Twitter Data Analytics Implementation on Google Cloud

Thejus Singh Jagadish (*Author*)
Department of Computer Science and Engineering
Santa Clara University
Santa Clara, USA

Priyadarshini (*Author*)
Department of Engineering Management and Leadership
Santa Clara University
Santa Clara, USA

Shashi Shekar (*Author*)
Department of Computer Science and Engineering
Santa Clara University
Santa Clara, USA

***Abstract*—The increasing popularity of the micro-blogging sites like Twitter, which facilitates users to exchange short messages, is an impetus for data analytics tasks for varied purposes, ranging from business intelligence to national security. Twitter is being used by a large number of users for events update and sentiment expression.**

In this project paper we describe how we implemented an efficient 'Twitter Web Service' using cloud computing. This paper describes the architecture, procedure and other important cloud computing concepts used as part of implementation on Google cloud Platform. This 'Twitter Web Service' will be used to query on twitter data based on different inputs and provide corresponding response in JSON or XML format. This paper also compares Amazon Web Services with Google Cloud instances and reasons of why we chose Google Cloud platform.

***Keywords*—*A*nalytics; Tweets; Cost Efficient; Web Service; Twitter Analytics, MapReduce, Hive, Rest API**

I. INTRODUCTION

Now a days, all the companies have high volume, high velocity and high variety of the databases. The main motivation here is to build the web service for particular web application called twitter for its big data analytics. All analytical services are very expensive so future goal is to make it very fast and cost efficient. In this project, we will try to build a cost effective, cost efficient and fast web service solution utilizing Google Cloud Platform.

The main idea of this project is to build a structure that will help a user to post certain queries on twitter data with certain inputs and get the corresponding results. The Twitter Web Service built on Google cloud platform will give us an exposure to most important areas and tools of cloud computing like processing unstructured data(big data), Apache Flume , Hive and Hadoop Mapreduce, building Rest APIs and deploying them on cloud.

A Similar web service can be developed on any other platform but the reason why we chose Google and how we implemented will be discussed further in this paper.

II. IMPLEMENTATION METHODOLOGY

Below is the architectural flow of the process.

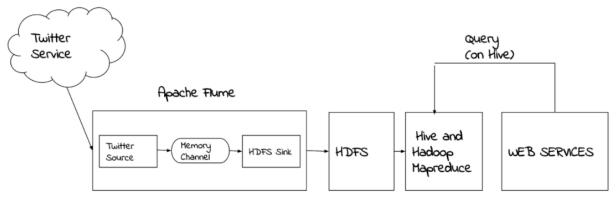


Fig 1. Architecture of implementation of Twitter Data Analytics on Google Cloud Platform

A. Features of Architecture

(i) Twitter Source : This is the data generator for our project and will be the source for getting the live stream of data by using streaming API of twitter.

(ii) Apache Flume : The tool that is used to get data from twitter and load into Hadoop Distributed File System (HDFS).
Flume is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data. It has a simple and flexible architecture based on streaming data flows. It is robust and fault tolerant with tunable reliability mechanisms and many failover and recovery mechanisms. It uses a simple extensible data model that allows for online analytic application.

(iii) Hadoop Distributed File System (HDFS): The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. In our project HDFS is the place where the data that is drawn from twitter is dumped in form of files

(iv) Hive and Hadoop MapReduce: Hive is hadoop tool that is used to perform the mapreduce operation on the large data set. The files from the HDFS is copied into the hive tables in a structured format. HiveQL is the language we are using to query and fetch data from these tables. Whenever the data is queried upon, the hive mapreduce is executed internally to fetch data from Hive with underlying Derby database.

(v) Web Services: These are the set of REST APIs which are having the queries to fetch the required data which is part of project requirement. Representational State Transfer (REST) is a style of architecture based on a set of principles that describe how networked resources are defined and addressed. We have created three REST services which helps to fetch required data which is part of project requirement. Jersey has implemented Rest architecture and provide to all of the users as jar file.

B. *Detail Description of Each Step of Implementation*

(i) Twitter Source:

Twitter source is the data generator, which generates the data, which can be collected by using the data collectors such as Apache Flume, which is running on them. In order to access the Twitter data, we need to have a Twitter account and create the Twitter application. By creating this application, we will be able to generate the keys and access tokens, which are useful later to build the configuration file of Apache Flume. At This step we ill be getting the following keys:

- Consumer key
- Consumer secret
- Access token
- Access token secret

(ii) Initial environment setup

(a) Creation of Google instance: We chose Google platform as we get a $300 worth of free transactions and this is helpful for our initial prototype and testing. The other reasons and various costs to choose this platform will be discussed further in section 3.3 of this paper.
The instances were created with 2 CPUs, 7.5 GB RAM and default Debian system.

(b) Installing necessary images to complete the setup

- Installation of Hadoop version 2.7.3
- Installation of Apache Flume version 1.7.0
- Installation of Hive
- Installation of Derby DB

(iii) Apache Flume :

Apache Flume is a tool/service ingestion mechanism for collecting, aggregating and transporting large amounts of streaming data from various web servers to a centralized data store. The reason we chose Apache Flume is because it is a highly reliable, distributed, and configurable tool that is principally designed to transfer streaming data from various sources to HDFS. Since we are handling big data in our project Flume is ideal to import huge volumes of event data produced by social networking sites like Twitter.[1]

Apache Flume Agent has three main features :

- Source: Component that receives the data from data generators and it is Twitter in our case

- Channel: Transient store which receives the events from the source and buffers them till they are consumed by sinks. In our project we are using Channel Type as File as we want to control the flow though the number of files and not based on memory.

- Sink: Stores the data into centralized stores like HDFS. In our project we are storing it in the HDFS

Some of the features of flume configuration file:

Source, channel and sink details:

```
TwitterAgent.sources = Twitter
TwitterAgent.channels = FileChannel
TwitterAgent.sinks = HDFS
```

Tokens and access keys received from Twitter :

```
TwitterAgent.sources.Twitter.consumerKey =
SqWu0uWAtrg6p6ROqAptbJlva
TwitterAgent.sources.Twitter.consumerSecret =
bPtncFTPBUwvbzLnW5yv0VmtMMPkcnst4r8HrAkZwxG5u
uB1xo
TwitterAgent.sources.Twitter.accessToken = 2235585415-
RLCVufXlDhzcXsAe4DknXnmoQV7ihlXpvB5Tevk
```

TwitterAgent.sources.Twitter.accessTokenSecret = XFbV51FS0BtkBXe7jOA2qe8Sd8kirNTkur9hwb5JZdqCy
Twitter batchsize:
TwitterAgent.sources.Twitter.maxBatchSize = 50000
TwitterAgent.sources.Twitter.maxBatchDurationMillis = 100000

By executing the flume commands, the data from twitter is loaded and stored in HDFS file system in form of files. Each file is around 126MB.

(iv)Hadoop Distributed File System (HDFS):

HDFS is a Java-based file system that provides scalable and reliable data storage, and it was designed to span large clusters of commodity servers. The Hadoop Distributed File System (HDFS) is the primary storage system used by Hadoop applications. [2]

This HDFS is very apt and efficient for our project because of its high performance access to data across the Hadoop clusters and its highly fault tolerant. It also allows operators to bring back the previous version of HDFS after an upgrade or in case of human or systematic error.

Processing tasks can occur on the physical node where the data resides, which significantly reduces network I/O and provides very high aggregate bandwidth.

HDFS will be a temporary storage to store the files from Twitter source. After the data is copied into the tables, the file system is flushed out and gets new data.

(v) Hive and MapReduce:

From the HDFS the data has to be moved to structured schema. For this we have used NoSQL Derby Database and Hive tool is used to run queries on the underlying database.

For moving data from HDFS File System to a database schema, we have used Avro. Avro is a remote procedure call and data serialization framework developed within Apache's Hadoop project. The reason for using Avro in our project is that Avro stores the data definition in JSON format making it easy to read and interpret and the data itself is stored in binary format making it compact and efficient.

Avro files also include markers that can be used to splitting large data sets into subsets suitable for MapReduce processing.[3] It helps in structuring the data well from HDFS and stores on the hive tables by handling the null and garbage values.

Configuration of Avro:
The schema structure is as below when data is loaded into the Hive tables

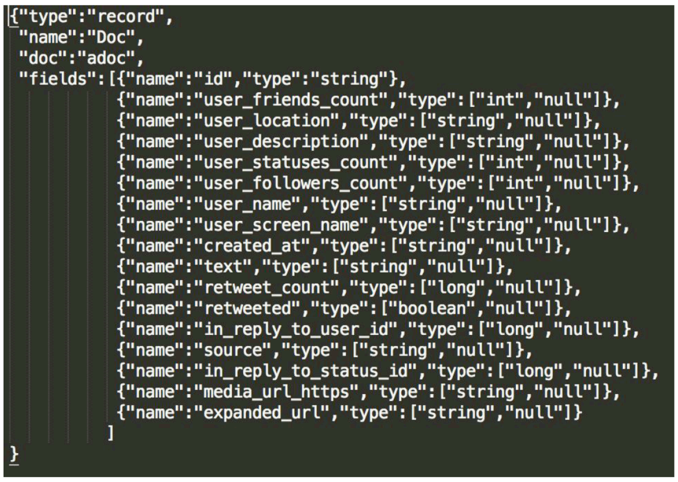

```
{"type":"record",
 "name":"Doc",
 "doc":"adoc",
 "fields":[{"name":"id","type":"string"},
           {"name":"user_friends_count","type":["int","null"]},
           {"name":"user_location","type":["string","null"]},
           {"name":"user_description","type":["string","null"]},
           {"name":"user_statuses_count","type":["int","null"]},
           {"name":"user_followers_count","type":["int","null"]},
           {"name":"user_name","type":["string","null"]},
           {"name":"user_screen_name","type":["string","null"]},
           {"name":"created_at","type":["string","null"]},
           {"name":"text","type":["string","null"]},
           {"name":"retweet_count","type":["long","null"]},
           {"name":"retweeted","type":["boolean","null"]},
           {"name":"in_reply_to_user_id","type":["long","null"]},
           {"name":"source","type":["string","null"]},
           {"name":"in_reply_to_status_id","type":["long","null"]},
           {"name":"media_url_https","type":["string","null"]},
           {"name":"expanded_url","type":["string","null"]}
          ]
}
```

Fig 2. Schema of tweet data

Hive :

Hive architectural features are very favorable for implementation of our project.

Some of the components of Hive architecture are:

(i) Metastore: Stores the metadata for each of the tables such as schemas and location.

(ii) Driver: Acts like the controller, which receives the Hive queries from web services in form of HiveQL statements. The queries are executed by creating the sessions and monitor the life cycle and program of execution.

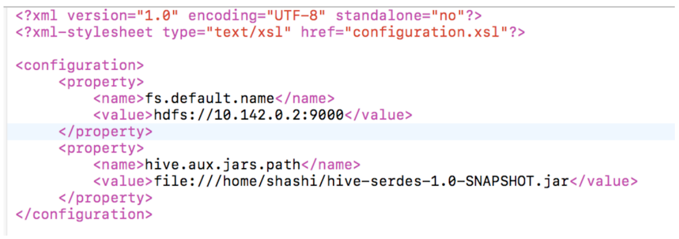
(iii) Compiler: Performs the compilation of the hive query to convert it into execution plan. These plans contain the tasks and steps needed to be performed by Hadoop MapReduce.

(iv) Optimizer: Performs various transformation on the execution plan to get the optimized Directed Acyclic Graph (DAG).

(v) Executor: Executes plan according to the DAG, which is generated by optimizer. Then it interacts with the job tracker of hadoop to schedule task to be run. It also supports external tables, which make it possible to process data without actually storing in HDFS, and also supports partitioning of data at the level of tables to improve performance.

Two files are important for configuring hive in our project
(i) hive-env.sh: Script file used for setting up environment variables

(ii) hive-site.xml: set up hadoop name nodes



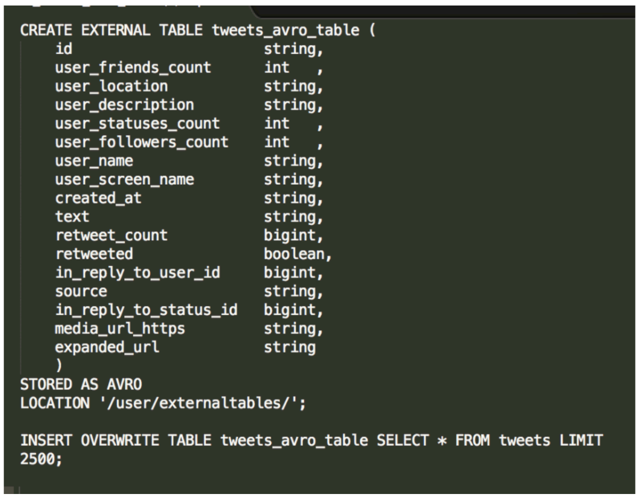
```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
     <property>
         <name>fs.default.name</name>
         <value>hdfs://10.142.0.2:9000</value>
     </property>
     <property>
         <name>hive.aux.jars.path</name>
         <value>file:///home/shashi/hive-serdes-1.0-SNAPSHOT.jar</value>
     </property>
</configuration>
```

Fig 3. Hive xml file to setup hadoop name nodes

MapReduce :

MapReduce is invoked on the hive table only when the query is made on certain columns. In the first case where we are getting data transferred from HDFS to Hive tables, we are just using a 'LOAD' command to load data. The Avro internally is converting the files into the rows and columns based on schema provided . In Order to have a structured data and invoke MapReduce we need to make queries on certain columns. Creation of below table tweets_avro_table invokes the MapReduce job.



```sql
CREATE EXTERNAL TABLE tweets_avro_table (
    id                      string,
    user_friends_count      int    ,
    user_location           string,
    user_description        string,
    user_statuses_count     int    ,
    user_followers_count    int    ,
    user_name               string,
    user_screen_name        string,
    created_at              string,
    text                    string,
    retweet_count           bigint,
    retweeted               boolean,
    in_reply_to_user_id     bigint,
    source                  string,
    in_reply_to_status_id   bigint,
    media_url_https         string,
    expanded_url            string
    )
STORED AS AVRO
LOCATION '/user/externaltables/';

INSERT OVERWRITE TABLE tweets_avro_table SELECT * FROM tweets LIMIT
2500;
```

Fig 4. Structure of final hive table

Apart from this MapReduce is also invoked whenever there is a query made on this above created table that is whenever the webservices are called. These web services have HiveQL queries and query on specific columns of tweets_avro_table to get the desired outputs.

(vi)Web Services:

Web Services are basically the Rest APIs which have hive queries. The services will connect to the Hive and underlying Derby DB to establish the connection and then execute the queries to get the desired output.

In this project as part of requirement we have developed 3 REST APIs each of which will have a different query. These are executed by taking inputs which internally are handled by using a POST method.
Every resource is uniquely addressable using a uniform and minimal set of commands. The protocol is client/server, stateless, layered, and supports caching.

Each of the query takes the input in the URI and result is given in form of XML format. Below are the query details :

Query 1 : All the tweets at a particular time instance
Input : Timestamp
Output : UserId, Tweet text

Query 2 : Total number of tweets sent by range of Users
Input : UserId
Output : UserId, number of tweets for user

Query 3 : Set of Users who have retweeted a tweet posted by given userid
Input : UserId
Output : UserId, Retweet Count

C. Reasons for Developing on Google Platform over Amazon

(i) Google's on-demand, real-time pricing is simple and straightforward, with discounts for sustained usage, and a stated commitment to pass future price reductions on to customers. With initial $300 worth of free features, it is very ideal for prototyping the project of data analytics.

(ii) Works easily in terms of agility and flexibility

(iii) Google cloud has taken the "Pay per use" concept to the extreme. GCE bills in minute level increments. Amazon EMR charges by the hour of use, rounded up to the nearest whole hour, If running a 2-hour workload and the job execution exceeds the 2-hour mark by even a minute, EMR will charge for 3 hours of work. But Google Cloud Dataproc round usage to the nearest minute, reducing the overall cost of the job and also significantly simplifying the application lifecycle management for customers.

(iv) Load Balancers in GCE don't need any pre warming. Amazon elastic load balance does not seem as elastic as it seems as it's not meant to handle large number of requests that is reason why prewarming is required to have minimum scaling.

(v) With respect to the management of Hadoop environment with automatic provisioning and configuration, simple job management, sophisticated monitoring and flexible the number of instance types supported by Google Cloud is in thousands when compared to Amazon EMT+R which is only 37.

(vi) Costs of creating instances on Google is comparatively lesser than that of Amazon EMR. Below table gives an idea about the prices based on some data assumptions.

Features	Google Cloud Instances	Amazon EMR
Data size for a month	50TB	50 TB
Cluster containing : 20 worker nodes with each node having 4 cores and 15 GBs of RAM	Equivalent to : 21-node cluster (1 master and 20 worker nodes) with n1-standard-4 instance type that has 4 virtual cores per node, 15 GBs of RAM and 80GB of SSD disk	Equivalent to : 21-node EMR cluster (1 head node and 20 workers) of m3.xlarge instance type that has 4 cores 15GB RAM and 2x40 GB locally attached SSD storage.
Monthly Pricing	$2154.40	$2613.27

TABLE I. Cost comparisons of data sample on Amazon EMR and Google Cloud

D. *Future Work and Enhancements*

(i) We have used only two instances as part of project prototyping, in future when this project requires to go live, we can increase number of nodes and scale it .

(ii) Load balancing can be implemented by creating cluster of instances to extend this project. The type of load balancing would be based on HTTP.

(iii) As part of requirement we have used simple queries to get the desired output. In future if there is a requirement for using complex queries, the REST APIs can be modified to write complex queries.

(iv) This model is easily expandable to add more web services based on requirement. This only means that we need to add more queries in the web services with the base structure unchanged.

(v) Since there is no static data dump, the query will always fetch different results based on real time data. Hence the implementation is not limited to static data which makes it easily expandable in future.

III. Summary

In this paper we have described the details on how the Twitter data Analytics data was implemented. As part of lessons learnt we had a hands on experience on Google Cloud Platform, Apache Hadoop, Flume, Avro, HDFS, Hive, MapReduce and Rest APIs. We also explored advantages of using Google platform cost wise compared to other cloud applications especially Amazon EMR.

IV. References

[1] Apache_Flume https://www.tutorialspoint.com/apache_flume/index.ht m

[2] Hadoop_Distributed_File_System https://hortonworks.com/apache/hdfs/

[3] Apache_Avro https://en.wikipedia.org/wiki/Apache_Avro

[4] https://cloud.google.com/blog/big-data/2016/03/underst anding-cloud-pricing-part-6-big-data-processing-engine

[5] https://hadoop.apache.org/docs/r2.7.3/

[6] http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-multi-node-cluster/

[7] Restful_API http://www.drdobbs.com/web-development/restful-web

[8] https://hadoop.apache.org/docs/stable/hadoop-yarn/had oop-yarn-common/yarn-default.xml

[9] https://www.linkedin.com/pulse/how-conduct-twitter-h ashtag-data-analytics-using-hadoop-mbabela

[10] https://cwiki.apache.org/confluence/display/Hive/Langu ageManual+DML

[11] https://cwiki.apache.org/confluence/display/Hive/Langu ageManual+DDL

[12] https://cwiki.apache.org/confluence/display/Hive/Tutorial

Github link : https://github.com/shashi8629/CloudProject