Project Report

Meetup.com Real-Time Analytics

By:
Venkatachalapathy Othisamy

Contents

Introduction

Motivation and Goal

The big data and the analytics markets have grown considerably over the last few years. In this age of data explosion, organizations are collecting and storing data at ever-increasing rates. However, simply collecting that data for your organization doesn't have any business value. Real-time analysis of this big data and visualization turn this mass of data into valuable statistics. Real-time insights must be derived from this data, to give a competitive edge to agile organizations that want to act on these insights before they lose their value. Real-time analytics is the use of, or the capacity to use, data and the related resources as soon as the data enters the system.

The advantages of real-time analytics are many. Real-time analytics help companies quickly recognize any errors in their system. This can be very helpful for systems that have a high volume of usage and store very sensitive information. Real-time analytics can help in effective marketing strategies by getting real time information about the user activities on the website.

Until recently, most of the analytics techniques and technologies would apply well to data that is collected and stored. With the emergence of big-data and real-time analytics, various tools and technologies such as Spark and Storm have been developed to facilitate real time analysis.

In our project, we wanted to dig deeper into this exciting field of real time analytics and get an insight into the technologies and the working of the real time streaming analytics. For our project, we are analyzing the RSVPs data from the Meetups website. We studied the most active cities that are responding at a particular point in time to the different meetups happening around the world. We studied about the different streaming different platforms and decided to use Spark streaming for our analysis. Apache Spark, along with Apache Storm has been the most widely used data stream processing platforms. Apache Spark, because of its in-memory processing capabilities and fault tolerance is widely preferred by many organizations.

Our methodology would also allow us to generate many other business insights related to the Meetups RSVPs. This report provides a detailed explanation of the processing of the real-time stream and the analysis performed on it.

Use Case Description

Problem context

While social media such as Facebook and Twitter is all around our lives, Meetup.com is one of the valuable social networking websites that bring people together in the real world who have common interests. To be more specific, the members are able to create groups or hold events for a particular interest. Members can RSVPs for an event that other people hold. With the website that has over 28.17 million members across over 179 countries, over 3.79 million RSVP events happen every month. We started to investigate more interesting facts about the website such as the most active locations at a point in time from which RSVPs are sent and most active topics at a point in time based on the RSVPs.

Type of data

We used streaming data from the server of Meetup.com and loaded data from Meetup Community's Long-Polling RSVP Stream. RSVP notifications are received as soon as our script finishes handling its last notification.

group

group	Group hosting the event
api_version	2
group_city	Group's home city
group_country	two-letter code of group's home country
group_id	Numeric identifier of the group
group_lat	Latitude of group's approximate location
group_lon	Longitude of group's approximate location
group_name	-
group_state	two-letter code of group's home state, if in US or CA
group_topics	Topics associated with this group
api_version	2
topic_name	Longer name
urlkey	Unique keyword
group_urlname	Unique portion of group's URL, no slashes

guests	Number of guests the member is bringing
member	Member who RSVP'd
api_version	2
member_id	Unique numeric id
member_name	Full name given
other_services	e.g. {"twitter": {"identifier": "MeetupAPI"}}
photo	Thumbnail URL for member photo if one exists
mtime	Last modified time of this RSVP, in milliseconds since the epoch
response	"yes" or "no"
rsvp_id	Unique numeric identifier
venue	Venue, if public
api_version	2
lat	Latitude of the venue
lon	Longitude of the venue
venue_id	Unique numeric identifier
venue_name	-

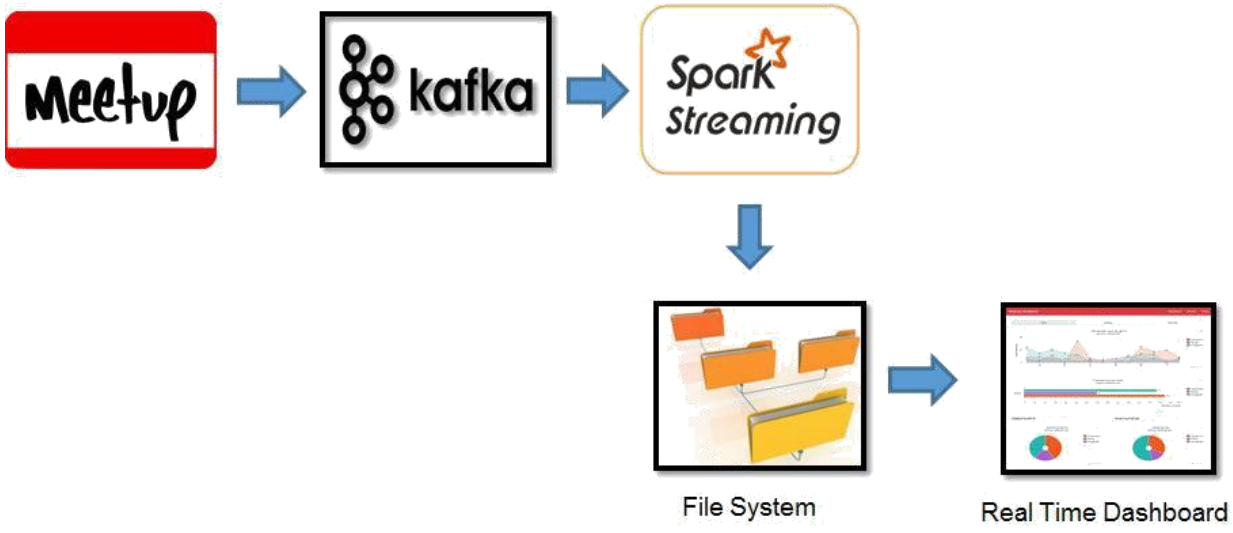
The Goal of Analysis

By answering the business questions that we've formed, including "What are the current active cities in the US?" and "What are the trending topics?". We hope to help small business to manage more effective events at the places where more meetups are hold and increase their visibility to more customers. Finally, these companies are able to seize opportunities and

partnerships with investors and generate more revenue. With our real-time analytics, the business will know the most trending topics and build more effective relationships in the networking events.

Methodology

In order to consume the Live HTTP stream of RSVPs we used Apache Kafka, a publish-subscribe messaging service. This involved building a custom Kafka Producer and Consumer. The Producer receives the data from the Meetup.com HTTP stream and sends the stream of data to topics in the Kafka cluster and the Consumer reads this stream of data from the topics in the Kafka cluster. Next, we built a spark application which configures Spark Streaming to receive data from Kafka. The stream of data received in Spark is a DStream of RDDs upon which we did our analysis. The processed data was pushed to the local file system (Since we were unable to configure Hbase to store data we resorted to store the data in the local file system). Finally, we visualized our results in near real time by plotting the results iteratively from the last written file.



Steps

Retrieving Data

Meetup.com Live HTTP Stream containing RSVPs form Meetup members was consumed using Apache Kafka. Kafka is a fast, scalable, distributed in nature by its design, partitioned and

replicated commit log service. Traditional messaging systems like ApacheMQ can also be used to retrieve the data but using Kafka over them is advantageous.

Advantages of Kafka

It is designed as a distributed system which is very easy to scale out. It offers high throughput for both publishing and subscribing.

It supports multi-subscribers and automatically balances the consumers during failure.

It persists messages on disk and thus can be used for batched consumption such as ETL, in addition to real time applications.

Kafka Architecture

Its architecture consists of the following components:

A stream of messages of a particular type is defined as a topic. A Message is defined as a payload of bytes and a Topic is a category or feed name to which messages are published.

A Producer can be anyone who can publish messages to a Topic.

The published messages are then stored at a set of servers called Brokers or Kafka Cluster.

A Consumer can subscribe to one or more Topics and consume the published Messages by pulling data from the Brokers.

Dependency on Apache Zookeeper

Apache Kafka heavily depends on Zookeeper. ZooKeeper is used for managing, coordinating Kafka clusters. Each Kafka cluster is coordinating with other Kafka clusters using ZooKeeper. Producer and consumer are notified by ZooKeeper service about the presence of new cluster in Kafka system or failure of the cluster in Kafka system. As per the notification received by the Zookeeper regarding presence or failure of the broker producer and consumer takes decision and start coordinating its work with some other broker.

Steps for Installing Apache Kafka

Below are the steps to install Apache Kafka in Cloudera QuickStart VM of Version CDH 5 and Cloudera Manager 5.

1. Navigate to a Directory like Downloads

   $cd Downloads

2. Download Apache Kafka of 0.9.1.0 version using the below command

   wget http://www-us.apache.org/dist/kafka/0.9.0.1/kafka_2.11-0.9.0.1.tgz

3. Extract the contents of the downloaded file

   tar -xvf kafka_2.11-0.9.0.1.tgz

Code Snippets

Producer.py

```
#!/usr/bin/env python
from kafka import KafkaClient, SimpleProducer
import json,requests

# Creating Kafka client
kafka = KafkaClient('localhost:9092')

#Creating a Kafka producer instance
meetup_producer = SimpleProducer(kafka)

r = requests.get("https://stream.meetup.com/2/rsvps",stream=True)

# Sending messages to Consumer.
for line in r.iter_lines():
        meetup_producer.send_messages('meetup',line)
        obj = json.loads(line.decode('utf-8'))
        rsvps= (obj['group']['group_city'])
        print (rsvps)

kafka.close()
```

Consumer.py

```
#!/usr/bin/env python

from kafka import KafkaConsumer
import json

# Subscribing a consumer to listen to topic 'meetup'
meetup_consumer = KafkaConsumer('meetup', group_id = '1', bootstrap_servers = ['localhost:9092'])

# Printing it for verification
for message in meetup_consumer:
        print (type(message.value))
```

Steps to consume the Meetup.com RSVP Stream

1. Navigate to Kafka root directory

   $cd kafka_2.11-0.9.0.1

2. Start the Zookeeper by executing the below command

   bin/zookeeper-server-start.sh config/zookeeper.properties

3. Start the Kafka Server

   bin/kafka-server-start.sh config/server.properties

4. Execute the Producer.py

   file $ ./Producer.py

5. Execute the Consumer.py file

   $ ./Consumer.py

Processing and Analyzing Data

Apache Spark

Apache Spark is a fast, in-memory data processing engine with elegant and expressive development APIs to allow data workers to efficiently execute streaming, machine learning or SQL workloads that require fast iterative access to datasets.

Advantages of using Apache Spark:

1. Faster batch processing than MapReduce. Spark executes batch-processing jobs 10 to 100 times faster than MapReduce.

2. Spark is ideal for iterative processing, interactive processing and event stream processing.

Spark Application Programming

Now we need to integrate Kafka with Spark Streaming. For this purpose we built a spark application. In this application we retrieve the streaming data from Kafka. We used the receiver based approach to accomplish this. The data is received in the form of Dstream of RDDs. We initially filter the data to contain only the cities in the US. For each of the cities, we aggregate the data by sum of occurrence. This value is updated as we get more data from the stream. We analyzed the data to know the number of cities from which Meetup.com members are responding to RSVP. After we do our analysis we store our data in the local file system as text files. Below is the code snippet of the Spark application.

Note:

By default, the Python API will decode Kafka data as UTF8 encoded strings.

Version of Spark used: 1.6

Version of Kafka used: 0.9.0.1

Code Snippets

```
from __future__ import print_function

import json
import sys

from pyspark import SparkContext
from pyspark.streaming import StreamingContext
from pyspark.streaming.kafka import KafkaUtils
```

```
f_count = 0;

# Function to aggregate the city counts to the alreaddy existing value
def city_count(newValue, oldValue):
    if oldValue is None:
        oldValue = 0

    return sum(newValue)+oldValue

if __name__ == "__main__":

    #Creaitng spark Context since we are submitting from the command line
    sc = SparkContext("local[2]",appName="MeetupStreaming")
    sc.setLogLevel("WARN")

    #Creating the streaming context
    ssc = StreamingContext(sc, 2)
    ssc.checkpoint("checkpoint")

    #Getting the zkQuorum port and topic value from command line
    zkQuorum, topic = sys.argv[1:]

    #Assigning the created stream to receive messages from kafka meetup
    sstream = KafkaUtils.createStream(ssc, zkQuorum, "1", {topic: 1})

    rsvps = sstream.map(lambda x: x[1])
    rsvps_json = rsvps.map(lambda x: json.loads(x.encode('ascii','ignore')))

    #Filtering only the messages belong to the U.S.
    us_only = rsvps_json.filter(lambda x: x['group']['group_country']=='us')

    #Extracting the city name from messages.
    city_pair = us_only.map(lambda x: (x['group']['group_city'],1))

    #Calling the function to get the city count
    city_count= city_pair.updateStateByKey(city_count)

    #Storing the output files in a local file system to get the visualization done.
    city_count.saveAsTextFiles('file:/home/cloudera/streamData/output')
    #rs.pprint()
    ssc.start()
ssc.awaitTermination()
```

Deploying the Spark Application

We need to add few components that our application is dependent on. Our application lacks SBT/Maven project management, spark-streaming-kafka-0-8_2.11 and its dependencies. This can be directly added to spark-submit command. Execute the below code in order to run the Spark Application

bin/spark-submit --packages org.apache.spark:spark-streaming-kafka-0-8_2.11:2.0.1 spark_meetup.py localhost:2181 meetup

Visualizing Data

We used PySpark, IPython and Jupyter Notebook and Python 2.6 to visualize our results. Below are the steps to install and configure PySpark, Ipython and Jupyter for Python 2.6

Steps to Install and configure IPython 1.2.1:

1. Install the IPython 1.2.1 dependencies Jinja2, pyzmq, and Tornado:

   $ pip install Jinja2

   $ pip install pyzmq

   $ pip install tornado

2. Install IPython 1.2.1:

   $ pip install 'ipython<2'

3. Set the following environment variables:

   $ export PYSPARK_DRIVER_PYTHON=ipython

   $ export PYSPARK_DRIVER_PYTHON_OPTS="notebook --NotebookApp.open_browser=False --NotebookApp.ip='*' --NotebookApp.port=8880"

Steps to Install and Configure Jupyter:

1. Install the Anaconda parcel
2. Execute the below command to install the Parcel

   bash Anaconda2-4.2.0-Linux-x86_64.sh

3. Set Environmental Variables on the driver host

   $ export PYSPARK_DRIVER_PYTHON=/opt/cloudera/parcels/Anaconda/bin/jupyter

   $ export PYSPARK_DRIVER_PYTHON_OPTS="notebook --NotebookApp.open_browser=False --NotebookApp.ip='*' --NotebookApp.port=8880"

4. Set the following variable on the driver and executor hosts

   $ export PYSPARK_PYTHON=/opt/cloudera/parcels/Anaconda/bin/python

5. To open a PySpark notebook type the below command when in the home directory

$ pyspark

In order to plot our results, we are retrieving data from the most recently created file by our Spark application and plotting it continuously in order to see the real time results

Code Snippet

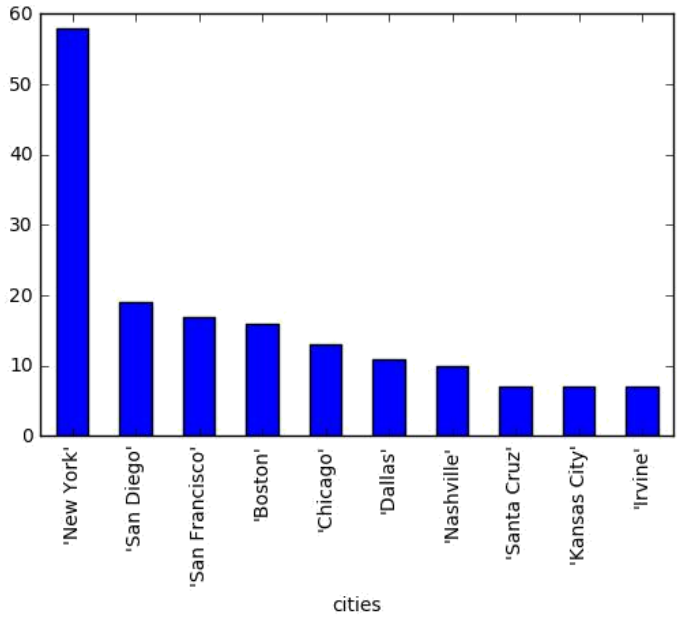
Below is the code snippet for visualizing our data.

```
import json
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)
import pandas as pd
import pandas
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
import os
import glob
mydir="/home/cloudera/streamData/output*"

#Running the program continuously
while(True):
#     mydir="file:/home/cloudera/streamData/output*"
    output_files = [file for file in glob.glob(os.path.join(mydir, 'part-*'))]
    output_files.sort(key=os.path.getmtime,reverse=True)

    #Reading the most rececntly modified file while plotting
    myrdd1 = sc.wholeTextFiles('file:'+output_files[0])
    print('file:'+output_files[0])

    #Converting to spark DataFrame
    dataDF=myrdd1.toDF()
    dataDF=dataDF.toPandas()
    dataDF.columns=["filename","cities"]

    #Filtering the empty cities
    dataDF=dataDF[dataDF['cities']!=""]
    dataDF=dataDF[dataDF['filename']!=""]
    if not dataDF.empty:
        newDF=dataDF

        if newDF.shape[0]!=0:
            location=pd.DataFrame(newDF['cities'][0].split('\n'))
```

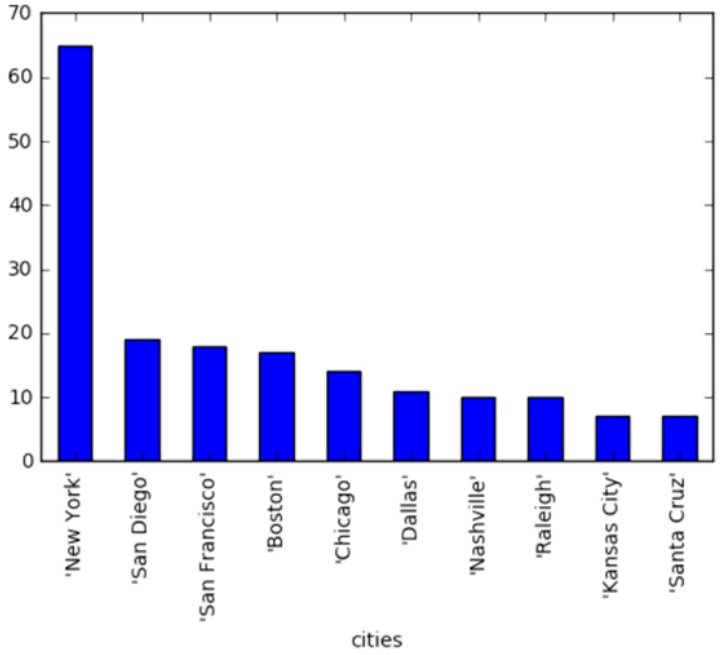
```
location.columns=['cities']
location["cities"]=location["cities"].astype(str)
splitDF=location['cities'].apply(lambda x: pd.Series(x.split(',')))
splitDF.columns=['cities','counts']
splitDF['counts']=splitDF['counts'].map(lambda x: str(x).replace(')',''))
splitDF=splitDF[splitDF["cities"]!='nan']
splitDF=splitDF[splitDF["counts"]!='nan']

splitDF['counts']=pd.to_numeric(splitDF['counts'])
splitDF['cities']=splitDF['cities'].map(lambda x: str(x).replace('(u',''))

#Sorting the city based on city counts
aggregated_df=splitDF.groupby("cities")["counts"].sum()
aggregated_df=aggregated_df.sort_values(ascending=False)
aggregated_df=aggregated_df.head(10)
aggregated_df.plot(kind='bar')
plt.show()
```

Sample Visualization

The below graphs show the top 10 city counts from which RSVPs are coming. The time difference between plotting of each curve is 2 seconds.



As you can see cities like ‘New York’ received more than 7 RSVPs within a time span of 2 seconds.



Demo

The demo of the project is available in the below link.

https://www.dropbox.com/s/d21ua6a965g7vu5/ProjectDemo.webm?dl=0

Conclusion

In conclusion, although it is more difficult to conquer the technical problems in real-time analytics, we believe that the instant information can generate higher business value for the companies. We were successful in plotting near real-time, the most active cities. We developed the structure that can answer many more questions from the meetups RSVPs.

For the future work, an interactive dashboard can be built which can be used to visualize the different results of the Spark processing. We can do this using d3.js. Right now, we are saving the results temporarily in the file system. Using HBASE to store the results after the processing would be much faster and would show the results instantly without much lag.

Ultimately, by working on this project we got a great insight into the world of streaming analytics and the difficulties associated with it. It is not just one platform that we had to study about during this project. We had to get a good understanding about the stream processing platform Kafka and the Zookeeper and the configurations that we need to make to get them up and running.

Bibliography

https://www.infoq.com/articles/apache-kafka

https://www.infoq.com/articles/apache-spark-streaming

https://www.vultr.com/docs/how-to-install-apache-kafka-on-centos-7

http://www.cloudera.com/documentation/enterprise/5-5-x/topics/spark_ipython.html

http://searchcrm.techtarget.com/definition/real-time-analytics

https://www.techopedia.com/2/31433/trends/big-data/advantages-of-real-time-analytics-for-enterprise