CS6847 : Cloud Computing

Twitter Play - Kafka

-Premnath CS14B022
-Anvesh Bagary CS14B037
-Uday Theja CS14B044

Introduction

Twitter Play - Kafka filters the real-time tweets based on the keywords and location (Latitude, Longitude bounds) provided. Later, the filtered tweets are visualised on a Map with respect to their Geolocation.

Overview and Specifications

- **Framework :**
  - Django - python
- **APIs :**
  - Kafka - A distributed streaming platform
  - Tweepy - An easy to use python library to access Twitter API
  - Google Maps API - Map interface for location filtering

The application provides an interface for the users to input :

- **Keywords :** The multiple keywords to match for while filtering real time tweets taken as a word per line input.
- **Location :** The latitude & longitude bounds , the real time tweet must fall in, choosed through the Google Maps API by drawing a rectangle on the map.

Stream Listener is created using tweepy module which listens to all location filtered tweets. The location filtered tweets are matched for keywords and if matched are pushed to Kafka through its producer and auto creating a topic based on inputs. Kafka acts like a distributed message queue to which we can publish and subscribe records.The tweets are pushed as json object into the kafka topic with name “keyword_LocationName”. The filtered tweets from kafka are later consumed and visualised on a map dropping a marker for each

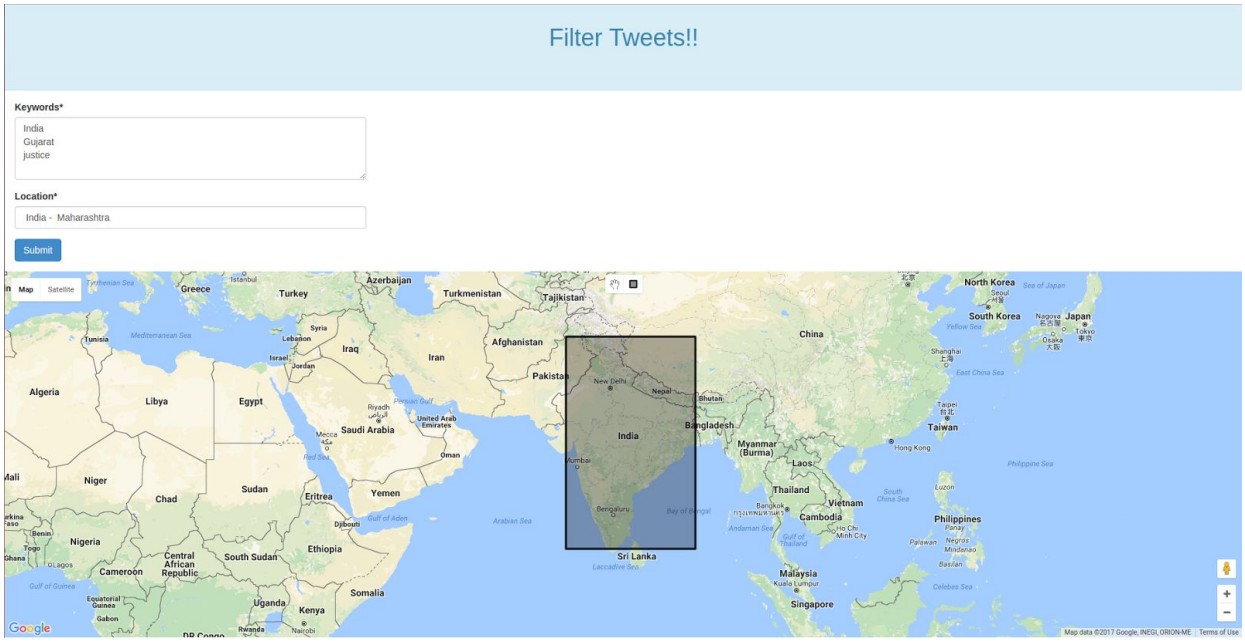
tweet and and info window displaying the tweet's content based on the source of the tweet.

Sample Testcase

Keywords = [India , Gujarat , justice]

Location = India - Maharashtra (center of the bounding box : ((8.581021215641853, 71.3671875), (34.379712580462204, 88.41796875)))

Topic = India_Gujarat_justice_India_Maharashtra



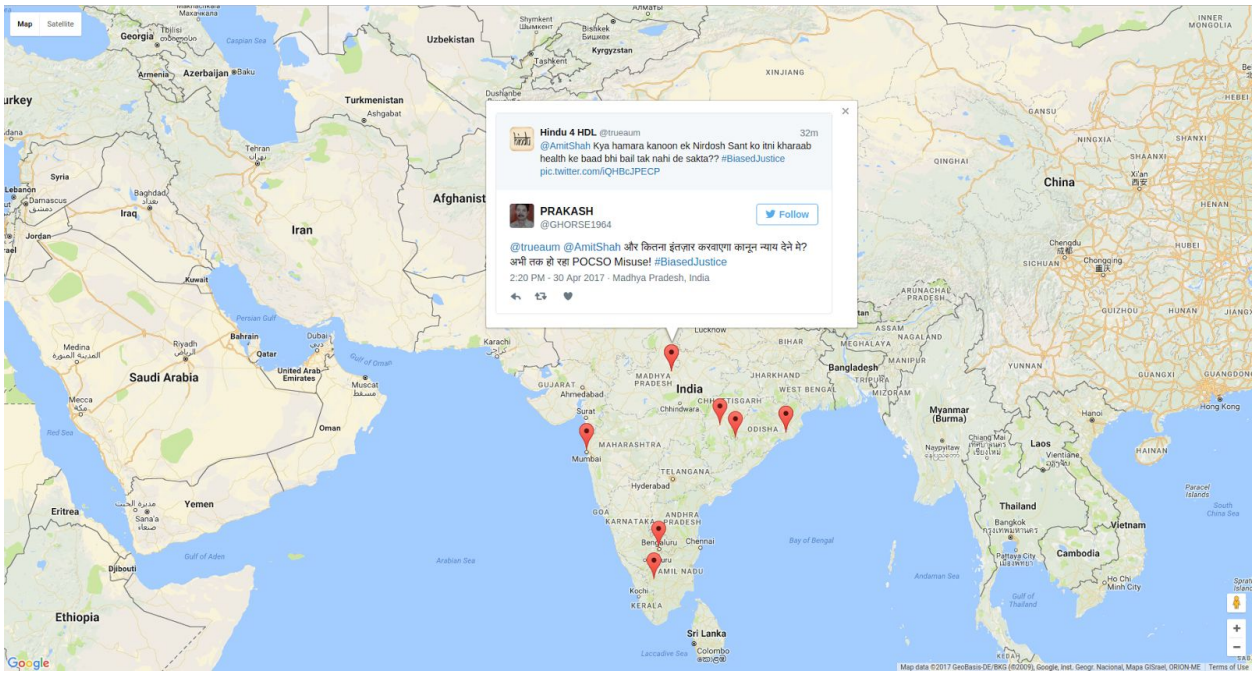
Input interface screenshot

Sample tweets filtered :

*@trueaum @dave_janak POCSO Misuse! #Biased**Justice***

*@vatsann Manmohan Desai was one of pioneers of **India**n cinema who bought commercial Bollywood with pan **India** appeal*

*this unusual message of peace, **India**'s most hostile neighbour Pakistan has fully opted out. Rest nations part of the SAARC are on-board*



Output interface screenshot

Configuration of Kafka

Kafka must be configured and running before running the web application.

Kafka Single Node and Multi Brokers

Create Kafka config files(*config/server.properties*) for each Kafka broker. In each Kafka config file, define different values for the following properties:

- **broker.id**
- **port**
- **log.dir**
- **host.name**

Later, we run all the brokers on server.

We ran 3 Kafka brokers. The following properties were defined in each broker:

Broker 0 (*server.properties*)

```
broker.id=0
port=9092
log.dir=/tmp/kafka0-logs
host.name=localhost
```

Broker 1 (*server_1.properties*)

```
broker.id=1
port=9093
log.dir=/tmp/kafka1-logs
host.name=localhost
```

Broker 2 (*server_2.properties*)

```
broker.id=2
port=9094
log.dir=/tmp/kafka2-logs
host.name=localhost
```

Later we ran the 3 brokers on the Server.

Observations:

- While producing to a list of brokers, if one of the brokers is killed, then if that broker was a leader, a new leader is elected and the messages are still published to the topics through other brokers.
- While producing to a single broker, if that broker is killed, then the messages pushed to that broker when it is down aren't lost and when the broker is alive again, those pending messages are written to the topic.

Kafka Multi Node and Multi Brokers

We set up and configured Kafka and Zookeeper on 3 different Nodes.One broker server on each node.

The internal IPs of the servers are :

```
Server 1 = 192.168.0.109
Server 2 = 192.168.0.110
Server 3 = 192.168.0.111
```

Zookeeper Configuration :

```
tickTime=2000
initLimit=10
syncLimit=5
dataDir=/tmp/zookeeper
clientPort=2181
server.1=192.168.0.109:2888:3888
server.2=192.168.0.110:2888:3888
server.3=192.168.0.111:2888:3888
```

Create the zookeeper unique identifiers on all the nodes:

```
@server1:# echo "1" > /tmp/zookeeper/myid
@server2:# echo "2" > /tmp/zookeeper/myid
@server3:# echo "3" > /tmp/zookeeper/myid
```

Configuring brokers (config/server.properties):

- **broker.id**
- **port**
- **log.dir**
- **host.name**
- **zookeeper.connect**

Node 1 - server.properties

```
broker.id=1
port=9092
log.dir=/tmp/kafka-logs
```

```
host.name=192.168.0.109
zookeeper.connect=192.168.0.109:2181,192.168.0.110:2181,192.168.0.111:2181
```

Node 2 - server.properties

```
broker.id=2
port=9092
log.dir=/tmp/kafka-logs
host.name=192.168.0.110
zookeeper.connect=192.168.0.109:2181,192.168.0.110:2181,192.168.0.111:2181
```

Node 3 - server.properties

```
broker.id=3
port=9092
log.dir=/tmp/kafka-logs
host.name=192.168.0.111
zookeeper.connect=192.168.0.109:2181,192.168.0.110:2181,192.168.0.111:2181
```

We now run the servers in all the nodes.

Experimentation on consistency and load-balancing:

Test1: replication factor : 1 partitions : 3

Enlisting Topic details where “Isr" is the set of "in-sync" replicas. This is the subset of the replicas list that is currently alive and caught-up to the leader.

```
user@user-HP-ENVY-15-Notebook-PC:~/Acads/Cloud/twitterstreamer/kafka_2.11-0.10.2
.0$ bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic test1
Topic:test1     PartitionCount:3        ReplicationFactor:1     Configs:
        Topic: test1    Partition: 0    Leader: 2       Replicas: 2     Isr: 2
        Topic: test1    Partition: 1    Leader: 3       Replicas: 3     Isr: 3
        Topic: test1    Partition: 2    Leader: 1       Replicas: 1     Isr: 1
```

On killing broker server on node 1 partition 2 gets lost forever and can’t be recovered.

```
user@user-HP-ENVY-15-Notebook-PC:~/Acads/Cloud/twitterstreamer/kafka_2.11-0.10.2
.0$ bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic test1
Topic:test1     PartitionCount:3        ReplicationFactor:1     Configs:
        Topic: test1    Partition: 0    Leader: 2       Replicas: 2     Isr: 2
        Topic: test1    Partition: 1    Leader: 3       Replicas: 3     Isr: 3
        Topic: test1    Partition: 2    Leader: -1      Replicas: 1     Isr:
```

Test2 : replication factor : 2 partitions : 3

All the partitions get evenly distributed on the three servers

```
user@user-HP-ENVY-15-Notebook-PC:~/Acads/Cloud/twitterstreamer/kafka_2.11-0.10.2
.0$ bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic test3
Topic:test3     PartitionCount:3        ReplicationFactor:2     Configs:
        Topic: test3    Partition: 0    Leader: 3       Replicas: 3,1   Isr: 3,1
        Topic: test3    Partition: 1    Leader: 2       Replicas: 1,2   Isr: 2,1
        Topic: test3    Partition: 2    Leader: 2       Replicas: 2,3   Isr: 2,3
```

On killing broker server on node 1 the partitons 0 and 1 fall out of place.

```
user@user-HP-ENVY-15-Notebook-PC:~/Acads/Cloud/twitterstreamer/kafka_2.11-0.10.2
.0$ bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic test3
Topic:test3     PartitionCount:3        ReplicationFactor:2     Configs:
        Topic: test3    Partition: 0    Leader: 3       Replicas: 3,1   Isr: 3
        Topic: test3    Partition: 1    Leader: 2       Replicas: 1,2   Isr: 2
        Topic: test3    Partition: 2    Leader: 2       Replicas: 2,3   Isr: 2,3
```

Test5 : replication factor : 2 partitions : 3

```
user@user-HP-ENVY-15-Notebook-PC:~/Acads/Cloud/twitterstreamer/kafka_2.11-0.10.2
.0$ bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic test5
Topic:test5     PartitionCount:3        ReplicationFactor:2     Configs:
        Topic: test5    Partition: 0    Leader: 3       Replicas: 3,1   Isr: 3,1
        Topic: test5    Partition: 1    Leader: 1       Replicas: 1,2   Isr: 1,2
        Topic: test5    Partition: 2    Leader: 2       Replicas: 2,3   Isr: 2,3
```

On killing broker server on node 1,leaders of partitions get reassigned to other replica present on other servers.In sync replicas also gets changed.

```
user@user-HP-ENVY-15-Notebook-PC:~/Acads/Cloud/twitterstreamer/kafka_2.11-0.10.2
.0$ bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic test5
Topic:test5     PartitionCount:3        ReplicationFactor:2     Configs:
        Topic: test5    Partition: 0    Leader: 3       Replicas: 3,1   Isr: 3
        Topic: test5    Partition: 1    Leader: 2       Replicas: 1,2   Isr: 2
        Topic: test5    Partition: 2    Leader: 2       Replicas: 2,3   Isr: 2,3
```

On reviving the server on node 1.the original configuration of the topic is restored.

On running kafka with only two broker servers (those on node 2 and node 3) the twitter stream application pushes tweets to partitions on node 2 and 3 only.This leads to major load on 2 and 3.After a couple of broker server is ran on node 1.At this point of time running twitter streamer with different keywords tends to create partition of new topics in the newly added broker server(on node 1) essentially to balance the node.This load balancing result could be realized with a better probability when zookeeper is configured on thousands of servers.