Analyzing and Visualizing Seattle Micromobility Data

SCS 3252 - Big Data Management System & Tools

By: Jessie Ma and Dorothy Liu

The Rise of Shareable Vehicles & GBFS Data

- Shareable vehicles have risen in popularity over the past decade; e.g. Bike Share
- Demand for micromobility data from operators has also increased
- In 2015: the North-American Bike-Share Association (NABSA) introduced General Bikeshare Feed Specification (GBFS) as a standardized way to share real-time data feeds
- GBFS data benefits everyone in the shared mobility space
- Today we are analyzing and visualizing data from a Seattle provider: Lime

Agenda

1. Spark structured streaming & Databricks built-in dashboards
2. Spark dataframe & Power BI

Spark Structured Streaming

What's the big idea?

- A real-time dashboard to display the number of available bikes per neighbourhood in Seattle
- Micromobility operator can look and the dashboard and quickly decipher where more supplies might be needed, and the current demand
- For privacy reasons data on bikes in use is not available to the public, if it was, we would include that in the dashboard as well to show number of available vs unavailable bikes

Problem with Streaming from an HTTP Endpoint

- Originally, we wanted to stream from the Lime web API, which produces something that looks like this every time a GET request is passed:


```
{"last_updated":1701476733,"ttl":0,"version":"1.0","data":{"bikes":[{"bike_id":"6863fc20-
7a8c-4e05-a4c1-
74094d1492a9","lat":47.6159,"lon":-122.3168,"is_reserved":0,"is_disabled":0,"vehicle_type":
"bike"},{"bike_id":"38a27678-38fe-41dc-9682-
1a36c55c5cc7","lat":47.6151,"lon":-122.3168,"is_reserved":0,"is_disabled":0,"vehicle_type":
"scooter"},{"bike_id":"7fe1de17-4653-4fc1-b925-
cfd42bee1a50","lat":47.6019,"lon":-122.3168,"is_reserved":0,"is_disabled":0,"vehicle_type":
"bike"},{"bike_id":"4f914993-eb6c-438b-9a59-
08da43caa1b2","lat":47.6108,"lon":-122.3168,"is_reserved":0,"is_disabled":0,"vehicle_type":
"bike"},{"bike_id":"7d90389f-21ff-4daa-9b2a-
5c224a20ae8b","lat":47.606,"lon":-122.3169,"is_reserved":0,"is_disabled":0,"vehicle_type":"
bike"},{"bike_id":"b62ca04a-d31b-4173-8c97-
```

Problem with Streaming from an HTTP Endpoint

- However we quickly realized it’s not supported by Spark structured streaming
- Commonly supported streaming sources include:
  - Kafka
  - Simple file source
  - HDFS
- So why not the web API?

Workaround

- Instead of streaming directly from the API, we decided to pull the API data into json files, and stream from the file source.
- A simple scraper was built to pass a get request every minute
- Each bike record looks like this:


```json
{
    "bike_id": "8511ee3a-900c-4eb5-b982-91dac75a75c6",
    "lat": 47.5611,
    "lon": -122.3249,
    "is_reserved": 0,
    "is_disabled": 0,
    "vehicle_type": "bike"
},
```

Problem with Streaming from an HTTP Endpoint

- Scalability: HTTP streaming implies a long-term open connection to the server, which is not scalable from the producer side. Streaming can consume a significant amount of resources.
- Nature of HTTP: HTTP's inherent design is request-response oriented, rather than for streaming. HTTP transactions are single requests.
- Fault tolerance: HTTP protocol is stateless, whereas Spark streaming's fault tolerance requires the state of processing to be known. With a stateless protocol, it is difficult to keep track of the records being processed, and reprocess data if a failure does occur.
- Rate limiting: Most HTTP endpoints have rate limiting imposed to prevent DDoS attacks. In a streaming context, this may impact the update frequency of the data source.

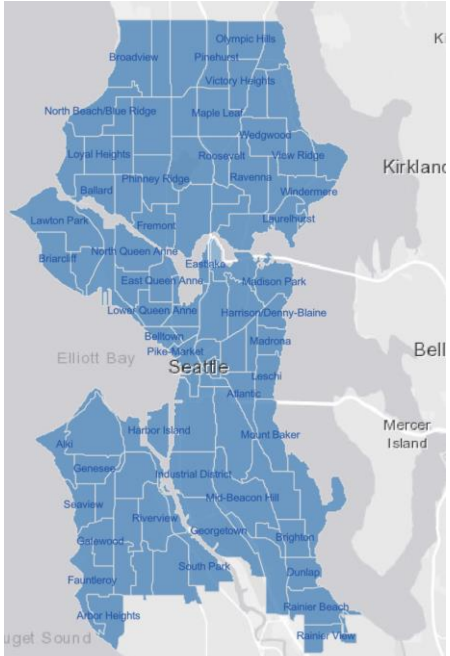
Processing Geospatial Data using Spark

- The bike json is missing a key data point: the district the bike is in.
- We do have the lat and lon, so we can assign districts to bikes using that
- First, we need a geojson file of Seattle's districts from the City's open data portal

Processing Geospatial Data using Spark

- Geojson is basically a json file with district boundary coordinates


```
"type": "Feature",
"properties": {
    "OBJECTID": 28,
    "L_HOOD": "Ballard",
    "S_HOOD_ALT_NAMES": "Loyal Heights, Adams,
    Ballard",
    "Shape__Area": 104603463.349548,
    "Shape__Length": 59248.568771862301
},
"geometry": {
    "type": "Polygon",
    "coordinates": [
        [
            [
                -122.402657483487005,
                47.696015653243201
            ],
            [
                -122.402362521434995,
                47.695276404014002
```

Processing Geospatial Data using Spark

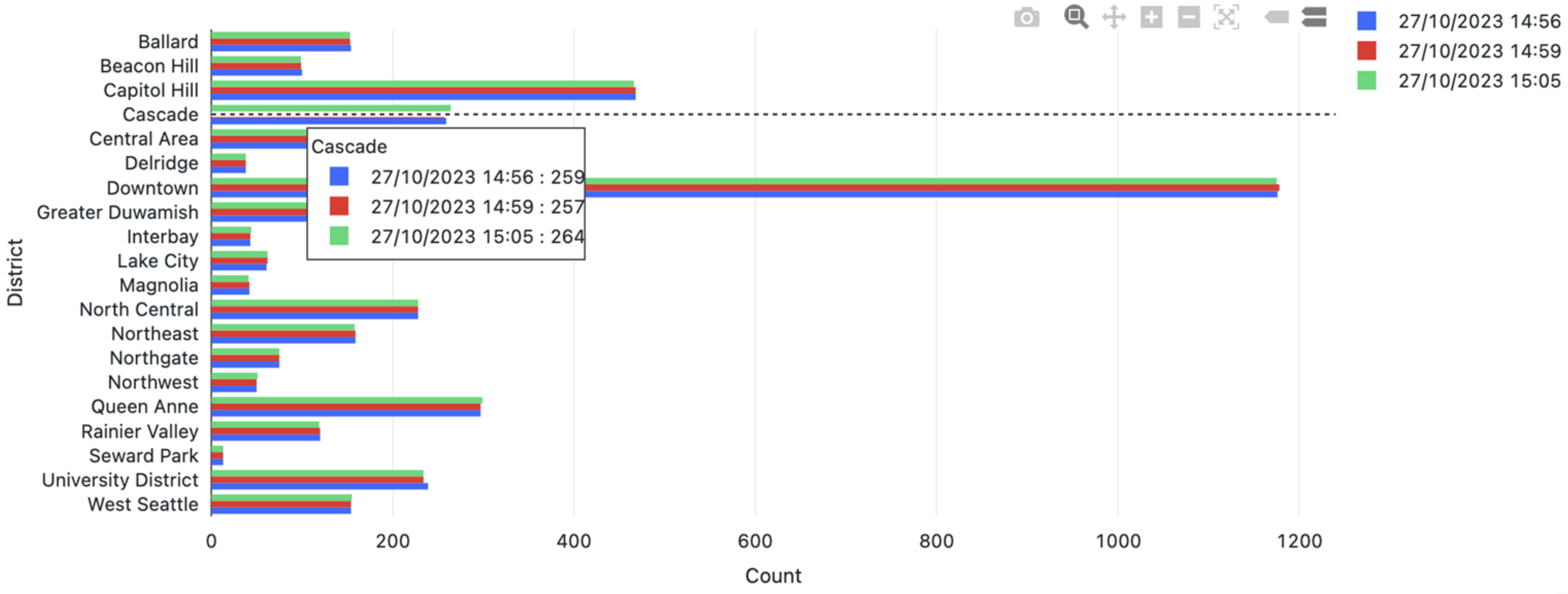
- The geojson was converted to a geopandas dataframe for easier processing
- In order to leverage Spark's parallel processing capability, the geopandas dataframe was broadcasted to all nodes - this way all nodes can work to quickly assign the correct district to each bike record
- A UDF was made to assign the correct districts to bikes

```
sparkDistrictGDF = sc.broadcast(districtGDF)
def assignDistrictToBike (lat,lon):
   mgdf = sparkDistrictGDF.value
   idx = mgdf[mgdf["geometry"].intersects(Point(lon, lat))].first_valid_index()
   return mgdf.loc[idx]["L_HOOD"] if idx is not None else None
findDistrictUDF = udf(assignDistrictToBike, StringType())
```

Watermark and Window for This Stream

- Since the GET request gives us all the data in one go when the request is executed, there's no need to apply a watermark - there will be no late arriving data
- The window for this stream is set to 1 second. Really we shouldn't need a window either - again the data is fetched in one go, so there's no need to further group the data by timestamp. All data fetched by one GET request will have the same timestamp.

Final Dashboard



The dashboard only gets updated if there's a change in bike status, which was sometimes more than a minute apart

Fun Anecdote

- We actually broke Spark… well, sort of
- We tried to execute this SQL command to display only the most recent data window, rather than all the windows:
  - "SELECT * FROM windowed_view WHERE window.end = (SELECT MAX(window.end) FROM windowed_view)"
- We then got this error:
  - org.apache.spark.SparkException: [INTERNAL_ERROR] The Spark SQL phase planning failed with an internal error. You hit a bug in Spark or the Spark plugins you use. Please, report this bug to the corresponding communities or vendors, and provide the full stack trace.
- It was funny, but we didn't end up having enough time to implement another method to just display the most recent window.

Spark Dataframe and Power BI

What is the goal?

- Another approach to visualizing the same dataset, using a Databricks partner connection: Power BI
- The visualizations can aid policy data analysts to view the distribution of bikes across the city and gauge supply and demand
- Steps discussed:
  - Perform some minor transformations on the JSON files
  - Read a file into a Spark dataframe
  - Cleanse the data
  - Query the data
  - Visualize in Power BI

Transformations on JSON files

- Removed unwanted formatting
- 'Last_updated' key-value was appended to every document
- 'Is_reserved' and 'is_disabled' values converted to boolean

```python
import os
import json

def update_json_file(input_file, output_path):
    with open(input_file, 'r') as file:
        data = json.load(file)
        last_updated = data.get('last_updated')
        new_data = data.get('data', {}).get('bikes')
        new_data2 = [dict(row, last_updated=last_updated) for row in new_data]

    for entry in new_data2:
        if "is_reserved" in entry and entry["is_reserved"] == 0:
            entry["is_reserved"] = False
        elif "is_reserved" in entry and entry["is_reserved"] == 1:
            entry["is_reserved"] = True

        if "is_disabled" in entry and entry["is_disabled"] == 0:
            entry["is_disabled"] = False
        elif "is_disabled" in entry and entry["is_disabled"] == 1:
            entry["is_disabled"] = True

      output_folder = os.path.join(output_path, 'lime_lastupdate')
    os.makedirs(output_folder, exist_ok=True)

    with open(os.path.join(output_folder, os.path.basename(input_file)), 'w') as file:
        json.dump(new_data2, file, indent=2)

def process_files_in_directory(directory):
    for filename in os.listdir(directory):
        if filename.endswith('.json'):
            input_file = os.path.join(directory, filename)
            update_json_file(input_file, directory)

if __name__ == "__main__":
      directory_path = r'C:\UofT SCS - 3252\Final Project\lime'

    process_files_in_directory(directory_path)
```


```json
{
   "bike_id": "c80f6da3-d152-4477-a7d1-4b377420828c",
    "lat": 47.6073,
    "lon": -122.3328,
    "is_reserved": false,
    "is_disabled": false,
    "vehicle_type": "scooter",
    "last_updated": 1698419158
}
```

Cleansing Pyspark dataframe

bike_id	is_disabled	is_reserved	last_updated	lat	lon	vehicle_type	district
c80f6da3-d152-447...	false	false	2023-10-27 15:05:58	47.6073	-122.3328	scooter	Downtown
ebdc686b-80af-461...	false	false	2023-10-27 15:05:58	47.5547	-122.2625	bike	Seward Park
ebd529c4-1253-4c9...	false	false	2023-10-27 15:05:58	47.6017	-122.2853	scooter	Central Area
a0204069-65b2-4d4...	false	false	2023-10-27 15:05:58	47.5902	-122.2865	scooter	Rainier Valley
b0e97838-4ab0-47b...	false	false	2023-10-27 15:05:58	47.6036	-122.2916	scooter	Central Area
79afc903-a3a4-4b9...	false	false	2023-10-27 15:05:58	47.5902	-122.2924	bike	Rainier Valley
8dffa473-3fb7-428...	false	false	2023-10-27 15:05:58	47.5983	-122.2945	scooter	Central Area
4da97672-2983-40d...	false	false	2023-10-27 15:05:58	47.6091	-122.2951	scooter	Central Area
d424e2fa-fa4a-407...	false	false	2023-10-27 15:05:58	47.6035	-122.2951	scooter	Central Area
8db6c175-9f4e-4ed...	false	false	2023-10-27 15:05:58	47.6075	-122.2958	bike	Central Area
4e1507ae-2c07-42e...	false	false	2023-10-27 15:05:58	47.6081	-122.2959	scooter	Central Area
ce6675e2-d9b1-4c1...	false	false	2023-10-27 15:05:58	47.6081	-122.2973	bike	Central Area
555df71e-91ca-4b3...	false	false	2023-10-27 15:05:58	47.6027	-122.2994	scooter	Central Area
4636c1f8-100b-4b4...	false	false	2023-10-27 15:05:58	47.6002	-122.2999	scooter	Central Area
d02c380f-6ea6-493...	false	false	2023-10-27 15:05:58	47.606	-122.3003	bike	Central Area
4164b1e7-4a92-4eb...	false	false	2023-10-27 15:05:58	47.5992	-122.3011	bike	Central Area
9b3cd7bb-4314-47b...	false	false	2023-10-27 15:05:58	47.6044	-122.3011	bike	Central Area
941cf181-daa8-40b...	false	false	2023-10-27 15:05:58	47.5966	-122.3013	bike	Central Area

- Uploaded one JSON for snapshot of point in time
- Read into Pyspark dataframe
- Assigned Seattle districts to bikes using geojson file
- Added new ‘district’ column
- Converted ‘last_updated’ UNIX time to readable format
- Dropped NAs from district column

Querying Pyspark dataframe Part I

```
grouped_districts=new_df.groupBy("district").count()
top_districts=grouped_districts.sort(col("count").desc()).show()
```

```
+-------------------+-----+
|           district|count|
+-------------------+-----+
|           Downtown| 1175|
|       Capitol Hill|  466|
|       Central Area|  344|
|         Queen Anne|  299|
|            Cascade|  264|
|University District|  234|
|    Greater Duwamish|  232|
|      North Central|  228|
|          Northeast|  158|
|       West Seattle|  155|
|            Ballard|  153|
|     Rainier Valley|  119|
|        Beacon Hill|   99|
|          Northgate|   75|
|          Lake City|   62|
|          Northwest|   51|
|           Interbay|   44|
|           Magnolia|   41|
```

- Downtown, Capitol Hill and Central Area have the highest numbers of Lime vehicles (top 3)
- Downtown has over 2x more bikes than Capitol Hill

Querying Pyspark dataframe Part II

```
grouped_vehicles=new_df.groupBy("vehicle_type").count()
grouped_vehicles.show()
```

```
+------------+-----+
|vehicle_type|count|
+------------+-----+
|     scooter| 2503|
|        bike| 1747|
+------------+-----+
```

- There were more scooters than bikes available

Querying Pyspark dataframe Part III

```
df_scooter=new_df.filter(new_df.vehicle_type=="scooter")
grouped_scooter=df_scooter.groupBy("district").count()
top_districts_scooter=grouped_scooter.sort(col("count").desc()).show()
df_bike=new_df.filter(new_df.vehicle_type=="bike")
grouped_bike=df_bike.groupBy("district").count()
top_districts_bike=grouped_bike.sort(col("count").desc()).show()
```

```
+-------------------+-----+
|           district|count|
+-------------------+-----+
|           Downtown|  662|
|        Capitol Hill|  285|
|        Central Area|  209|
|          Queen Anne|  180|
|             Cascade|  164|
|    Greater Duwamish|  161|
|        West Seattle|  135|
|University District|  128|
|       North Central|  109|
|           Northeast|   77|
|      Rainier Valley|   75|
|         Beacon Hill|   64|
|             Ballard|   63|
|           Northgate|   49|
|           Lake City|   37|
|            Interbay|   30|
|           Northwest|   28|
|            Delridge|   22|
```

```
+-------------------+-----+
|           district|count|
+-------------------+-----+
|           Downtown|  513|
|        Capitol Hill|  181|
|        Central Area|  135|
|       North Central|  119|
|          Queen Anne|  119|
|University District|  106|
|             Cascade|  100|
|             Ballard|   90|
|           Northeast|   81|
|    Greater Duwamish|   71|
|      Rainier Valley|   44|
|         Beacon Hill|   35|
|           Northgate|   26|
|            Magnolia|   26|
|           Lake City|   25|
|           Northwest|   23|
|        West Seattle|   20|
|            Delridge|   16|
```

- Downtown, Capitol Hill and Central Area had the highest counts of both scooters and bikes (top 3)

Querying Pyspark dataframe Part IV

```
df_disabled_scooter=df_scooter.where(col("is_disabled"))
df_disabled_scooter.show()
#grouped_disabled_scooter=df_disabled_scooter.groupBy("district").count()
#top_districts_disabled_scooter=grouped_disabled_scooter.sort(col("count").desc()).show()
df_disabled_bike=df_bike.where(col("is_disabled"))
df_disabled_bike.show()
#grouped_disabled_bike=df_disabled_bike.groupBy("district").count()
#top_districts_disabled_bike=grouped_disabled_bike.sort(col("count").desc()).show()
```

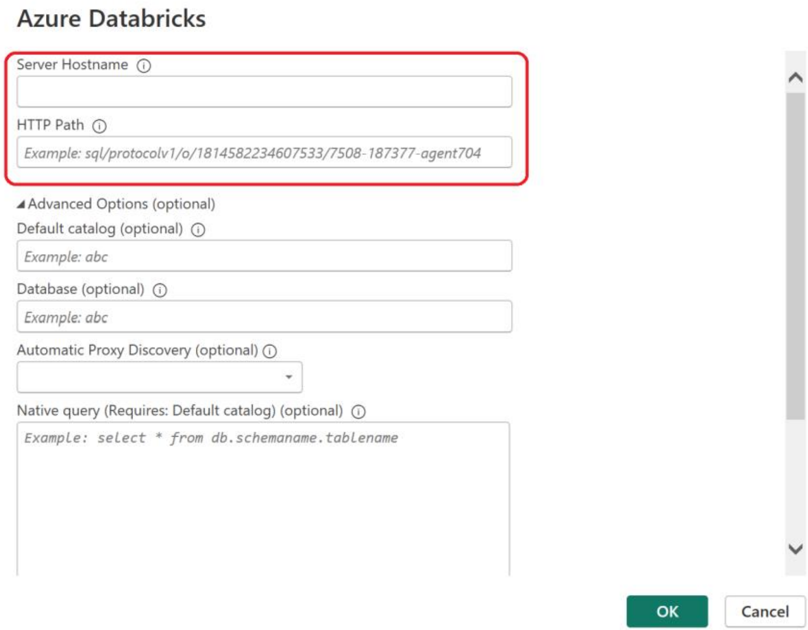
```
+-------+-----------+-----------+------------+---+---+------------+--------+
|bike_id|is_disabled|is_reserved|last_updated|lat|lon|vehicle_type|district|
+-------+-----------+-----------+------------+---+---+------------+--------+
+-------+-----------+-----------+------------+---+---+------------+--------+

+-------+-----------+-----------+------------+---+---+------------+--------+
|bike_id|is_disabled|is_reserved|last_updated|lat|lon|vehicle_type|district|
+-------+-----------+-----------+------------+---+---+------------+--------+
+-------+-----------+-----------+------------+---+---+------------+--------+
```

- No broken scooters or bikes in this snapshot - no further analysis conducted

Power BI Integration

```
new_df.write.mode("overwrite").option("path","/power_bi").saveAsTable("finalDF")
```



- Saved dataframe as delta table in Databricks
- Connected to Databricks from Power BI
- Selected table and loaded data
- Renamed the columns

Power BI Demo

Key Learnings

- Demonstrated our end-to-end data pipelines consisting of ingesting, processing, querying and serving GBFS Lime data to end user
- Streaming requires very specific data sources
- Leveraging existing Python libraries in Spark context requires some tweaking (e.g. broadcasting)
- Data preparation is one of the longest and most crucial steps of designing a pipeline
- Connecting Power BI to Databricks was seamless
- Only a snapshot, to gain a comprehensive understanding, more data will need to be ingested and analyzed

Thank you for listening!