# Path Finder: A cycling data analytic tool using Apache Hive

MIDS W205  Final Project Report, Fall 2015


Lei Yang, Nilesh Bhoyar, Tuhin Mahmud

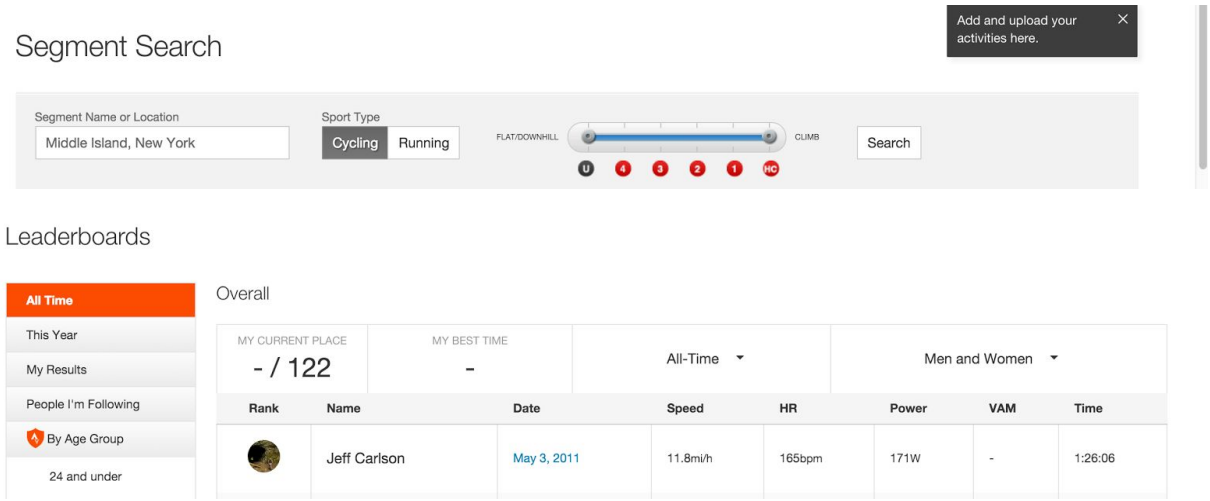
December 14, 2015

# Abstract

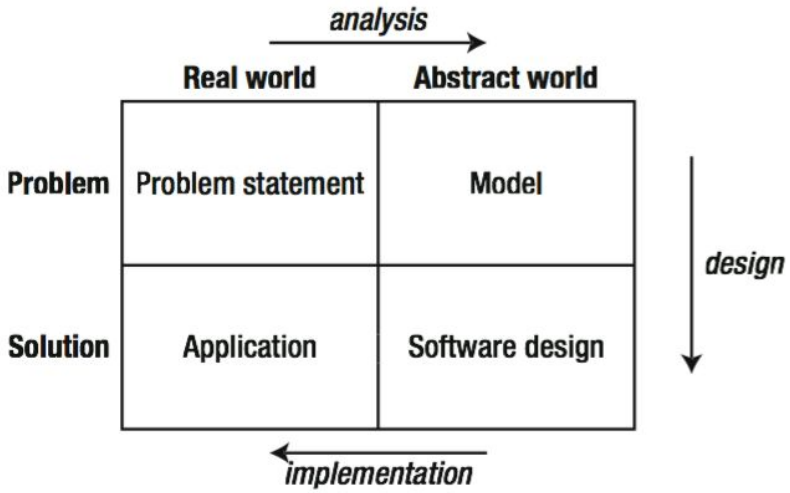
Strava is a community of athletes from all over the world. Strava lets you experience what we call social fitness - connecting and competing with each other via mobile and online apps. Riders compete with each other by climbing on segment leaderboard. With PathFinder We are trying to address complex use cases which are important and not address by current available functionality. For a new user, it is impossible to search for interesting segment by categories such as activity type, distance or weather. Leaderboard shows the time taken for completion but it does not display weather conditions especially wind direction, which affects the completion time. If it had snowed at some location yesterday it is quite possible that segment is not in ideal condition though current weather conditions are good. We tried to find solution to such uses cases by integrating freely available strava and weather data.

# Introduction

Fresh graduate from NYU. Kevin is consultant by profession but athlete by passion. He is frequent traveller. Wherever he goes he makes sure his running shoes are with him. He has started using strava to monitor his activities recently. Due to his travel, he is in constant exploration mode for new places to run.  Strava gives list of segments at particular place but it is quite cumbersome to search for segment with particular criteria.



# Overall design




Since the inception of project we were clear about use cases but just to make sure we are really designing our database that can cater any future analytics requirement , we used above procedure in our software design. We captured our uses cases and revised them multiple times using UML diagrams. With that exercise we came to know about our entities and their relationships.

## Strava data

We have 4 entities,

- Activity- This entity stands for the activities by athlete.
- Segment: This entity stores the meta information about particular segment.
- Streams- Entity stores the geo spatial data of particular segment.
- Leaderboard: stores the leader by segment.

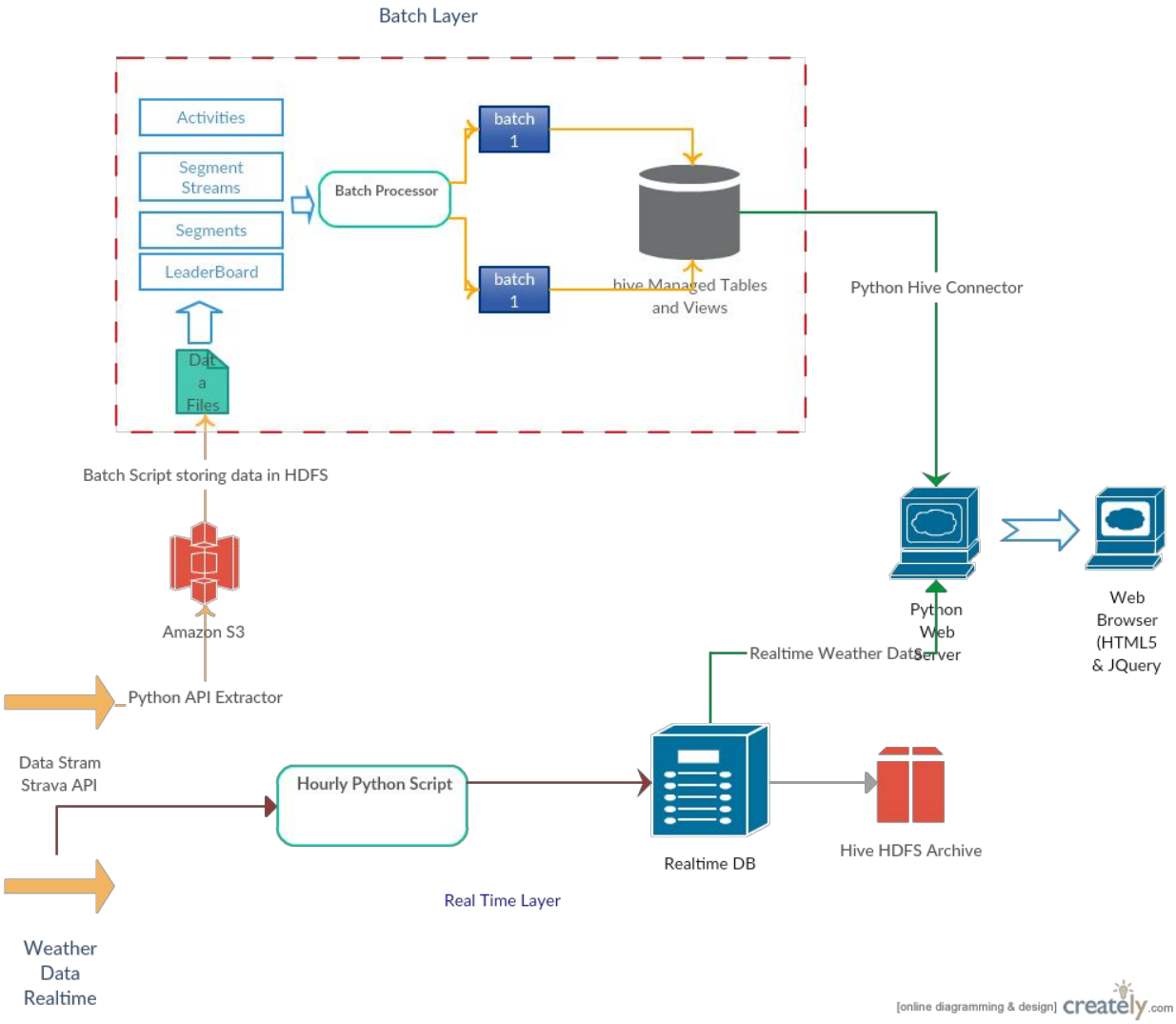
We could not get personal data of individual athlete as we mostly worked on freely available information. So we couldn't include that entity. Unlike relational database design, which gives emphasize on normalization[1] we followed the normalization techniques for big databases such as Hadoop/Hive.[2]

## Weather data

Weather data organizes all important weather indicators by city. We need near real-time data as well as historical data about weather conditions for particular segment.

## Lambda architecture

Our approach to design these data requirements i.e. combining real time as well as batch data requirements into one unit; we used engine known as Lambda Architecture, introduced by Nathan Marz [3]
Architecture consist of batch layer, that stores the historical data and speed layer which process the near real time data, and serving layer which can be required to build visuals. Batch layer can also be called as Data lake, as we dump Huge Strava data every day in that database.




## Data ingestion layer

Python scripts extracts the data daily using Strava API. We are using Stravalib library for calling Strava API. [4]. Every Day script extracts data for 4 US regions, we have selected due to API

limitations and dumps it as 4 CSV files in AMAZON S3 storage. Linux Bash script loads them into hive externally managed tables. We build this staging areas after looking at data quality. Data needs to be transformed before sending it to Managed tables from where they can be queried for further analytics. We found below issues during transformations:

1. City name were: New York /NY
2. Sometimes fields were blank
3. Distance came with measure i.e. 135M vs 120KM . We fixed them so that values can be sorted.

Python script extracts weather data hourly using API [5]. Data is directly loaded to PostgreSQL table from where it can be efficiently queried. Before we extract new data, existing data is moved to Hive tables, from where data can be further analyzed. This step is important considering querying data from PostgresQL slows down if we maintain large Volume in that table.

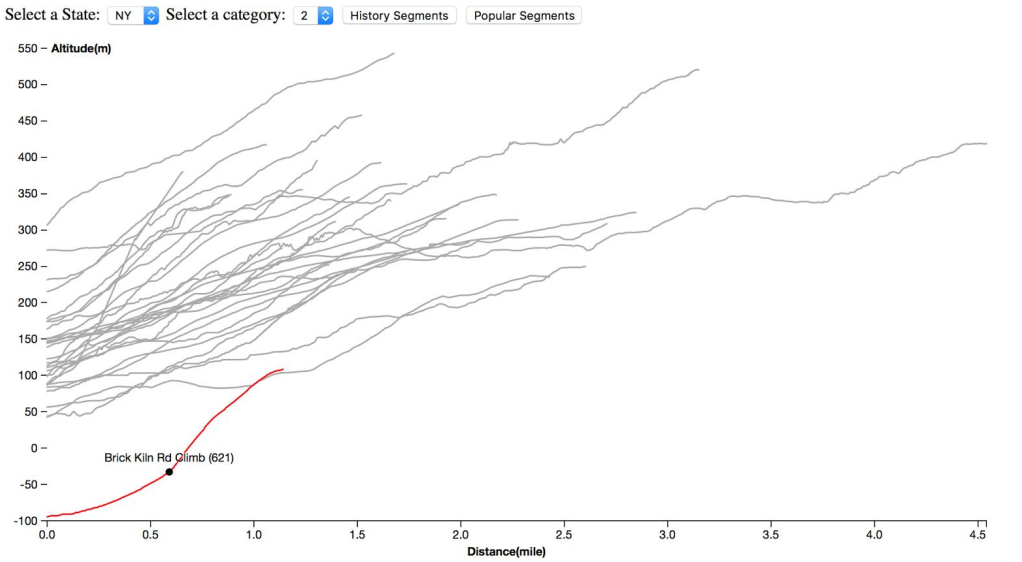
## Data processing layer

A CRON job (job.py) extracts the most popular segment for each state and stores them in a PostgreSQL database. These tables can be queried using pycopg2 library. We tried to query segment geo spatial data from hive directly for real time update but found it very slow. So we built JSON file for each segment from geo-map visualization.

## Data serving layer

We have two use cases for visuals, first segment search webpage with city and category criteria, and google maps showing segment route. The HTML will send http request via AJAX to a Python CGI server on AWS host. Two python scripts, HQL_SELECT.py and SQL_SELECT.py, are create to accept SELECT statements from the request. And they will query Hive (via pyhs2) and Postgres (via pycopg2) respectively, and return the result in JSON format. Upon receiving the query results, javascript on client browser will parse the data locally to visualize.

## Segment Search

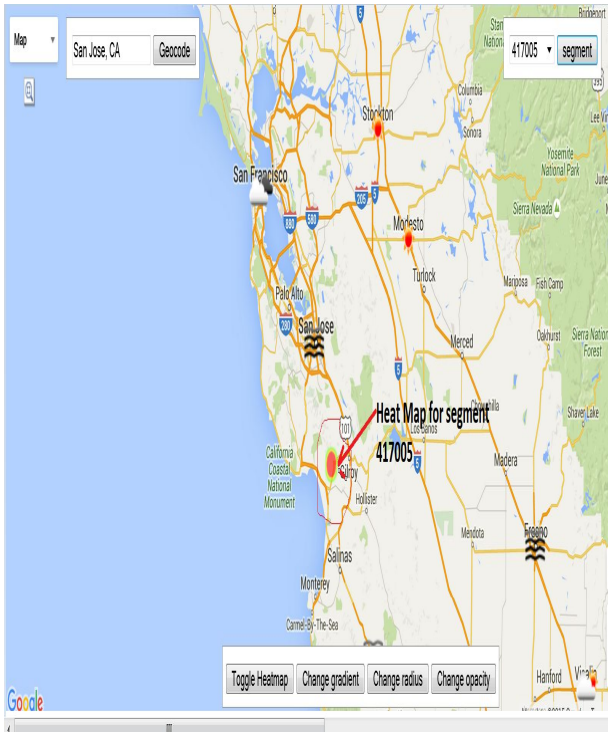


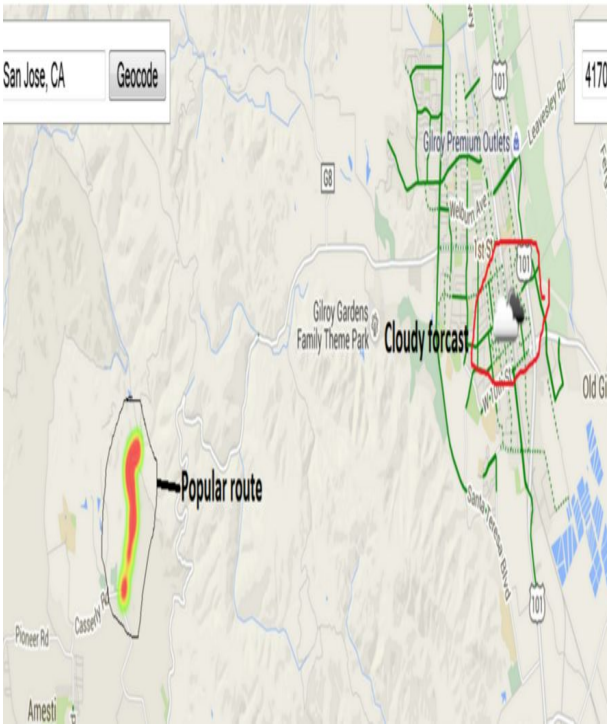

Show 10 entries Search:

| Segment | Athlete | Gender | Time | Avg. Heartrate | Rank |
|---|---|---|---|---|---|
| Brick Kiln Rd Climb | Alan Buday | M | 0:03:07 | 150.5 | 1 |
| Brick Kiln Rd Climb | Michael Dorrell | M | 0:03:14 | None | 2 |
| Brick Kiln Rd Climb | david kutcipal | M | 0:03:30 | None | 3 |
| Brick Kiln Rd Climb | Fran Roberts | M | 0:03:34 | None | 4 |
| Brick Kiln Rd Climb | Andrew McGee | M | 0:03:36 | 142.1 | 5 |

On the segment search web page, user is able to filter segment based on state and category. There are two retrieving modes, with “history segment” we will query Hive to get all segments; while “popular segment” will query Postgres for popular segment only. A Voronoi chart is build to show the segments, as mouse hovers over segment name and the number of previous attempts will be displayed. A single click on the chart will refresh the table below to update the leaderboard of this segment. Due to inherent batch nature of hive i.e. map and reduce jobs for every query, initial load of web page was pretty slow.

# Geolocation and Live weather Data Visualization on Web Server

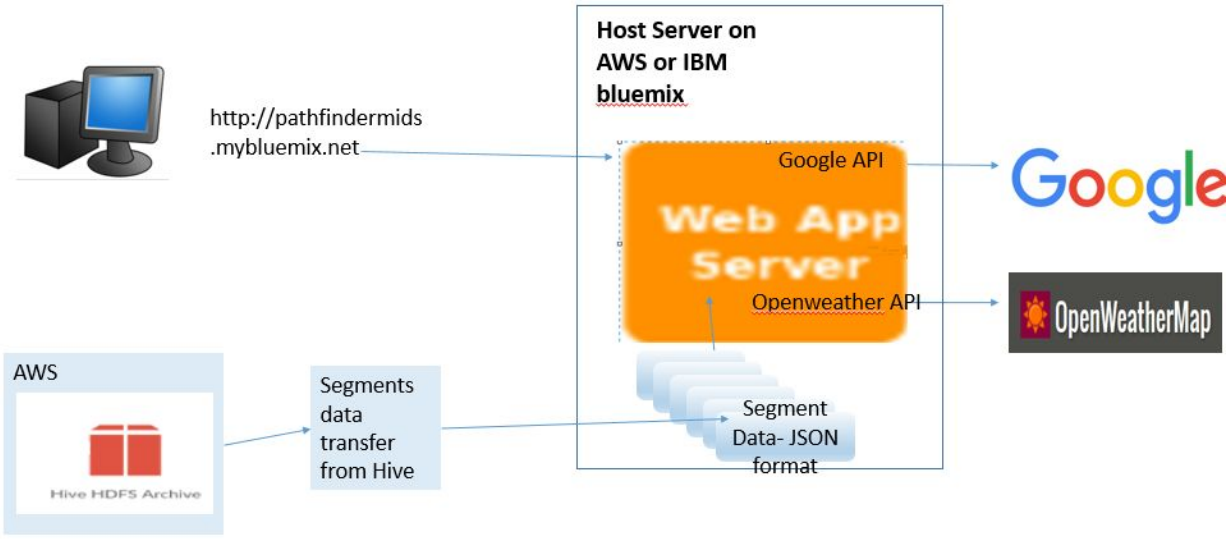



We installed the web server on IBM bluemix platform using node.js (http://pathfindermids.mybluemix.net/)
We used

- Google map API
    - Heat map generation and displayed for each of the segments of popular routes obtained from Strava API .These segment datas are in JSON format obtained after processing and filtering them.
- openweathermap api from openweathermap.org
    - openweathermap api serves live weather conditions for nearby locations in JSON format

## Architecture of Geolocation and Weather Data integration on Web Server




# Summary

## Technical Accomplishments

- We are able to design a complete data retrieval infrastructure based on Lambda architecture.
- Data serving layer consists of two modes -faster mode via PostgreSQL and Slower via Hive Thrift.
- Segment search criteria has been enhanced to include additional parameters.
- Overlaid the segment and weather data on Google maps using IBM BlueMix.
- Applied technology : AWS, IBM BlueMix, Hive, PostgreSQL, AJAX, CGI, Javascript.

## Limitations and Possible Improvements

### Data Integration

- Weather data is near real-time, we can use Spark to make it more closure to reality.
- Weather Data is not integrated with Google maps for visuals.
- Segment search does not display weather conditions.
- API limits (50K for weather and 30K for Strava) .

### Data Transformations

- Existing data cleaning is done using only Hive queries, ETL tools can be used to built robust data clearing mechanism.
- Complex views by integrating both datasets are in place for finding interesting patterns are not built yet.

### Data Serving and Visualization

- Only partial speed layer for faster visuals is in place.
- Python CGI server for Hive Thrift and PostgreSQL is not scalable.
- Dashboard can enhanced to add more reports such as leader board report with weather data.
- Existing reports can be enhanced to add mode selections criteria.

## References


[1] Database Design:  Clare Churcher
[2] Programming Hive: By: Edward Capriolo; Dean Wampler; Jason Rutherglen
[3] Big Data: Principles and best practices of scalable realtime data systems- Nathan Marz
[4] stravalib : https://github.com/hozn/stravalib
[5] Weather Data Library: https://github.com/csparpa/pyowm
[6] JSON serde: https://github.com/rcongiu/Hive-JSON-Serde
[7] Openweathermap API: http://openweathermap.org/api
[8] Google Map API: https://developers.google.com/maps/
[9] Google Maps Javascript API cookbook:By:  by Balkan Uraz , Alper Dincer