Citi Bike share analysis

https://gitlab.com/cloudmesh_fall2016/project-061.git

Aruna Kumaraswamy
F16-DG-4034
Indiana University
akumaras@iu.edu

ABSTRACT

Purpose of this project is to analyze the NYC Citi Bike share system data in batch and streaming mode. Following report summarizes the finding based on the September 2016 bike share data. Report will also provide instructions on installing the necessary software, build and deploy the software created for this project.

This is a draft version. Anticipated completion date is December 9, 2016.

1. INTRODUCTION

Apache Flink is used in this project to process the data in batch and streaming mode. There are 3 main components in this project

- BikeRentTrendByAge
- BikeRentTrendByGender
- LiveStationStatusStream

Components are built in Java. Maven is used for building the components.

1.1 Bike Rent Trend - By Age

Objective of this analysis is to find in which age group bike share is popular. This component takes csv file as input. For age group analysis, we are interested only in the age, so we skip all the other columns in the spreadsheet. Based on the age group, we decide if the bike rider is Boomer I, Boomer II, Gen X, Gen Y or Gen Z. Invalid age in the data is filtered. Final result is the count of the age groups for September 2016. Figure 1 depicts the visualization for this batch program. Visualization is created using a python program.

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

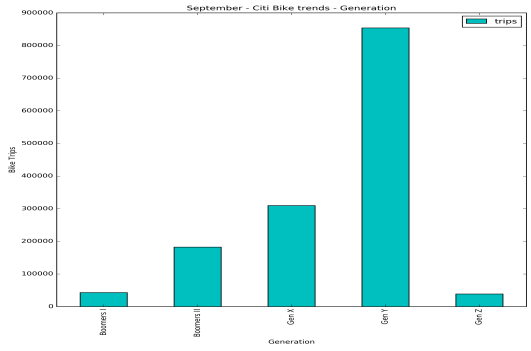

Figure 1: Trend By age

1.2 Bike Rent Trend - By Gender

Objective of this analysis is to find whether bike ride is popular among men or women. This component takes CSV file as input and creates a CSV output. We are interested in trip date, gender column of the spreadsheet, so we skip other columns. We also filter unknown gender. We transform the input data using tables and join them to produce the desired CSV output. Figure 2 is the visualization for this batch program. Visualization is created using python program.

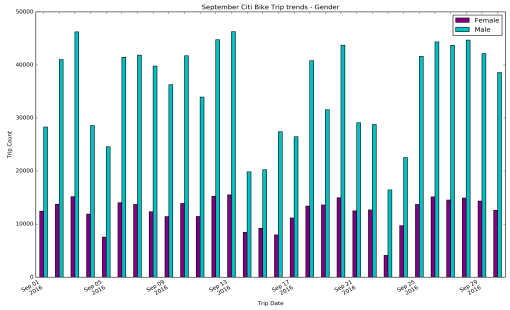

Figure 2: Trend By gender

1.3 Live Station Status Stream

Live station status feed is used to check the availability of bikes across all stations. If the bike availability of a station is

less than 2 then those stations are filtered. Result is pushed to elastic search. Kibana is used for visualization of real-time alerts across geographic map. Figure 3 is a representation on Geo map done via Kibana.

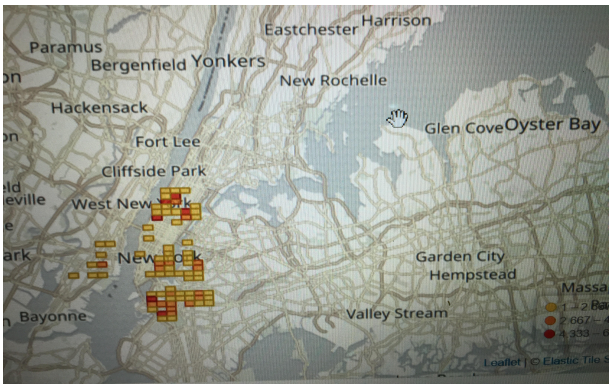

Figure 3: Bike alerts

2. TECHNOLOGY

2.1 Apache Flink

When I wrote the initial proposal for this project, I was planning to use Spark. I was interested in deploying a lambda architecture using spark and kafka. More I read about flink, it became obvious that flink is ideal for both batch and streaming data. Even though sparks supports streaming, it is usually a micro batch.

[2] Figure 4 and Figure 5 shows the comparison between spark and flink.

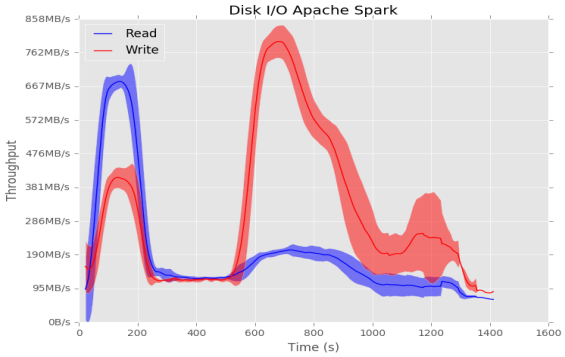

Figure 4: Spark Vs Flink Disk I/O

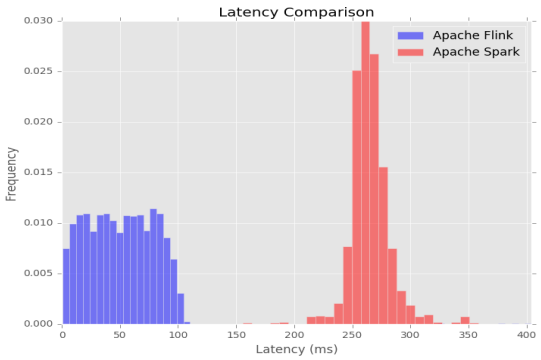

Figure 5: Latency comparison spark vs flink

2.2 Elastic search and Kibana

In this project I am using elastic search and Kibana for streaming visualization. It is based on the training on flink by data artisan [1]. Elastic search indexes are created via curl or Kibana appsense. LiveStationStatusStream component fetches data from bike share realtime feed. It can be modified to continuously fetch real time data. Current implementation is to show the behaviour of data streaming via real time feed. This data is cleansed, filtered and sent to Elastic search sink specifying the index we created via curl. Kibana can now be used to setup indexes, time characteric and search to display the bike alert data in geo map. Kibana does not allow download of the visualization, hence the of kibana is submitted in this report.

3. INSTALLATION

Flink - Install flink version 1.1.3
Elastic search - Install version 2.3.5
Kibana - 4.5.4 (with appsense plugin) Python 2.7

4. BUILD AND DEPLOY

Clone GIT repository

4.1 Step1 Build Java code

Build java code using mvn clean package

4.2 Step2 Prepare data

Use wget or bitsadmin to get trips data from https://s3.amazonaws.com/tripdata/201609-citibike-tripdata.zip

Flink is unable to process ZIP directly. So unzip the data file.

4.3 Step 3 Elastic Search Index

Start Kibana Go to http://localhost:5601/appsense Create the citi-bikes index (Refer data instruction in GIT) Start Elastic Search

4.4 Step 4 Run Flink

Refer to README.rst in GIT for detailed instructions on how to run batch and stream processing components in Flink.

5. CONCLUSION

Apache Flink offered ease of use and better performance to perform analysis in batch environment. SQL processing in batch mode simplifies data preparation and storage.
Stream Execution environment has rich set of features and connector. Even though Table API is same for batch and stream, I had difficulty getting Kafka stream to dump to a table sink.
Flink does not have a REST API similar to Spark to submit jobs, hence creating a web dashboard would take more effort.

6. REFERENCES

[1] D. artisan. Apache flink training, 2016.

[2] K. Jacobs. Apache flink: The next distributed data processing revolution?, 2016.