Universidad Nacional Autónoma de México

Por mi raza hablará el espíritu”

Facultad de Estudios Superiores Acatlán

Licenciatura en Matemáticas Aplicadas y Computación

Práctica Spark

Programación Paralela y Concurrente

Peralta Cortés Jorge Alejandro

Doctor José Gustavo Fuentes Cabrera

1. Introducción

Esta práctica consiste en la aplicación de Apache Spark para analizar el tráfico en el sistema de bicicletas públicas Ecobici de la Ciudad de México. El objetivo es procesar un gran volumen de datos de viajes para extraer información útil sobre el uso del sistema, como las estaciones más populares y las horas pico de actividad. Spark es una herramienta ideal para este tipo de tareas debido a su capacidad para procesar grandes conjuntos de datos de manera distribuida y eficiente.

2. Descripción de los datos

Los datos utilizados en este análisis provienen del portal de datos abiertos del sistema Ecobici de la Ciudad de México (Ecobici Datos Abiertos). Se descargaron los archivos históricos mensuales en formato CSV, cubriendo el período desde abril de 2024 hasta marzo de 2025. Estos datos están organizados principalmente en dos tipos de archivos:

- **Historial de Viajes:** Archivos CSV mensuales que contienen detalles sobre cada viaje realizado.
- **Ubicación de Cicloestaciones:** Un archivo CSV que contiene información sobre cada estación de Ecobici, incluyendo su ubicación geográfica.

A continuación, se describen las columnas más relevantes utilizadas en el análisis:

`df (Viajes)` - Dataframe que contiene la información unificada de todos los viajes mensuales.

- `Genero_Usuario`: Género registrado del usuario ('M', 'F', 'O').
- `Edad_Usuario`: Edad registrada del usuario.
- `Bici`: Identificador único de la bicicleta utilizada.
- `Ciclo_Estacion_Retiro`: ID numérico de la cicloestación donde se retiró la bicicleta.
- `Fecha_Retiro`: Fecha en que se retiró la bicicleta (YYYY-MM-DD).
- `Hora_Retiro`: Hora en que se retiró la bicicleta (HH:MM:SS).
- `Ciclo_EstacionArribo`: ID numérico de la cicloestación donde se devolvió la bicicleta.
- `Fecha_Arribo`: Fecha en que se devolvió la bicicleta (YYYY-MM-DD).
- `Hora_Arribo`: Hora en que se devolvió la bicicleta (HH:MM:SS).

`df_stations (Estaciones)` - Dataframe que contiene la información de las cicloestaciones.

- `num_cicloe`: Identificador numérico único de la cicloestación. Usado como clave para unir con los datos de viajes.

- `calle_prin`: Nombre de la calle principal donde se ubica la estación. Usado para identificar las estaciones en los resultados.
- `latitud`: Latitud geográfica de la estación. Usada para visualización en mapas.
- `longitud`: Longitud geográfica de la estación. Usada para visualización en mapas.

3. Tratamiento y Operaciones con Spark

El procesamiento de los datos se realizó utilizando Spark, con su API de Python PySpark. Las operaciones clave fueron las siguientes:

Dado que los datos históricos de viajes estaban divididos en archivos CSV mensuales, el primer paso fue cargarlos y unificarlos en un único DataFrame Spark.

- Cada archivo CSV fue leído en un DataFrame Spark usando `spark.read.csv`
- Los nombres de las columnas que contenían espacios fueron renombrados reemplazando los espacios por guiones bajos para facilitar su manejo en Spark SQL y operaciones de DataFrame.
- Todos los DataFrames individuales fueron combinados en uno solo usando la operación `union`. Esto con el propósito de tener un dataset con una gran cantidad de datos para probar las capacidades de Spark.

Listing 1: Carga y unificación de archivos CSV mensuales.

```
1
2  path = 'Datos/Historicos/*.csv'
3  files = glob.glob(path)
4
5  dfs = []
6  for file in files:
7
8  temp_df = spark.read.csv(file, header=True, inferSchema=True)
9      temp_df = temp_df.toDF(*[col.replace(' ', '_') for col in
           temp_df.columns])
10     print(f"Loaded {file} with {temp_df.count()} rows")
11     dfs.append(temp_df)
12
13 # Unificar todos los dataframes en uno solo
14 df = dfs[0]
15 for i in range(1, len(dfs)):
16     df = df.union(dfs[i])
17
18 df.printSchema()
```

Para determinar las estaciones más populares, se agruparon los datos de viajes por estación de retiro y arribo usando `groupBy`. Esto permitió contar el número de viajes iniciados y

finalizados en cada estación. Las columnas `Ciclo_Estacion_Retiro` y `Ciclo_EstacionArribo` fueron utilizadas como claves para estas agrupaciones.

Para añadir información geográfica (latitud, longitud) y el nombre de la calle a las cuentas de retiros y arribos, se realizó una unión (join) con el DataFrame de estaciones (`df_stations`).

- Se utilizó `join` para combinar las cuentas de retiros (`station_departure_counts`) con la información de las estaciones (`stations_info`) usando `station_id` como clave. Se usó un `left join` para mantener todas las estaciones que tuvieron retiros.
- Se realizó otro `left join` para añadir las cuentas de arribos (`station_arrival_counts`) al resultado anterior, nuevamente usando `station_id` como clave.

Listing 2: Cálculo de viajes por estación y unión con información geográfica.

```
station_departure_counts = df.groupBy('Ciclo_Estacion_Retiro') \
                             .count() \
                             .withColumnRenamed('
                                Ciclo_Estacion_Retiro', 'station_id'
                                ) \
                             .withColumnRenamed('count', '
                                count_retiro')

station_arrival_counts = df.groupBy('Ciclo_EstacionArribo') \
                           .count() \
                           .withColumnRenamed('Ciclo_EstacionArribo'
                              , 'station_id') \
                           .withColumnRenamed('count', 'count_arribo
                              ')

stations_info = df_stations.select(
    F.col('num_cicloe').alias('station_id'),
    F.col('calle_prin').alias('station_name'),
    'latitud',
    'longitud'
)

station_counts_with_info = station_departure_counts.join(
    stations_info,
    on='station_id',
    how='left'
)

station_count_clean = station_counts_with_info.join(
    station_arrival_counts,
    on='station_id',
    how='left'
)
```

```
29
30 station_count = station_count_clean.fillna(0, subset=['count_retiro'
       , 'count_arribo'])
31
32 station_count.orderBy('count_retiro', ascending=False).show(10)
```

Para identificar las horas pico de uso, se extrajo la hora del día de las columnas `Hora_Retiro` y `Hora_Arribo`.

- Se aplicó `groupBy` sobre la columna de hora extraída (`hora_retiro` o `hora_arribo`) seguido de `count()` para obtener el número de viajes por cada hora del día.
- Los resultados se ordenaron para ver las horas con mayor tráfico.

Listing 3: Extracción y conteo de viajes por hora de retiro.

```
1 df_with_hour = df.withColumn("hora_retiro", F.hour(F.to_timestamp(F.
      col("Hora_Retiro"), "HH:mm:ss")))
2
3 hourly_traffic = df_with_hour.groupBy("hora_retiro").count().orderBy
      ("count", ascending=False)
4
5 print("Horas de mayor retiro:")
6 hourly_traffic.show()
```

Se utilizaron las bibliotecas `Plotly Express` y `Pandas` para visualizar los resultados. Los DataFrames de Spark con los resultados agregados (conteos por estación, tráfico por hora) se convirtieron a DataFrames de Pandas usando `.toPandas()`. **Nota:** Esta operación recolecta todos los datos en el nodo driver y debe usarse con precaución en conjuntos de datos muy grandes. Para incluir las gráficas en este reporte, fueron guardadas como archivos de imagen (e.g., PNG) desde el notebook.

- Se generaron mapas de dispersión geográficos (`px.scatter_mapbox`) para mostrar la ubicación de las estaciones, con el tamaño y color de los puntos representando la cantidad de retiros o arribos.
- Se crearon gráficos de barras (`px.bar`) para visualizar la distribución del tráfico a lo largo de las horas del día.

4. Análisis de los Resultados

El análisis de los datos procesados con Spark reveló varios patrones interesantes sobre el uso de Ecobici:

- **Estaciones Populares:** Al observar los resultados de `station_count.orderBy('count_retiro', ascending=False).show(10)` y `station_count.orderBy('count_arribo', ascending=False).sh` se identificaron las 10 estaciones con mayor número de retiros y arribos, respectivamente. Estas estaciones suelen concentrarse en zonas de alta densidad poblacional,

centros de trabajo u oficinas, y puntos de conexión con otros sistemas de transporte. La visualización en el mapa (Figura 1 y 2) confirma esta concentración geográfica.

- **Horas Pico:** El análisis del tráfico por hora (`hourly_traffic.show()`) mostró picos claros de actividad.
  - **Retiros:** Se observó un pico importante en las mañanas (típicamente entre 7-9 AM) y otro aún mayor en las tardes (alrededor de 5-7 PM). Esto sugiere un uso predominante para traslados hacia y desde el trabajo o estudio (ver Figura 3).
  - **Arribos:** Los picos de arribos siguen una lógica similar pero ligeramente desfasada, con un pico por la mañana (llegadas al destino) y un pico más pronunciado por la tarde/noche (regresos a casa o puntos de origen) (ver Figura 4).

  Los gráficos de barras generados con Plotly visualizaron claramente estas tendencias horarias.

- **Distribución Geográfica:** Los mapas interactivos (Figuras 1 y 2) mostraron que la actividad de Ecobici no es uniforme en toda la red. Hay "hotspotsçlaros donde la demanda (retiros) y la oferta (arribos) son significativamente más altas. Comparar los mapas de retiros y arribos puede indicar desbalances (estaciones donde se acumulan o faltan bicicletas en ciertos horarios).

4.1. Visualizaciones

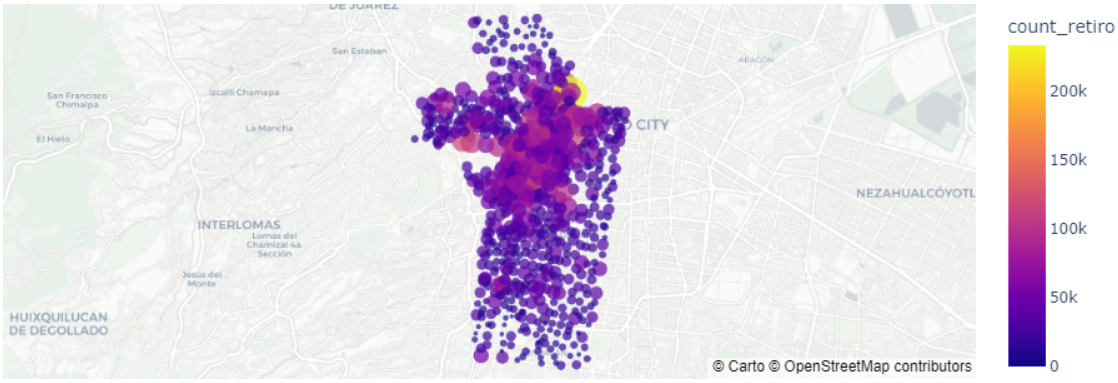

Figura 1: Mapa de calor de retiros por estación. El tamaño y color indican mayor número de retiros.

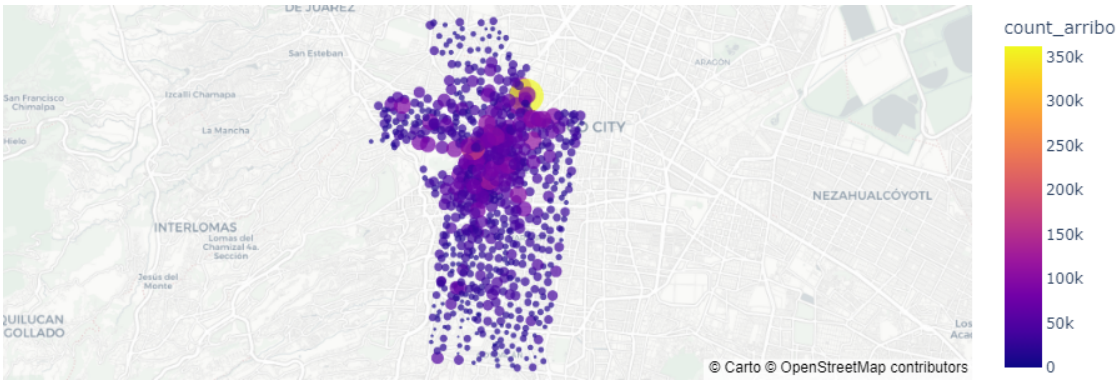

Figura 2: Mapa de calor de arribos por estación. El tamaño y color indican mayor número de arribos.

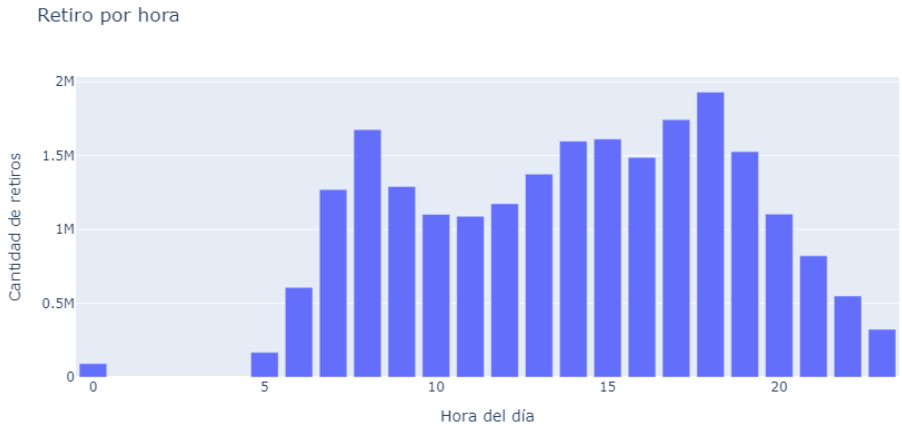

Figura 3: Número total de retiros por hora del día.

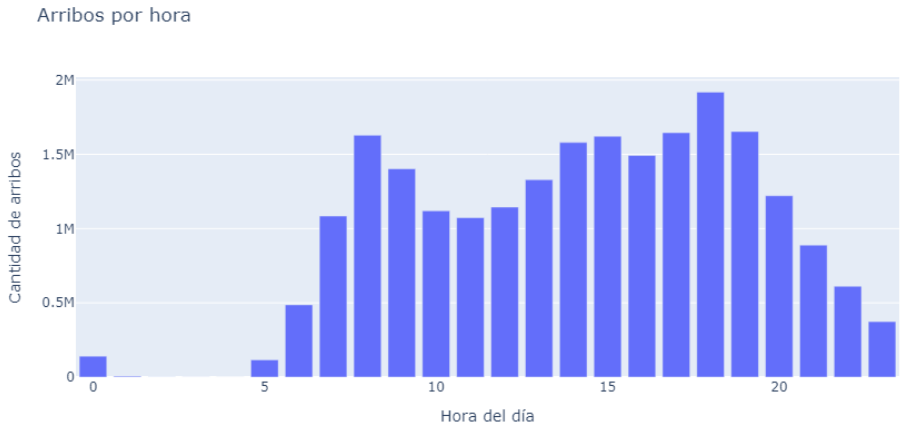

Figura 4: Número total de arribos por hora del día.

5. Reflexión sobre el uso de Spark

Spark está diseñado para el procesamiento distribuido, y muchas de las operaciones realizadas en este análisis se benefician enormemente de esta capacidad:

- **Operaciones Eficientemente Distribuibles:**
  - **Transformaciones por Fila (`withColumn`, `toDF`, `select`):** Operaciones como renombrar columnas, extraer la hora (`hour(to_timestamp(...))`), o seleccionar columnas se aplican a cada partición de datos de forma independiente y paralela.
  - **Agregaciones (`groupBy().count()`):** Estas operaciones son altamente paralelizadas en Spark. Lo que permite tener mejor rendimiento que en pandas.
  - **Uniones (`join`):** Spark puede realizar la unión de manera distribuida, lo que significa que puede combinar grandes conjuntos de datos sin pasar por un único núcleo o dispositivo.
- **Operaciones Menos Distribuibles o Potenciales Cuellos de Botella:**
  - **`.toPandas()`:** Esta operación recolecta *todos* los datos del DataFrame para crear un DataFrame de Pandas en la memoria. Si el DataFrame resultante es muy grande, puede causar errores de memoria o ser extremadamente lento. Es útil para visualización o integración con bibliotecas que no son de Spark.
  - **Lectura de Muchos Archivos Pequeños:** Si bien Spark puede leer múltiples archivos, leer una cantidad masiva de archivos muy pequeños puede ser ineficiente debido a la sobrecarga de iniciar una tarea por cada archivo. En tales casos, sería mejor consolidar los archivos pequeños primero. En este caso, con archivos mensuales, la sobrecarga es probablemente manejable.

6. Uso de IA

En esta práctica, se utilizó IA para generar el texto del reporte y las visualizaciones. La IA ayudó a resumir los resultados. Asimismo, se utilizó IA principalmente para encontrar las equivalencias entre las funciones de pandas y las de Spark, aunque al final no fue tan necesario, ya que la mayoría de funciones son muy similares El código generado se puede consultar en el siguiente repositorio de GitHub: Spark-practice.