and calculated for the
window, and not individually for each object within a
window.

 * <p/>

 * Note: During the startup phase you will usually
observe that the bolt warns you about the actual
sliding window

 * length being smaller than the expected length.
This behavior is expected and is caused by the way
the sliding window

 * counts are initially "loaded up". You can safely
ignore this warning during startup (e.g. you will see
this warning

 * during the first ~ five minutes of startup time if
the window length is set to five minutes).

 */

public class RollingCountBolt extends BaseRichBolt {



  private static final long serialVersionUID =
5537727428628598519L;

  private static final Logger LOG =
Logger.getLogger(RollingCountBolt.class);

  private static final int NUM_WINDOW_CHUNKS = 5;

  private static final int
DEFAULT_SLIDING_WINDOW_IN_SECONDS = NUM_WINDOW_CHUNKS
* 60;
```

```
  private static final int
DEFAULT_EMIT_FREQUENCY_IN_SECONDS =
DEFAULT_SLIDING_WINDOW_IN_SECONDS /
NUM_WINDOW_CHUNKS;

  private static final String
WINDOW_LENGTH_WARNING_TEMPLATE =

      "Actual window length is %d seconds when it
should be %d seconds"

          + " (you can safely ignore this warning
during the startup phase)";



  private final SlidingWindowCounter<Object> counter;

  private final int windowLengthInSeconds;

  private final int emitFrequencyInSeconds;

  private OutputCollector collector;

  private NthLastModifiedTimeTracker
lastModifiedTracker;



  public RollingCountBolt() {

    this(DEFAULT_SLIDING_WINDOW_IN_SECONDS,
DEFAULT_EMIT_FREQUENCY_IN_SECONDS);

  }
```

```
  public RollingCountBolt(int windowLengthInSeconds,
int emitFrequencyInSeconds) {

    this.windowLengthInSeconds =
windowLengthInSeconds;

    this.emitFrequencyInSeconds =
emitFrequencyInSeconds;

    counter = new
SlidingWindowCounter<Object>(deriveNumWindowChunksFro
m(this.windowLengthInSeconds,

        this.emitFrequencyInSeconds));

  }



  private int deriveNumWindowChunksFrom(int
windowLengthInSeconds, int
windowUpdateFrequencyInSeconds) {

    return windowLengthInSeconds /
windowUpdateFrequencyInSeconds;

  }



  @SuppressWarnings("rawtypes")

  @Override

  public void prepare(Map stormConf, TopologyContext
context, OutputCollector collector) {

    this.collector = collector;
```

```
    lastModifiedTracker = new
NthLastModifiedTimeTracker(deriveNumWindowChunksFrom(
this.windowLengthInSeconds,
        this.emitFrequencyInSeconds));
  }

  @Override
  public void execute(Tuple tuple) {
    if (TupleHelpers.isTickTuple(tuple)) {
      LOG.debug("Received tick tuple, triggering emit
of current window counts");
      emitCurrentWindowCounts();
    }
    else {
      countObjAndAck(tuple);
    }
  }

  private void emitCurrentWindowCounts() {
    Map<Object, Long> counts =
counter.getCountsThenAdvanceWindow();
```

```
    int actualWindowLengthInSeconds =
lastModifiedTracker.secondsSinceOldestModification();

    lastModifiedTracker.markAsModified();

    if (actualWindowLengthInSeconds !=
windowLengthInSeconds) {


LOG.warn(String.format(WINDOW_LENGTH_WARNING_TEMPLATE
, actualWindowLengthInSeconds,
windowLengthInSeconds));

    }

    emit(counts, actualWindowLengthInSeconds);

  }



  private void emit(Map<Object, Long> counts, int
actualWindowLengthInSeconds) {

    for (Entry<Object, Long> entry :
counts.entrySet()) {

      Object obj = entry.getKey();

      Long count = entry.getValue();

      //LK changed due to Integer to Long cast
runtime error.

      //Integer intCount = count != null ?
count.intValue() : null;
```

```
      //collector.emit(new Values(obj, intCount,
actualWindowLengthInSeconds));

      Long longCount = count != null ? count : null;

      collector.emit(new Values(obj, longCount,
actualWindowLengthInSeconds));

    }

  }


  private void countObjAndAck(Tuple tuple) {

    Object obj = tuple.getValue(0);

    counter.incrementCount(obj);

    collector.ack(tuple);

  }


  @Override

  public void
declareOutputFields(OutputFieldsDeclarer declarer) {

    declarer.declare(new Fields("obj", "count",
"actualWindowLengthInSeconds"));

  }


  @Override
```

```
  public Map<String, Object>
getComponentConfiguration() {

    Map<String, Object> conf = new HashMap<String,
Object>();

    conf.put(Config.TOPOLOGY_TICK_TUPLE_FREQ_SECS,
emitFrequencyInSeconds);

    return conf;

  }

}
```

TopNTweetTopology.java

```
package storm;

import backtype.storm.Config;

import backtype.storm.LocalCluster;

import backtype.storm.StormSubmitter;

import backtype.storm.spout.SpoutOutputCollector;

import backtype.storm.task.OutputCollector;

import backtype.storm.task.TopologyContext;

import backtype.storm.testing.TestWordSpout;

import backtype.storm.topology.OutputFieldsDeclarer;

import backtype.storm.topology.TopologyBuilder;

import backtype.storm.topology.base.BaseRichSpout;

import backtype.storm.topology.base.BaseRichBolt;

import backtype.storm.tuple.Fields;

import backtype.storm.tuple.Tuple;

import backtype.storm.tuple.Values;

import backtype.storm.utils.Utils;


import storm.spout.RandomSentenceSpout;
```

```
class TopNTweetTopology

{

  public static void main(String[] args) throws
Exception

  {

    //Variable TOP_N number of words

    int TOP_N = 10;

    // create the topology

    TopologyBuilder builder = new TopologyBuilder();




    // now create the tweet spout with the
credentials

    TweetSpout tweetSpout = new TweetSpout(

    "MLWnUQpNnx296SZX9OlaAkgDP",


"1OtqpCWwRWRcF06EUNRyuB2LeyBMDDyvLI833qdlhlhoEby3u3",


"840911560616759296-4OKFD4VJKrcmtMM9rqeRp8ZHuA7SNvi",

    "yrwSfgOLIrjT8rKUtem46hlMNZrMQcIQZtsogqkMPVIbF"
```

```
    );


    // attach the tweet spout to the topology -
parallelism of 1

    builder.setSpout("tweet-spout", tweetSpout, 1);


    // attach the Random Sentence Spout to the
topology - parallelism of 1

    //builder.setSpout("random-sentence-spout", new
RandomSentenceSpout(), 1);


    // attach the parse tweet bolt using shuffle
grouping

    builder.setBolt("parse-tweet-bolt", new
ParseTweetBolt(), 10).shuffleGrouping("tweet-spout");

    //builder.setBolt("parse-tweet-bolt", new
ParseTweetBolt(), 10).shuffleGrouping("random-
sentence-spout");


    // attach the count bolt using fields grouping -
parallelism of 15

    builder.setBolt("count-bolt", new CountBolt(),
15).fieldsGrouping("parse-tweet-bolt", new
Fields("tweet-word"));
```

```
    // attach rolling count bolt using fields
grouping - parallelism of 5

    // TEST

    //builder.setBolt("rolling-count-bolt", new
RollingCountBolt(30, 10), 1).fieldsGrouping("parse-
tweet-bolt", new Fields("tweet-word"));



    //from incubator-storm/.../storm/starter/
RollingTopWords.java

    //builder.setBolt("intermediate-ranker", new
IntermediateRankingsBolt(TOP_N),
4).fieldsGrouping("rolling-count-bolt", new
Fields("obj"));



    builder.setBolt("intermediate-ranker", new
IntermediateRankingsBolt(TOP_N),
4).fieldsGrouping("count-bolt", new Fields("word"));

    builder.setBolt("total-ranker", new
TotalRankingsBolt(TOP_N)).globalGrouping("intermediat
e-ranker");



    // attach the report bolt using global grouping -
parallelism of 1

    builder.setBolt("report-bolt", new ReportBolt(),
1).globalGrouping("total-ranker");
```

```
    // create the default config object
    Config conf = new Config();

    // set the config in debugging mode
    conf.setDebug(true);

    if (args != null && args.length > 0) {

      // run it in a live cluster

      // set the number of workers for running all
spout and bolt tasks
      conf.setNumWorkers(3);

      // create the topology and submit with config
      StormSubmitter.submitTopology(args[0], conf,
builder.createTopology());

    } else {
```

```
        // run it in a simulated local cluster


        // set the number of threads to run - similar
to setting number of workers in live cluster

        conf.setMaxTaskParallelism(3);


        // create the local cluster instance

        LocalCluster cluster = new LocalCluster();


        // submit the topology to the local cluster

        cluster.submitTopology("tweet-word-count",
conf, builder.createTopology());


        // let the topology run for 300 seconds. note
topologies never terminate!

        Utils.sleep(300000);


        // now kill the topology

        cluster.killTopology("tweet-word-count");


        // we are done, so shutdown the local cluster
```

```
        cluster.shutdown();

      }

    }

}
```

TotalRankingsBolt.java

```
package storm;

import backtype.storm.tuple.Tuple;

import org.apache.log4j.Logger;

import storm.tools.Rankings;


/**

 * This bolt merges incoming {@link Rankings}.

 * <p/>

 * It can be used to merge intermediate rankings
generated by {@link IntermediateRankingsBolt} into a
final,

 * consolidated ranking. To do so, configure this
bolt with a globalGrouping on {@link
IntermediateRankingsBolt}.

 */

public final class TotalRankingsBolt extends
AbstractRankerBolt {


  private static final long serialVersionUID =
-8447525895532302198L;
```

```
  private static final Logger LOG =
Logger.getLogger(TotalRankingsBolt.class);


  public TotalRankingsBolt() {
    super();
  }


  public TotalRankingsBolt(int topN) {
    super(topN);
  }


  public TotalRankingsBolt(int topN, int
emitFrequencyInSeconds) {
    super(topN, emitFrequencyInSeconds);
  }


  @Override
  void updateRankingsWithTuple(Tuple tuple) {
    Rankings rankingsToBeMerged = (Rankings)
tuple.getValue(0);

super.getRankings().updateWith(rankingsToBeMerged);
```

```
    super.getRankings().pruneZeroCounts();
  }

  @Override
  Logger getLogger() {
    return LOG;
  }

}
```

TweetSpout.java

```
package storm;

import backtype.storm.Config;

import backtype.storm.LocalCluster;

import backtype.storm.StormSubmitter;

import backtype.storm.spout.SpoutOutputCollector;

import backtype.storm.task.OutputCollector;

import backtype.storm.task.TopologyContext;

import backtype.storm.testing.TestWordSpout;

import backtype.storm.topology.OutputFieldsDeclarer;

import backtype.storm.topology.TopologyBuilder;

import backtype.storm.topology.base.BaseRichSpout;

import backtype.storm.topology.base.BaseRichBolt;

import backtype.storm.tuple.Fields;

import backtype.storm.tuple.Tuple;

import backtype.storm.tuple.Values;
```

```
import backtype.storm.utils.Utils;


import twitter4j.conf.ConfigurationBuilder;
import twitter4j.TwitterStream;
import twitter4j.TwitterStreamFactory;
import twitter4j.Status;
import twitter4j.StatusDeletionNotice;
import twitter4j.StatusListener;
import twitter4j.StallWarning;


import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.LinkedBlockingQueue;


/**
 * A spout that uses Twitter streaming API for
continuously
 * getting tweets
 */
public class TweetSpout extends BaseRichSpout
```

```
{

  // Twitter API authentication credentials

  String custkey, custsecret;

  String accesstoken, accesssecret;



  // To output tuples from spout to the next stage
bolt

  SpoutOutputCollector collector;



  // Twitter4j - twitter stream to get tweets

  TwitterStream twitterStream;



  // Shared queue for getting buffering tweets
received

  LinkedBlockingQueue<String> queue = null;



  // Class for listening on the tweet stream - for
twitter4j

  private class TweetListener implements
StatusListener {
```

```
    // Implement the callback function when a tweet
arrives

    @Override

    public void onStatus(Status status)

    {

      // add the tweet into the queue buffer

      queue.offer(status.getText());

    }



    @Override

    public void onDeletionNotice(StatusDeletionNotice
sdn)

    {

    }



    @Override

    public void onTrackLimitationNotice(int i)

    {

    }



    @Override
```

```
    public void onScrubGeo(long l, long l1)
    {
    }

    @Override
    public void onStallWarning(StallWarning warning)
    {
    }

    @Override
    public void onException(Exception e)
    {
      e.printStackTrace();
    }
  };

  /**
   * Constructor for tweet spout that accepts the
credentials
   */
```

```
public TweetSpout(
    String                key,
    String                secret,
    String                token,
    String                tokensecret)
{
  custkey = key;
  custsecret = secret;
  accesstoken = token;
  accesssecret = tokensecret;
}

@Override
public void open(
    Map                     map,
    TopologyContext         topologyContext,
    SpoutOutputCollector    spoutOutputCollector)
{
  // create the buffer to block tweets
  queue = new LinkedBlockingQueue<String>(1000);
```

```
    // save the output collector for emitting tuples
    collector = spoutOutputCollector;



    // build the config with credentials for twitter
4j
    ConfigurationBuilder config =
        new ConfigurationBuilder()
                .setOAuthConsumerKey(custkey)
                .setOAuthConsumerSecret(custsecret)
                .setOAuthAccessToken(accesstoken)
                .setOAuthAccessTokenSecret(accesssecre
t);


    // create the twitter stream factory with the
config
    TwitterStreamFactory fact =
        new TwitterStreamFactory(config.build());


    // get an instance of twitter stream
```

```
    twitterStream = fact.getInstance();

    // provide the handler for twitter stream
    twitterStream.addListener(new TweetListener());

    // start the sampling of tweets
    twitterStream.sample();
  }

  @Override
  public void nextTuple()
  {
    // try to pick a tweet from the buffer
    String ret = queue.poll();

    // if no tweet is available, wait for 50 ms and
return
    if (ret==null)
    {
      Utils.sleep(50);
```

```
      return;
    }

    // now emit the tweet to next stage bolt
    collector.emit(new Values(ret));
  }

  @Override
  public void close()
  {
    // shutdown the stream - when we are going to
exit
    twitterStream.shutdown();
  }

  /**
   * Component specific configuration
   */
  @Override
  public Map<String, Object>
getComponentConfiguration()
```

```
  {
    // create the component config
    Config ret = new Config();

    // set the parallelism for this spout to be 1
    ret.setMaxTaskParallelism(1);

    return ret;
  }

  @Override
  public void declareOutputFields(
      OutputFieldsDeclarer outputFieldsDeclarer)
  {
    // tell storm the schema of the output tuple for
this spout
    // tuple consists of a single column called
'tweet'
    outputFieldsDeclarer.declare(new
Fields("tweet"));
  }}
```

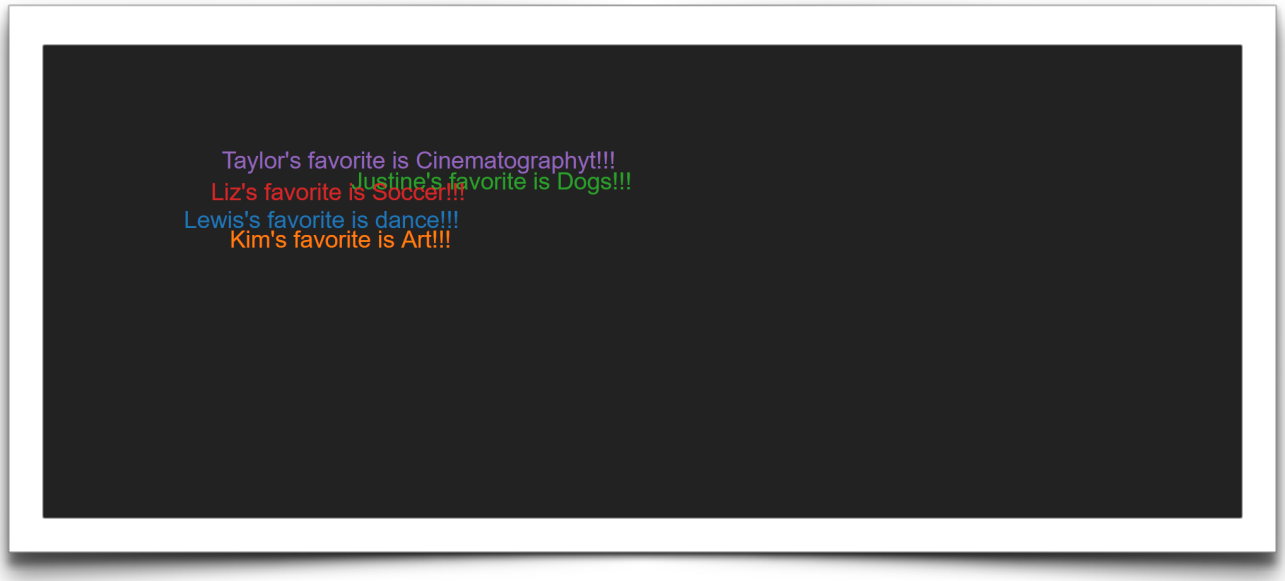
RandomSentenceSpout.java

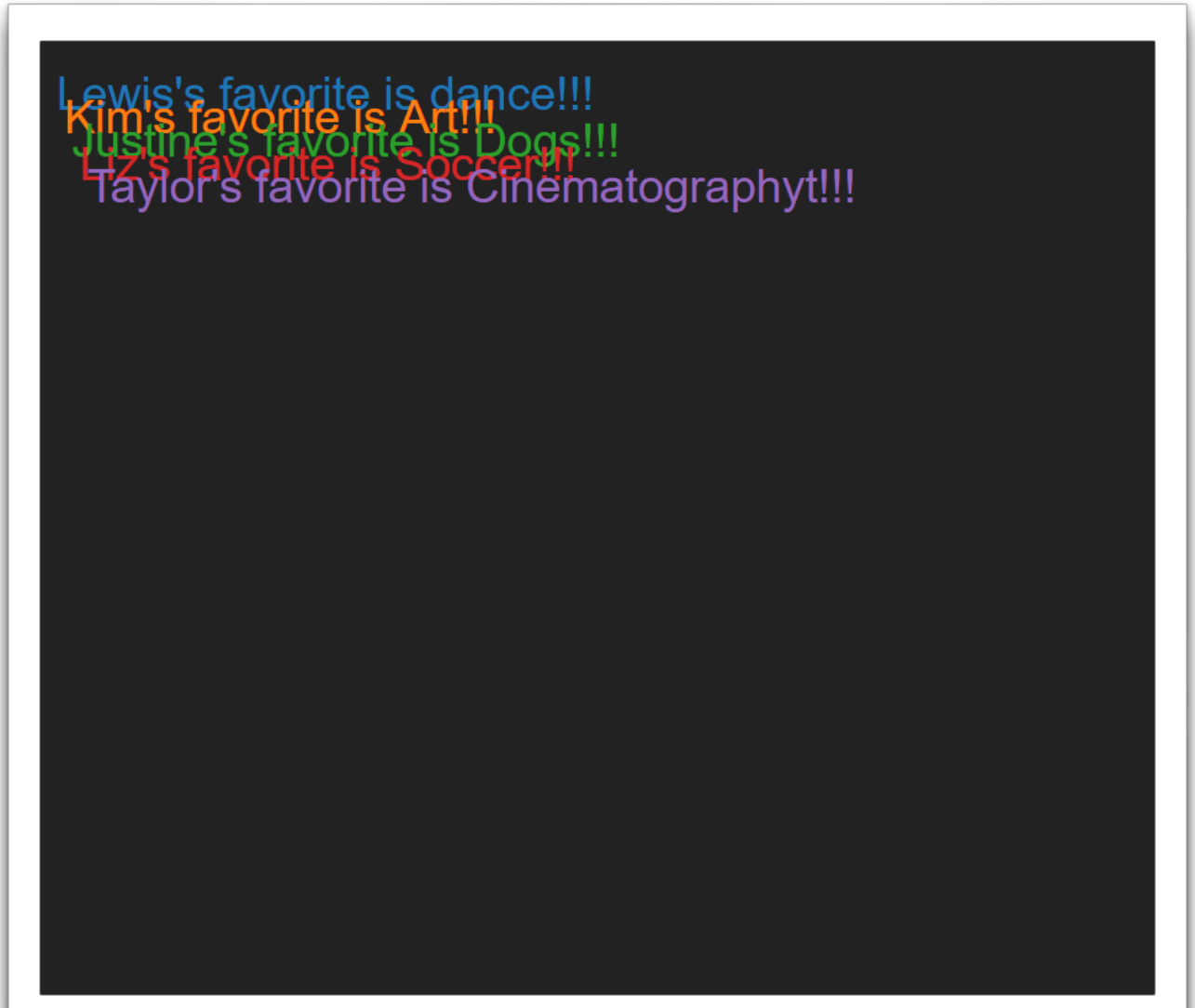
```
package storm.spout;

import backtype.storm.spout.SpoutOutputCollector;

import backtype.storm.task.TopologyContext;

import backtype.storm.topology.OutputFieldsDeclarer;

import backtype.storm.topology.base.BaseRichSpout;

import backtype.storm.tuple.Fields;

import backtype.storm.tuple.Values;

import backtype.storm.utils.Utils;


import java.util.Map;

import java.util.Random;


public class RandomSentenceSpout extends
BaseRichSpout {

  SpoutOutputCollector _collector;

  Random _rand;
```

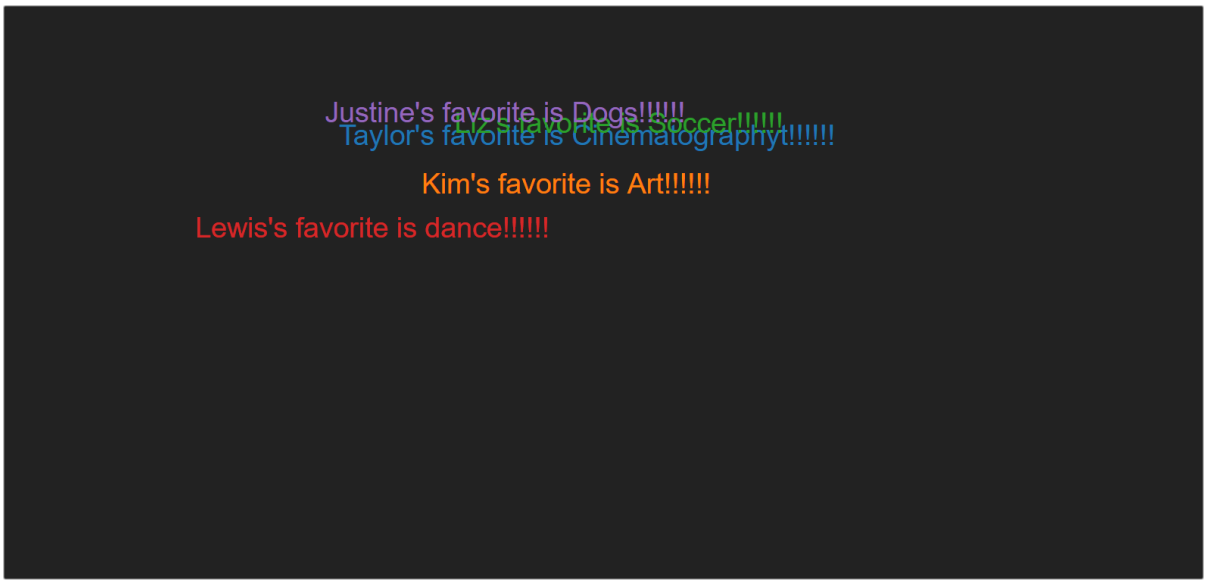
```
  @Override
  public void open(Map conf, TopologyContext context,
SpoutOutputCollector collector) {
    _collector = collector;
    _rand = new Random();
  }

  @Override
  public void nextTuple() {
    Utils.sleep(100);
    String[] sentences = new String[]{
      "the cow jumped over the moon",
      "an apple a day keeps the doctor away",
      "four score and seven years ago",
      "snow white and the seven dwarfs",
      "i am at two with nature"
      };
    String sentence =
sentences[_rand.nextInt(sentences.length)];
```

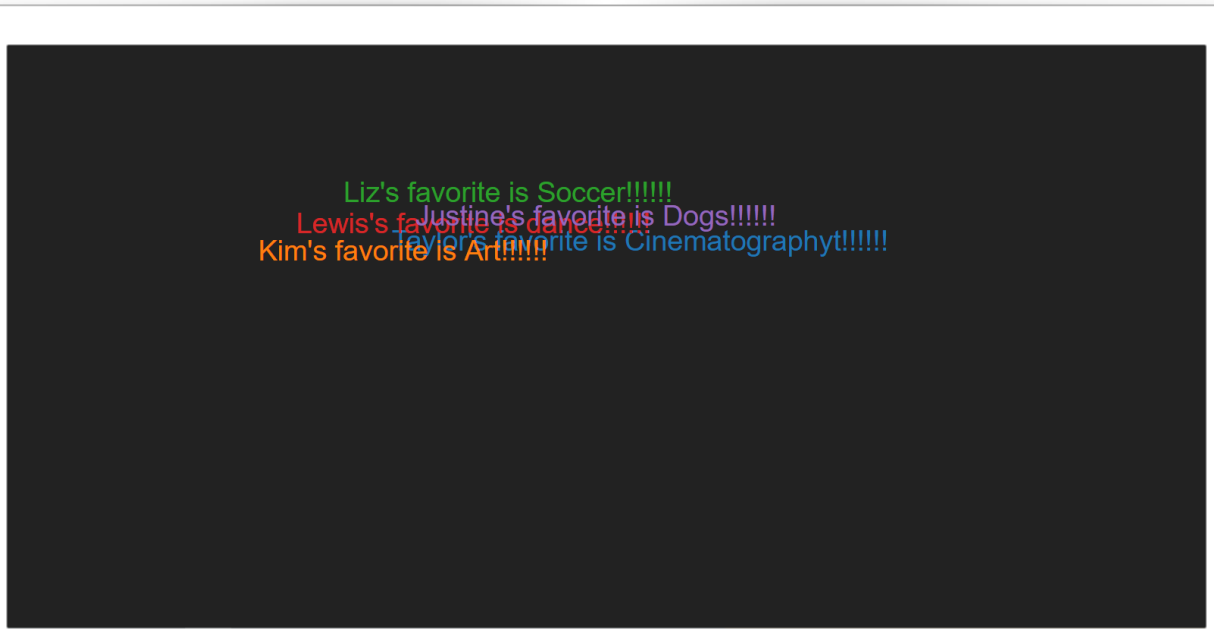
```
    _collector.emit(new Values(sentence));
  }

  @Override
  public void
declareOutputFields(OutputFieldsDeclarer declarer) {
    declarer.declare(new Fields("sentence"));
  }

}
```

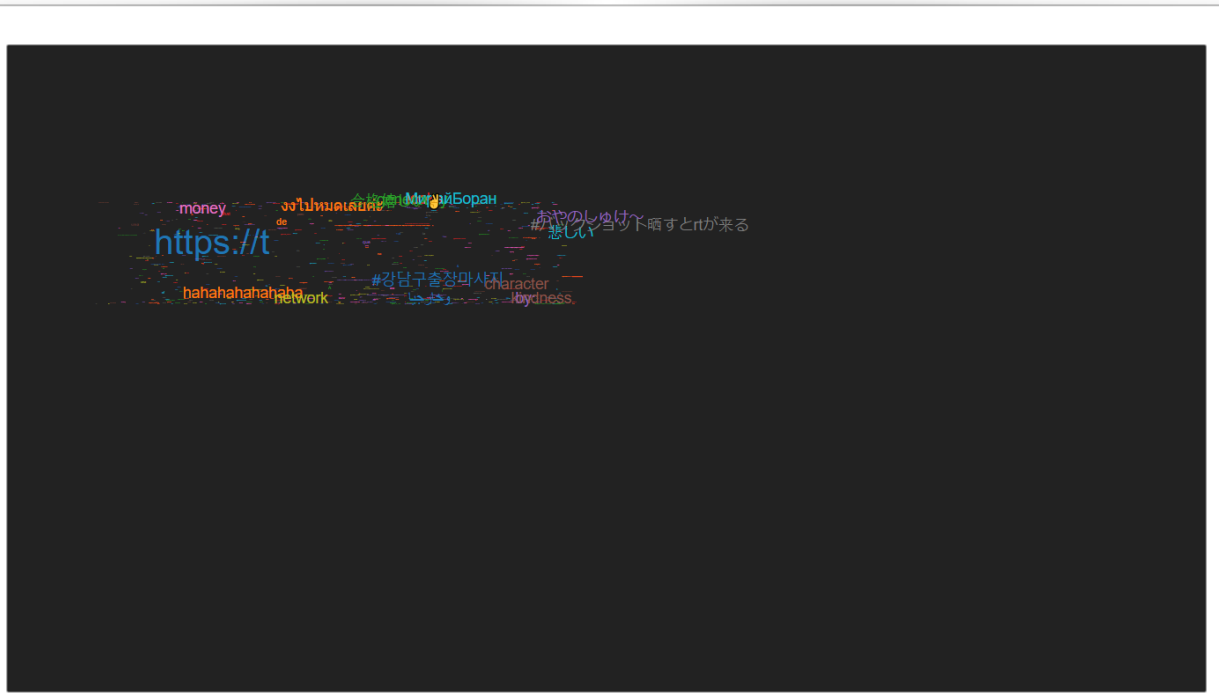
ScreenShot Of The Other Output Which We got During Various Phases.

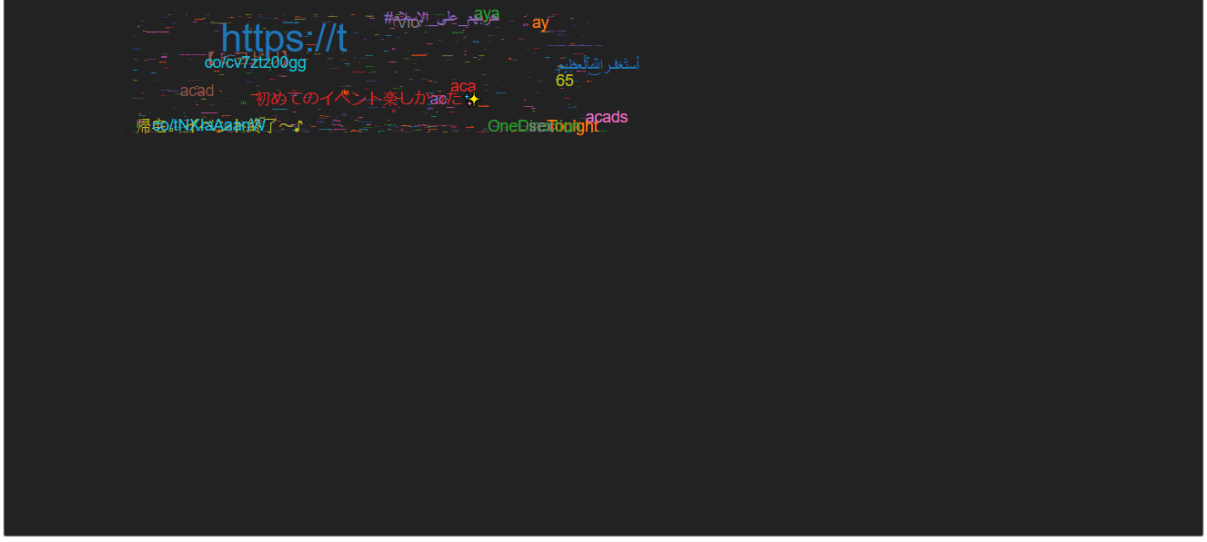

Taylor's favorite is Cinematographyt!!!
Justine's favorite is Dogs!!!
Liz's favorite is Soccer!!!
Lewis's favorite is dance!!!
Kim's favorite is Art!!!

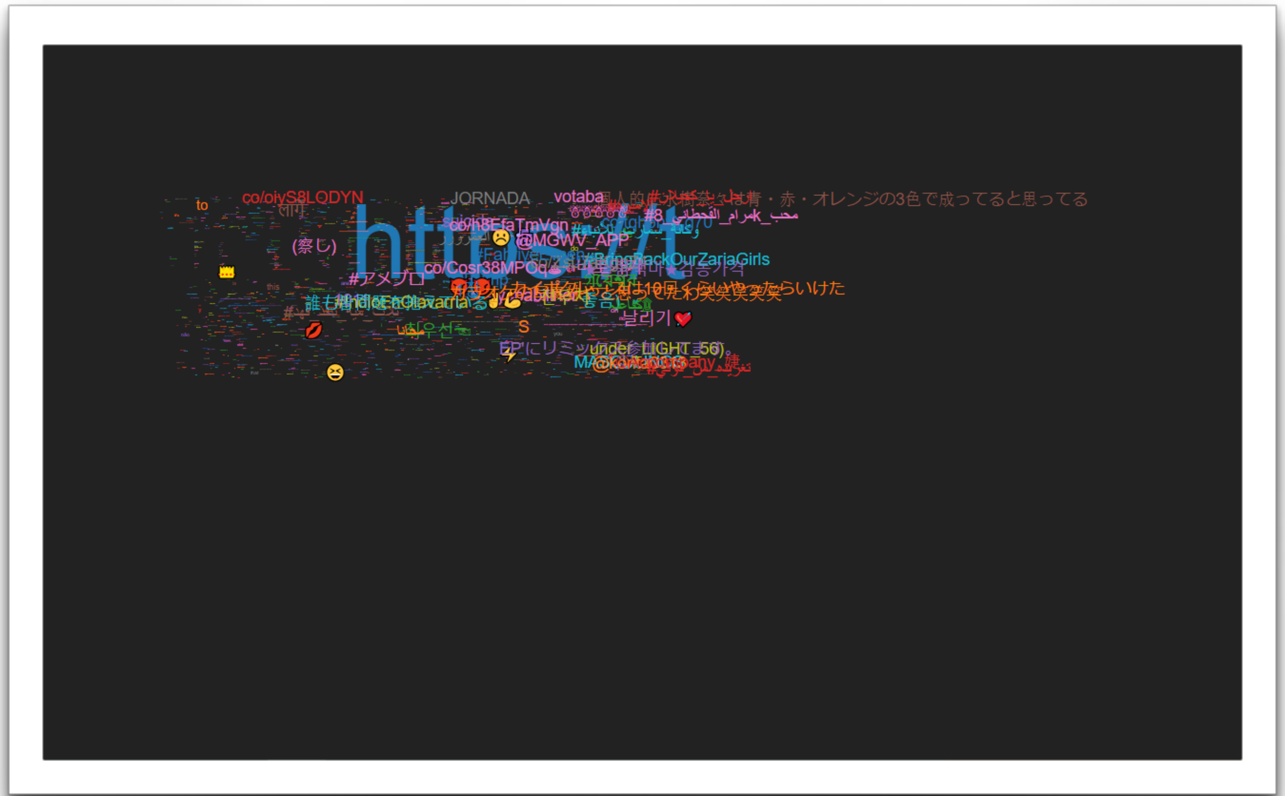

Lewis's favorite is dance!!!
Kim's favorite is Art!!!
Justine's favorite is Dogs!!!
Liz's favorite is Soccer!!!
Taylor's favorite is Cinematographyt!!!


Justine's favorite is Dogs!!!!!!
Liz's favorite is Soccer!!!!!!
Taylor's favorite is Cinematographyt!!!!!!
Kim's favorite is Art!!!!!!
Lewis's favorite is dance!!!!!!


Liz's favorite is Soccer!!!!!!
Justine's favorite is Dogs!!!!!!
Lewis's favorite is dance!!!!!!
Taylor's favorite is Cinematographyt!!!!!!
Kim's favorite is Art!!!!!!


https://t
money
งงไปหมดเลยค่ะ
de
Боран
#おやのしゃけ
悲しい
当り下晒すとrtが来る
#강남구출장마사지
character
hahahahahahaha
network
وخارجيا
boydness


https://t
#vibe
aya
ay
do/cv7ztz00gg
أستغفر الله العظيم
65
acad
aca
初めてのイベント楽しかった
OneDsexTonight
acads



co/oiyS8LQDYN
JORNADA
votaba
@MGWV_APP
#BringBackOurZariaGirls
co/Cosr38MPOq
#アメブロ
(祭し)
to
S

my
on
to
of
and
so
Dress
bottle
amarres


to
and
with
love
is
about
your
for
on
from
spicy
it
hot
are
my
any
instant
The
Tweet
chicken
what
but
Nakhua
Thai
Indian
most
more
up
de
Suresh
that
chilli
food
incroyable"
Advertisements
account.
instantly.
currency
unfold.
share
updates
you
be
на
не
и
что
11,
toko.
am
5.0
this
about.
EXILE
MUSIC
who
TRIBE
sauces
This
good
know
click
best
way
in.
by
someone
people
charges
send
they
top
latest
fastest
Following
said
curry
get
sauce
topics
AMn.
I

CONCLUSION

- Finally after a number of attempts we got the correct output.Our visualisation was showing the same tweets which were trending at that time.
- This project was really an interesting one and we have learnt a lot from it.
- Knowledge of storm has been obtained upto a significant knowledge.