WHERE'S MY RIDE?

SANKET PATEL

W205 FINAL PROJECT

WHY REAL TIME ?

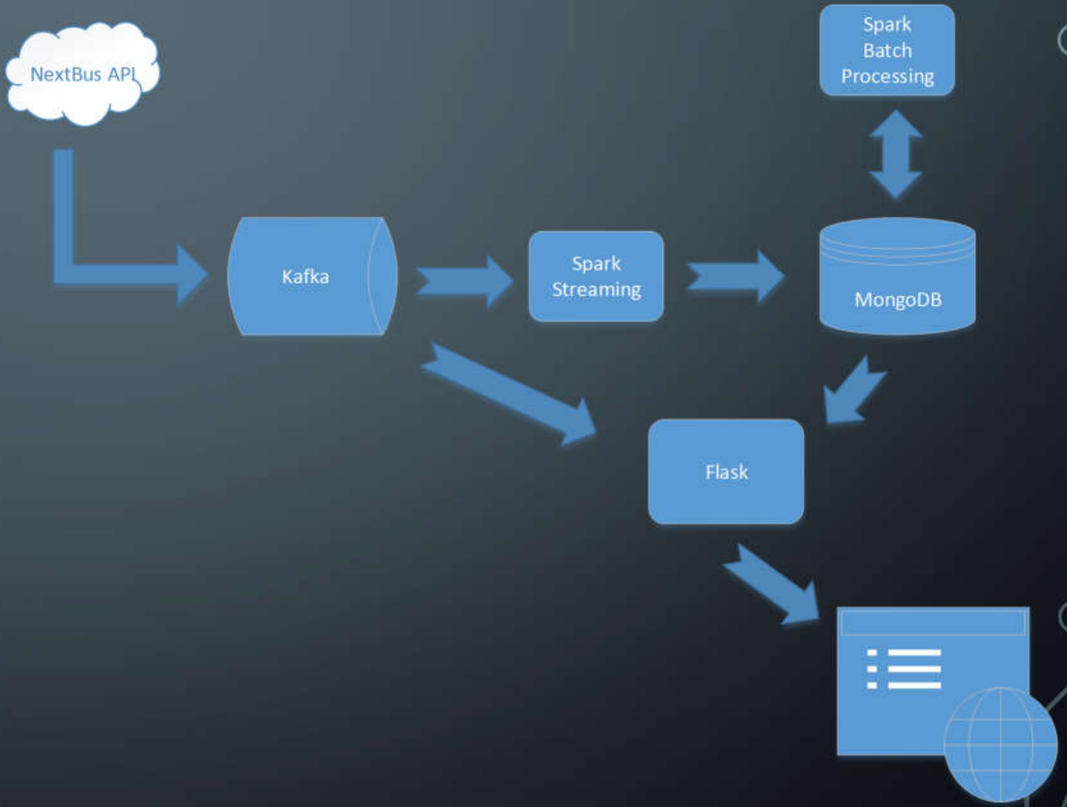
- Helps consumers better plan their commute and save time
- A better consumer experience can potentially result in increased revenue for the transit authority
- Real-time and aggregated data can be used for improving bus scheduling

DATA SOURCES

- Data sources explored
  - NextBus API http://www.nextbus.com/xmlFeedDocs/NextBusXMLFeed.pdf
  - Real Time BART API https://www.bart.gov/schedules/developers
  - OneBusAway http://developer.onebusaway.org/modules/onebusaway-application-modules/1.1.13/api/where/index.html
- Data source used
  - NextBus API http://www.nextbus.com/xmlFeedDocs/NextBusXMLFeed.pdf

ARCHITECTURE & TECHNOLOGIES

- Lambda Architecture
- All data stored in Mongo DB
- Kafka serves as pipe for all data ingestion
- Streaming layer
  - Spark Streaming job
  - Using Kafka DirectStream and checkpointing to store offsets to achieve exactly-once semantics
- Batch Layer
  - Spark job to aggregate hourly data and store in MongoDB
- Serving Layer
  - Flask
  - SocketIO
  - HTML and Google Maps JavaScript API



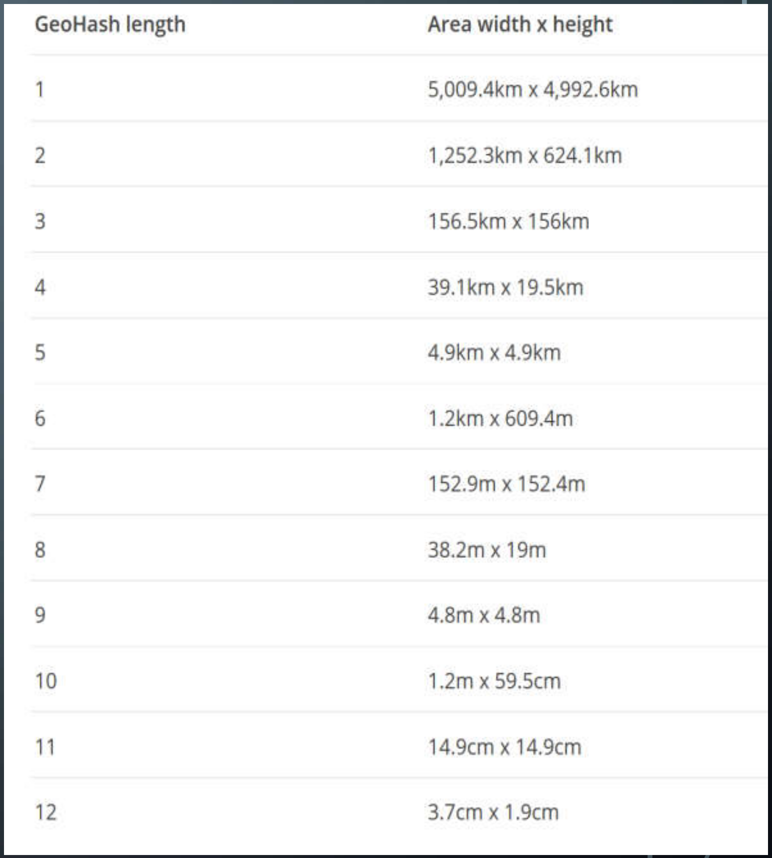
IMPLEMENTATION

- Python script runs as service to query NextBus API every 10 ms and pushes data to kafka
- Spark Stream job cleans the data for storing in MongoDB and also calculates GeoHash for the coordinates
- The streaming data is stored in MongoDB Collection indexed on Geo Coordinates
- A Spark Bath job is kicked off via Cron Job every hour to aggregate data based on GeoHash and stored back into MongoDB collection
- The aggregated data is used for generating HeatMap
- Flask as the serving layer,
  - Implements kafka consumer to get real-time data
  - The real-time data is pushed to WEB UI via WebSockets
  - Serves aggregate data by querying MongoDB aggregate collection

ALGORITHMS

- GeoHashing
  - Encode(lat,lon) -> “String”
  - Generates same hash for coordinates in a given rectangle
  - The size of the rectangle is determined by the length of the hash
  - Allows easy aggregation based on location

- Other algorithms
  - DBSCAN
    - Works bottom-up by picking a point and looking for more points within a given distance
    - Expands the cluster by repeating this process for new points until the cluster cannot be further expanded
    - Its tuned using 2 parameters:
      - ***Epsilon –*** This determines how far to search for points near a given point
      - ***MinPoints -*** This determines how many points should be present in the neighborhood of a given point in order to keep expanding a given cluster

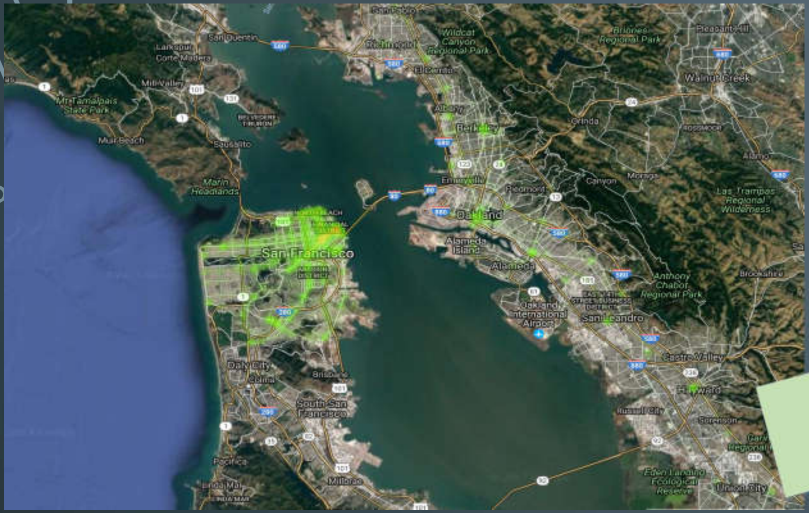


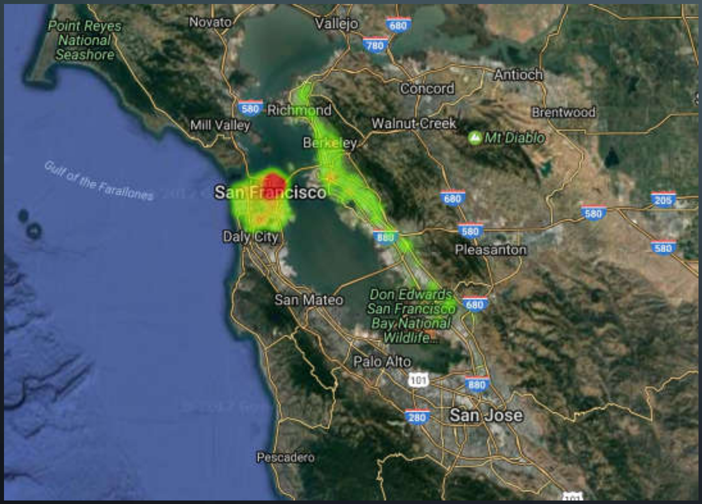
GeoHash length	Area width x height
1	5,009.4km x 4,992.6km
2	1,252.3km x 624.1km
3	156.5km x 156km
4	39.1km x 19.5km
5	4.9km x 4.9km
6	1.2km x 609.4m
7	152.9m x 152.4m
8	38.2m x 19m
9	4.8m x 4.8m
10	1.2m x 59.5cm
11	14.9cm x 14.9cm
12	3.7cm x 1.9cm

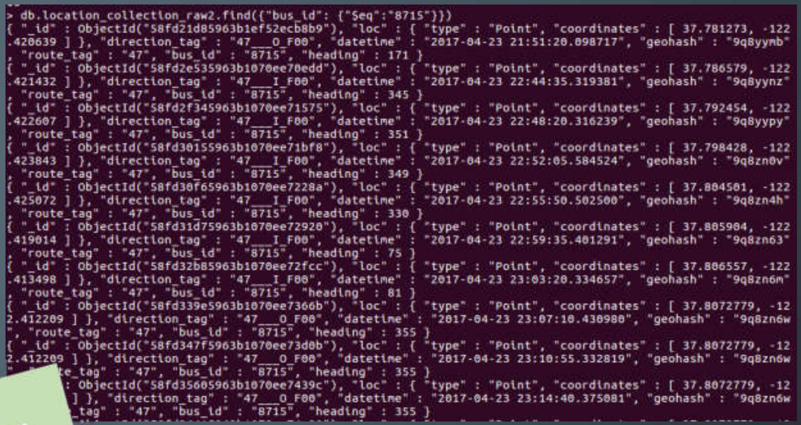
TECHNICAL CHALLENGES

- Since the data ingest scripts queries NextBus API every 10ms, it creates bursts of data. Had to make sure streaming job is never overwhelmed by setting *spark.streaming.kafka.maxRatePerPartition*
- Achieving exactly-once semantics
- Serving Real-time locations to Web Page
- JavaScript/HTML (not a big fan ☺)

RESULTS

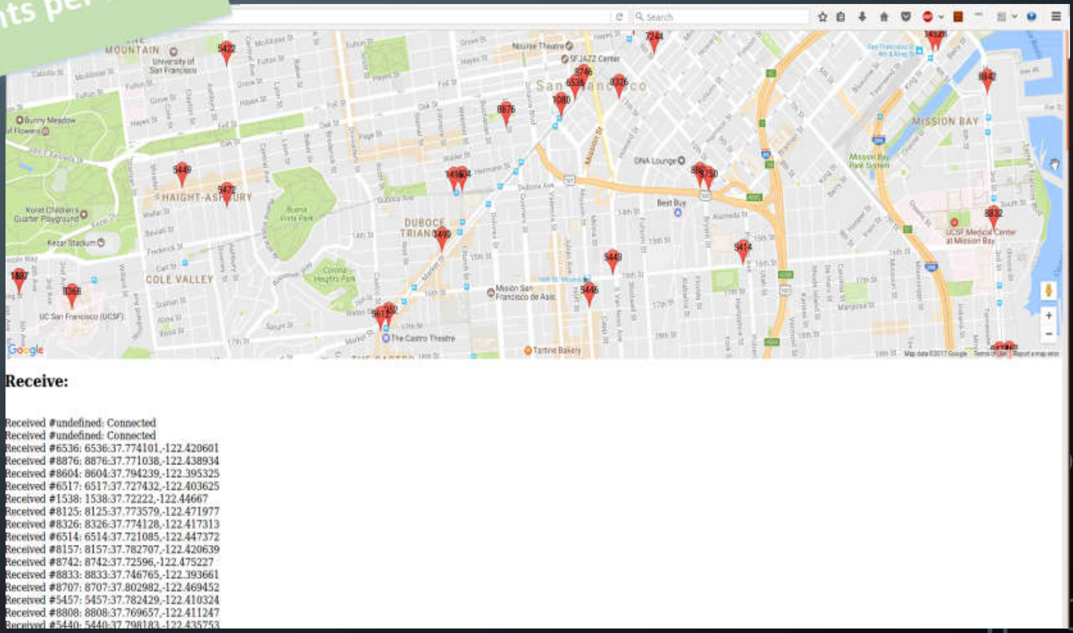






```
> db.location_collection_raw2.find({"bus_id": {"$eq":"8715"}})
{ "_id" : ObjectId("58fd21d85963b1ef52ecb8b9"), "loc" : { "type" : "Point", "coordinates" : [ 37.781273, -122
.420639 ] }, "direction_tag" : "47___O_F00", "datetime" : "2017-04-23 21:51:20.098717", "geohash" : "9q8yymb"
, "route_tag" : "47", "bus_id" : "8715", "heading" : 171 }
{ "_id" : ObjectId("58fd2e535963b1070ee70edd"), "loc" : { "type" : "Point", "coordinates" : [ 37.786579, -122
.421432 ] }, "direction_tag" : "47___I_F00", "datetime" : "2017-04-23 22:44:35.319381", "geohash" : "9q8yynz"
, "route_tag" : "47", "bus_id" : "8715", "heading" : 345 }
{ "_id" : ObjectId("58fd2f345963b1070ee71575"), "loc" : { "type" : "Point", "coordinates" : [ 37.792454, -122
.422607 ] }, "direction_tag" : "47___I_F00", "datetime" : "2017-04-23 22:48:20.316239", "geohash" : "9q8yypy"
, "route_tag" : "47", "bus_id" : "8715", "heading" : 351 }
{ "_id" : ObjectId("58fd30155963b1070ee71bf8"), "loc" : { "type" : "Point", "coordinates" : [ 37.798428, -122
.423843 ] }, "direction_tag" : "47___I_F00", "datetime" : "2017-04-23 22:52:05.584524", "geohash" : "9q8zn0v"
, "route_tag" : "47", "bus_id" : "8715", "heading" : 349 }
{ "_id" : ObjectId("58fd30f65963b1070ee7228a"), "loc" : { "type" : "Point", "coordinates" : [ 37.804501, -122
.425072 ] }, "direction_tag" : "47___I_F00", "datetime" : "2017-04-23 22:55:50.502500", "geohash" : "9q8zn4h"
, "route_tag" : "47", "bus_id" : "8715", "heading" : 330 }
{ "_id" : ObjectId("58fd31d75963b1070ee72920"), "loc" : { "type" : "Point", "coordinates" : [ 37.805904, -122
.419014 ] }, "direction_tag" : "47___I_F00", "datetime" : "2017-04-23 22:59:35.401291", "geohash" : "9q8zn63"
, "route_tag" : "47", "bus_id" : "8715", "heading" : 75 }
{ "_id" : ObjectId("58fd32b85963b1070ee72fcc"), "loc" : { "type" : "Point", "coordinates" : [ 37.806557, -122
.413498 ] }, "direction_tag" : "47___I_F00", "datetime" : "2017-04-23 23:03:20.334657", "geohash" : "9q8zn6m"
, "route_tag" : "47", "bus_id" : "8715", "heading" : 81 }
{ "_id" : ObjectId("58fd339e5963b1070ee7366b"), "loc" : { "type" : "Point", "coordinates" : [ 37.8072779, -12
2.412209 ] }, "direction_tag" : "47___O_F00", "datetime" : "2017-04-23 23:07:10.430980", "geohash" : "9q8zn6w
", "route_tag" : "47", "bus_id" : "8715", "heading" : 355 }
{ "_id" : ObjectId("58fd347f5963b1070ee73d0b"), "loc" : { "type" : "Point", "coordinates" : [ 37.8072779, -12
2.412209 ] }, "direction_tag" : "47___O_F00", "datetime" : "2017-04-23 23:10:55.332819", "geohash" : "9q8zn6w
", "route_tag" : "47", "bus_id" : "8715", "heading" : 355 }
{ "_id" : ObjectId("58fd35605963b1070ee7439c"), "loc" : { "type" : "Point", "coordinates" : [ 37.8072779, -12
2.412209 ] }, "direction_tag" : "47___O_F00", "datetime" : "2017-04-23 23:14:40.375081", "geohash" : "9q8zn6w
", "route_tag" : "47", "bus_id" : "8715", "heading" : 355 }
```

40K events per hour



Receive:

```
Received #undefined: Connected
Received #undefined: Connected
Received #6536: 6536:37.774101,-122.420601
Received #8876: 8876:37.771038,-122.438934
Received #8604: 8604:37.794239,-122.395325
Received #6517: 6517:37.727432,-122.403625
Received #1538: 1538:37.72222,-122.44667
Received #8125: 8125:37.773579,-122.471977
Received #8326: 8326:37.774128,-122.417313
Received #6514: 6514:37.721085,-122.447372
Received #8157: 8157:37.782707,-122.420639
Received #8742: 8742:37.72596,-122.475227
Received #8833: 8833:37.746765,-122.393661
Received #8707: 8707:37.802982,-122.469452
Received #5457: 5457:37.782429,-122.410324
Received #8808: 8808:37.769657,-122.411247
Received #5440: 5440:37.798183,-122.435753
```

LIMITATIONS AND EXTENSION

- Limitations
  - Only one client can connect to real-time data. SocketIO can support multiple clients using Rooms and Broadcasting
  - Aggregate job doesn't back fill if system goes down for multiple hours. This can be fixed by having the Cron Job script query MongoDB for missing hours and launch multiple aggregate jobs
  - Only one data source
- Future Extensions
  - Add more data sources (different cities, modes of transport etc)
  - Integrating Uber/Lyft data along with weather data can help consumers make more efficient transport selection