72.42 Programación de Objetos Distribuidos

Trabajo Práctico Especial 2: Alquiler de Bicicletas

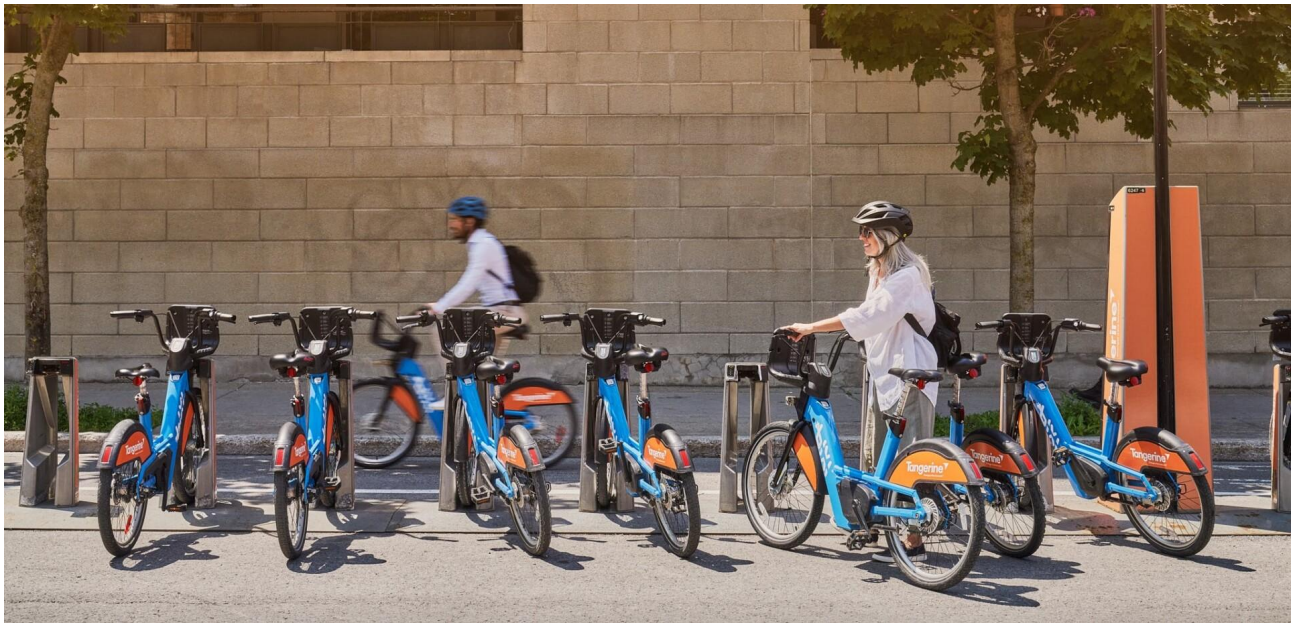
18 de Octubre de 2023

Objetivo

Diseñar e implementar una aplicación de consola que utilice el **modelo de programación MapReduce** junto con el framework **HazelCast** para el procesamiento de **alquileres de bicicletas**, basado en datos reales.

Para trabajo se busca poder procesar datos de bicicletas de la ciudad de Montreal, Canadá.

Los datos son extraídos del portal de gobierno en formato CSV.



Descripción Funcional

A continuación se lista el ejemplo de uso que se busca para la aplicación: procesar los datos de bicicletas de la ciudad de Montreal. Sin embargo, es importante recordar que la mayor parte de la implementación no debe estar atada a la realidad de los ejemplos de uso. **Por ejemplo, las estadísticas serán las que la aplicación obtenga en ejecución a partir del archivo de estaciones de bicicletas y no serán aceptadas implementaciones que tengan fijos estos datos.** En otras palabras, la implementación deberá funcionar también para procesar los datos de bicicletas de cualquier otra ciudad, manteniendo siempre la estructura de los archivos que se presentan a continuación.

Datos de alquileres de bicicletas de Montreal, a partir de ahora `bikes.csv`

- **Origen:** Open Data | BIXI Montréal
- **Descarga:** `/afs/it.itba.edu.ar/pub/pod/bikes.csv`
- **Cantidad de registros:** `14.266.436`
- **Campos:**
  - **`start_date`:** Fecha y hora del alquiler de la bicicleta (inicio del viaje) en formato yyyy-MM-dd HH:mm:ss
  - **`emplacement_pk_start`:** Identificador de la estación de inicio (número entero)
  - **`end_date`:** Fecha y hora de la devolución de la bicicleta (fin del viaje) en formato yyyy-MM-dd HH:mm:ss
  - **`emplacement_pk_end`:** Identificador de la estación de fin (número entero)
  - **`is_member`**: Si el usuario del alquiler es miembro del sistema de alquiler (0 si no es miembro, 1 si lo es)

El archivo se compone de una primera línea de encabezado, con los títulos de cada campo. De la segunda línea en adelante, **cada línea representa un viaje** conteniendo los datos de cada uno de los campos, separados por ";". Por ejemplo:

```
start_date;emplacement_pk_start;end_date;emplacement_pk_end;is_member
2021-09-20 06:31:28;348;2021-09-20 07:02:22;332;1
2022-09-02 11:19:47;753;2022-09-02 11:22:23;702;0
...
```

Datos de estaciones de bicicletas de Montreal, a partir de ahora `stations.csv`

- **Descarga:** `/afs/it.itba.edu.ar/pub/pod/stations.csv`
- **Cantidad de registros:** `996`
- **Campos:**
  - **`pk`:** Identificador de la estación (número entero)
  - **`name`:** Nombre de la estación (cadena de caracteres)
  - **`latitude`:** Latitud de la ubicación de la estación (numéro real)
  - **`longitude`:** Longitud de la ubicación de la estación (número real)

El archivo se compone de una primera línea de encabezado. De la segunda línea en adelante, **cada línea representa una estación**. Por ejemplo:

```
pk;name;latitude;longitude
327;Sanguinet / de Maisonneuve;45.513405;-73.562594
267;Gilford / de Brébeuf;45.530816;-73.581626
...
```

Se asume que el formato y contenido de los archivos es correcto.

72.42 Programación de Objetos Distribuidos

Requerimientos

La aplicación debe poder resolver un conjunto de consultas listadas más abajo. En cada una de ellas se indicará un ejemplo de invocación con un script propio para correr únicamente esa query con sus parámetros necesarios.

Cada corrida de la aplicación resuelve sólo una de las queries sobre los datos obtenidos a partir de los archivos CSV provistos en esa invocación (archivos CSV de estaciones y de alquileres).

La respuesta a la *query* quedará en un archivo de salida CSV.

Para medir performance, se deberán escribir en otro archivo de salida los *timestamp* de los siguientes momentos:

- Inicio de la lectura de los archivos de entrada
- Fin de lectura de los archivos de entrada
- Inicio de un trabajo MapReduce
- Fin de un trabajo MapReduce (incluye la escritura del archivo de respuesta)

Todos estos momentos deben ser escritos en la salida luego de la respuesta con el timestamp en formato: `dd/mm/yyyy hh:mm:ss:xxxx` y deben ser claramente identificables.

Ejemplo del archivo de tiempos:

```
17/05/2023 14:43:09:0223 INFO  [main] Client (Client.java:76) - Inicio de la
lectura del archivo
17/05/2023 14:43:23:0011 INFO  [main] Client (Client.java:173) - Fin de lectura
del archivo
17/05/2023 14:43:23:0013 INFO  [main] Client (Client.java:87) - Inicio del
trabajo map/reduce
17/05/2023 14:43:23:0490 INFO  [main] Client (Client.java:166) - Fin del
trabajo map/reduce
```

Por ejemplo:

```
$> ./queryX -Daddresses='xx.xx.xx.xx:XXXX;yy.yy.yy.yy:YYYY' -DinPath=XX
-DoutPath=YY [params]
```

donde

- `queryX` es el script que corre la query `X`.
- `-Daddresses` refiere a las direcciones IP de los nodos con sus puertos (una o más, separadas por punto y coma)
- `-DinPath` indica el path donde están los archivos de entrada **`bikes.csv`** y **`stations.csv`**.
- `-DoutPath` indica el path donde estarán ambos archivos de salida **`query1.csv`** y **`time1.txt`**.
- `[params]`: los parámetros extras que corresponden para algunas queries.

De esta forma,

```
$> ./query1 -Daddresses='10.6.0.1:5701;10.6.0.2:5701'
-DinPath=/afs/it.itba.edu.ar/pub/pod/
-DoutPath=/afs/it.itba.edu.ar/pub/pod-write/l12345/
```

resuelve la *query* 1 a partir de los datos presentes en `/afs/it.itba.edu.ar/pub/pod/stations.csv` y `/afs/it.itba.edu.ar/pub/pod/bikes.csv` utilizando los nodos `10.6.0.1` y `10.6.0.2` para su procesamiento. Se crearán los archivos `/afs/it.itba.edu.ar/pub/pod-write/l12345/query1.csv` y `/afs/it.itba.edu.ar/pub/pod-write/l12345/time1.txt` que contendrán respectivamente el resultado de la *query* y los *timestamp* de inicio y fin de la lectura del archivo y de los trabajos map/reduce.

Query 1: Viajes entre estaciones

Donde cada línea de la salida contenga, separados por ";" el **nombre de la estación A, el nombre de la estación B** y **la cantidad total de viajes iniciados en la estación A y finalizados en la estación B**.

El orden de impresión es **descendente por cantidad total de viajes** y desempata **alfabético** por el nombre de la estación A y luego **alfabético** por nombre de la estación B.

Sólo se deben listar las estaciones presentes en el archivo CSV de estaciones.

No se deben listar los pares de estaciones que no tengan viajes entre ellas, es decir, salidas que contengan `0` en la última columna.

No se deben listar los viajes circulares, es decir, salidas que contengan el mismo nombre de estación para las dos primeras columnas.

- Parámetros adicionales: Ninguno
- Ejemplo de invocación: `./query1 -Daddresses='10.6.0.1:5701' -DinPath=. -DoutPath=.`
- Salida de ejemplo:

```
station_a;station_b;trips_between_a_b
Métro Mont-Royal (Rivard / du Mont-Royal);Marquette / du Mont-Royal;346
de la Commune / St-Sulpice;de la Commune / Place Jacques-Cartier;296
Marquette / du Mont-Royal;Métro Mont-Royal (Rivard / du Mont-Royal);296
Métro Joliette  (Joliette / Hochelaga);Aylwin / Ontario;270
...
```

Query 2: Top N estaciones con mayor promedio de distancia aproximada

Donde cada línea de la salida contenga, separados por ";" el **nombre de la estación** y el **promedio de la distancia aproximada** (en km) del total de **viajes iniciados por miembros** en esa estación que **finalizan en una estación distinta**.

Para calcular la **distancia aproximada** (en km) de un viaje considerar la distancia en línea recta entre una estación y otra. Utilizar la fórmula de Haversine.

El **promedio de la distancia aproximada** consiste en la suma de las distancias aproximadas de los viajes iniciados en la estación dividido por la cantidad de viajes iniciados en la estación.

Sólo se deben listar las estaciones presentes en el archivo CSV de estaciones.

No se deben listar las estaciones que no tengan viajes iniciados, es decir, las que implicarían una división por cero.

El orden de impresión es descendente por el promedio de distancia aproximada y luego alfabético por nombre de la estación de inicio del viaje.

Sólo se deben listar los N con mayor promedio, donde N se recibe por parámetro.

El formato de impresión del promedio de distancia aproximada debe ser truncado a dos decimales.

- Parámetros adicionales:
  - **n** límite de cantidad de resultados (número entero)

- Ejemplo de invocación: `./query2 -Daddresses='10.6.0.1:5701' -DinPath=. -DoutPath=. -Dn=4`

- Salida de ejemplo:

```
station;avg_distance
Parc Dixie (54e avenue / Dixie);9.35
11e avenue / du Souvenir;7.78
Parc Henri-Durant (Léger / Henri-Durant);7.15
Parc Langelier - Marie-Victorin;7.03
```

Query 3: Viaje más largo (en minutos) por estación

Donde cada línea de la salida contenga, separados por ";" la **estación de inicio** del viaje, la **estación de destino** del viaje, **la fecha y hora de inicio** del viaje y la duración en minutos de ese viaje más largo.

Sólo se deben listar las estaciones presentes en el archivo CSV de estaciones.

Sólo se deben considerar los viajes que tengan estaciones de inicio y destino distintas.

El orden de impresión es descendente por la cantidad de minutos del viaje más largo y luego alfabético por nombre de la estación de inicio del viaje.

En caso de que, para una estación de origen, existan dos o más viajes más largos con la misma cantidad de minutos quedarse con el viaje que tenga la fecha de inicio más reciente.

El formato de impresión de las fechas debe ser DD/MM/YYYY HH:mm:ss.

- Parámetros adicionales: Ninguno

- Ejemplo de invocación: `./query3 -Daddresses='10.6.0.1:5701' -DinPath=. -DoutPath=.`

- Salida de ejemplo:

```
start_station;end_station;start_date;minutes
Molson / Beaubien;Chabot / de Bellechasse;10/04/2021 16:48:15;2782
Laval / Rachel;Crescent / René-Lévesque;12/04/2021 07:44:58;1774
St-Dominique / Napoléon;Ropery / Augustin-Cantin;17/04/2021 15:06:45;1327
Chambord / Jean-Talon;Alexandre-DeSève / Ste-Catherine;15/06/2021 14:04:07;1234
...
```

Query 4: Días de afluencia neta positiva, neutra y negativa por estación

Donde cada línea de la salida contenga, separados por ";" el **nombre de la estación**, la **cantidad de días con afluencia neta positiva** de esa estación, la **cantidad de días con afluencia neta neutra** de esa estación y la **cantidad de días con afluencia neta negativa** de esa estación, dentro de un rango de fechas que se indica por parámetro.

La **afluencia neta** de una estación consiste en el total de bicicletas que llegaron a la estación (viajes que finalizan en la estación) menos el total de bicicletas que salieron de la estación (viajes que inician en la estación).

La suma de los tres valores de afluencia neta para cada estación debe coincidir con el total de días comprendidos en el rango de fechas que se indica por parámetro. Si para un día en una estación no salieron ni llegaron bicicletas entonces se contabiliza como afluencia neta neutra.

El orden de impresión es descendente por la cantidad de afluencia neta positiva y luego alfabético por nombre de la estación.

Sólo se deben listar las estaciones presentes en el archivo CSV de estaciones.

- Parámetros adicionales:
  - **`startDate`** la fecha de inicio del rango en formato DD/MM/YYYY
  - **`endDate`** la fecha de fin del rango en formato DD/MM/YYYY

El rango es `[startDate, endDate]`, es decir, arranca a las 0 horas de `startDate` y finaliza a las 23:59 inclusive de `endDate`.

- Ejemplo de invocación: `./query4 -Daddresses='10.6.0.1:5701' -DinPath=. -DoutPath=. -DstartDate=01/05/2021 -DendDate=31/05/2021`

- Salida de ejemplo (con un rango de 31 días):

```
station;pos_afflux;neutral_afflux;negative_afflux
W 21 St & 6 Ave;15;3;13
Laurier / de Brébeuf;13;9;9
Métro Atwater (Atwater / Ste-Catherine);13;15;3
du Mont-Royal / Clark;11;4;16
Boyer / du Mont-Royal;7;20;4
...
```

Muy Importante:

> ➔ **Respetar exactamente los nombres de los *scripts*, los nombres de los archivos de entrada y salida y el orden y formato de los parámetros del *scripts*.**
>
> ➔ En todos los `pom.xml` que entreguen deberán definir el `artifactId` de acuerdo a la siguiente convención: "`tpe2-gX-Z`" donde X es el número de grupo y Z es `parent`, `api`, `server` o `client`. Por ejemplo: **`<artifactId>`** `tpe2-g5-api` **`</artifactId>`**

> ➔ En todos los `pom.xml` que entreguen deberán incluir el tag `name` con la siguiente convención: "`tpe2-gX-Z`" donde X es el número de grupo y Z es `parent`, `api`, `server` o `client`. Por ejemplo: **`<name>`**`tpe2-g5-api`**`</name>`**
>
> ➔ Utilizar la versión **3.8.6** de **`hazelcast-all`**
>
> ➔ **El nombre del cluster (`<group><name>`) y los nombres de las colecciones de Hazelcast** a utilizar en la implementación deben comenzar con "g" seguido del número de grupo. Por ej `g5` para así evitar conflictos con las colecciones y poder hacer pruebas de distintos alumnos en simultáneo utilizando la misma red.
>
> ➔ La implementación debe **respetar exactamente el formato de salida enunciado**. Tener en cuenta que los archivos de salida deben contener las líneas de encabezado correspondientes indicadas en las salidas de ejemplo para todas las *queries*.

Condiciones del trabajo práctico

- **El trabajo práctico debe realizarse en los mismos grupos formados para el primer trabajo práctico especial.**
- Cada una de las opciones debe ser implementada **con uno o más *job*s MapReduce** que pueda correr en un ambiente distribuido utilizando un *grid* de Hazelcast.
- Los componentes del *job*, clases del modelo, tests y el diseño de cada elemento del proyecto queda a criterio del alumno, pero debe estar enfocado en:
    - Que funcione correctamente en un ambiente concurrente MapReduce en Hazelcast.
    - Que sea eficiente para un gran volumen de datos, particularmente en tráfico de red.
    - Mantener buenas prácticas de código como comentarios, reutilización, legibilidad y mantenibilidad.

Material a entregar

Cada alumno deberá subir al **Campus ITBA** un archivo compactado conteniendo:

- El **código fuente** de la aplicación:
    - Utilizando el arquetipo de Maven utilizado en las clases.
    - Con una correcta separación de las clases en los módulos *api*, *client* y *server*.
    - Un README indicando cómo preparar el entorno a partir del código fuente para ejecutar la aplicación en un ambiente con varios nodos.
    - No se deben entregar los binarios.
- Un **documento breve** (no más de dos carillas) explicando:
    - Cómo se diseñaron los componentes de cada trabajo MapReduce, qué decisiones se tomaron y con qué objetivos. Además alguna alternativa de diseño que se evaluó y descartó, comentando el porqué.
    - El análisis de los tiempos para la resolución de cada query: En caso de poder, analizar la diferencia de tiempos de correr cada query aumentando la cantidad de

nodos (hasta 5 nodos) en una red local. De no poder, intentar predecir cómo sería el comportamiento.
    - Potenciales puntos de mejora y/o expansión.
    - La comparación de los tiempos de las queries ejecutándose con y sin *Combiner*.
    - Otro análisis de tiempos de ejecución de las queries utilizando algún otro elemento de optimización a elección por el alumno.
    - Para todos los puntos anteriores, no olvidar de indicar el tamaño de los archivos utilizados como entrada para las pruebas (cantidad de registros).
- La **historia de Git**: El directorio oculto .git/ donde se detallan todas las modificaciones realizadas.

Corrección

El trabajo no se considerará aprobado si:

- No se entregó el trabajo práctico en tiempo y forma.
- Faltan algunos de los materiales solicitados en la sección anterior.
- El código no compila utilizando Maven en consola (de acuerdo a lo especificado en el README a entregar).
- El servicio no inicia cuando se siguen los pasos del README.
- Los clientes no corren al seguir los pasos del README.

Si nada de esto se cumple, se procederá a la corrección donde se tomará en cuenta:

- Que los procesos y queries funcionen correctamente según las especificaciones dadas.
- El resultado de las pruebas y lo discutido en el coloquio.
- La aplicación de los temas vistos en clase: Concurrencia y Hazelcast.
- La modularización, diseño testeo y reutilización de código.
- El contenido y desarrollo del informe.

Uso de Git

Es obligatorio el uso de un repositorio Git para la resolución de este TPE. No se aceptarán entregas que utilicen un repositorio git con un único *commit* que consista en la totalidad del código a entregar.

Los distintos *commits* deben permitir ver la evolución individual del trabajo.

Muy importante: **los repositorios creados deben ser privados, solo visibles para los autores y la cátedra en caso de que se lo solicite específicamente.**

Cronograma

- **Presentación del Enunciado: <u>miércoles 18/10</u>**
- **Entrega del trabajo: Estará disponible hasta el <u>jueves 02/11 a las 23:59</u>** la actividad "Entrega TPE 2" localizada en la sección Contenido / Evaluación / Entrega TPE 2. En la misma deberán cargar el paquete con el **código fuente** y el **documento**

72.42 Programación de Objetos Distribuidos

- **El día <u>miércoles 15/11 a las 18:00</u>** cada grupo tendrá un espacio para un coloquio. Durante el mismo se les hará una devolución del trabajo, indicando la corrección y los principales errores cometidos. A criterio de la cátedra también se podrán realizar preguntas sobre la implementación. Opcionalmente se les podrá solicitar la ejecución de la aplicación a la cátedra para revisar el funcionamiento de alguna funcionalidad. Para ello es necesario que un alumno del grupo tenga el cluster “levantado”.
- **El día del recuperatorio será el <u>miércoles 22/11</u>.**
- **<u>No se aceptarán entregas pasado el día y horario establecido como límite.</u>**

Dudas sobre el TPE

Las mismas deben volcarse en los **Debates** del Campus ITBA.

Recomendaciones

- **Clases útiles para consultar**
  - **Hazelcast**
    - `com.hazelcast.mapreduce.`**`Combiner`**
    - `com.hazelcast.mapreduce.`**`Collator`**
  - **Java**
    - `java.nio.file.`**`Files.lines`**
    - `java.text.`**`DecimalFormat`** y `java.math.`**`RoundingMode`**
    - `java.time.format.`**`DateTimeFormatter`**
    - `java.time.`**`Duration`**