**Paradigma NoSQL**

Trabajo Práctico Especial

Grupo 1

Santiago Ocamica - 53346

Nicolás Castaño - 53384

15 de noviembre de 2017

# 

[**I. Entorno**](#_qpz8bpy5uxl4) **2**

[**II. Implementación**](#_vgwzkdfflhc5) **3**

[Q](#_z2rs5jiq0bu5)1\_1 3

[Q](#_tfk59q1r9tt)1\_2 3

[Q](#_yoygwvsrailc)1\_3 4

[Q](#_btm9doq17aiu)1\_4 4

[Q](#_tjb3j4r5ffas)1\_5 4

[Q](#_ozquxxvccpfj)1\_6 5

[Q](#_hzg76nugvf3b)2\_1 5

[**III. Tiempos de consultas**](#_j0xzo1iaa6lg) **6**

[Tiempo de Subida y Ejecución](#_nudrqbyd3k0n) 6

[Tiempo de Ejecución](#_fjlgon3xp5vy) 7

[**IV. Análisis de resultados**](#_et7ij58dml52) **9**

[Q](#_fd72k89gco8p)1\_1 9

[Q](#_7jcu8t9nzaju)1\_2 9

[Q](#_3kuvzfpdx5uk)1\_3 9

[Q](#_jkmfgh4cx5ha)1\_4 9

[Q](#_2opsm9azu4xp)1\_5 9

[Q](#_lbxesh8pl8kj)1\_6 10

[Q](#_kxr66ca5swjh)2\_1 10

[**V. Alternativas no utilizadas**](#_ojz5wteu4m60) **11**

[**VI.**](#_ojz5wteu4m60) **Conclusiones 11**

# 

# 

# **I. Entorno**

## Generación de Datos

Escribimos un script en **ruby** que genera datos aleatorios y los serializa en dos archivos csv, correspondientes a cada una de las tablas de nuestra base de datos relacional: **telefonos** y **llamadas**.

Dicho script se encuentra en data\_generator/build\_data.rb

Para ejecutar el mismo se deben modificar las constantes CLIENT\_COUNT y DAYS\_TO\_GENERATE, para elegir la cantidad de clientes a generar y la cantidad de dias a simular respectivamente.

Luego ejecutar:

ruby data\_generator/build\_data.rb

Al finalizar el script se habrán creado los archivos llamadas.csv y telefonos.csv con los datos deseados.

## GraphFrames

Escribimos un programa de Java capaz de generar, persistir y cargar grafos basados en los datos generados con el script mencionado anteriormente. El mismo se encuentra en la carpeta graphframes.

Antes de ejecutar el mismo se pueden modificar algunas características de su funcionamiento:

* CALLS\_FILE\_LOCATION: La ubicación (hdfs) del csv que contiene las llamadas
* PHONES\_FILE\_LOCATION: La ubicación (hdfs) del csv que contiene los telefonos
* VERTICES\_PARQUET\_LOCATION: La ubicación en la que persistir o de la que cargar los vertices.
* EDGES\_PARQUET\_LOCATION: La ubicación en la que persistir o de la que cargar las aristas.
* BUILD\_GRAPH: Booleano que determina si se debe construir un nuevo grafo
* QUERY\_NUMBER: Un string con formato “1\_1” que indica la query a ejecutar

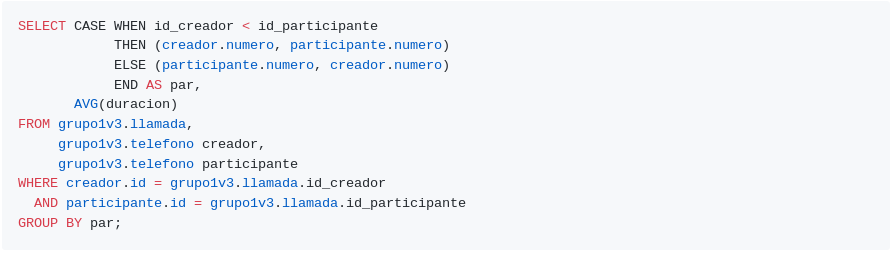
Una vez configurado se puede generar un JAR del mismo con:  
mvn package

# **II. Implementación**

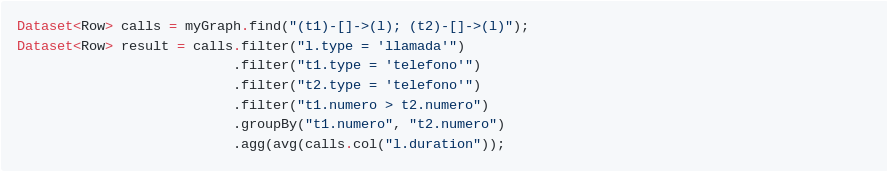
## Q1\_1

*Por cada* ***par de números de teléfonos,*** *calcular la duración promedio de las llamadas en las que ellos participaron,* ***sin diferenciar*** *quien inició la llamada*

//Postgre



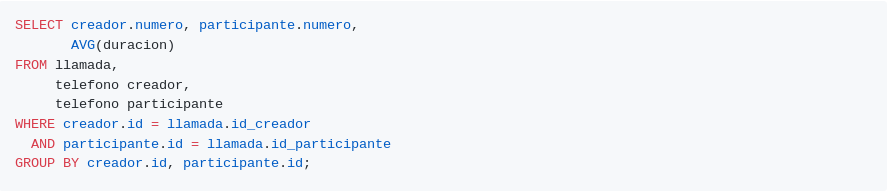
//GraphFrames



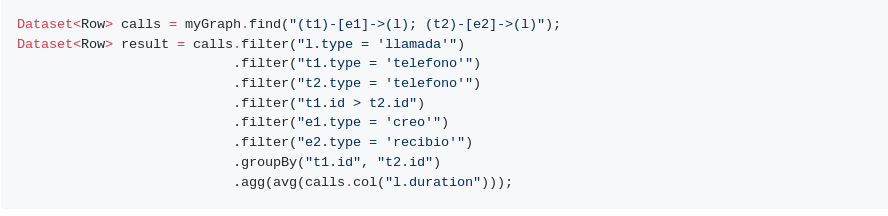
## Q1\_2

*Por cada* ***par de números de teléfonos*** *calcular la duración* ***promedio*** *de las llamadas en las que ellos participaron,* ***diferenciando*** *quien inició la llamada*

//Postgre



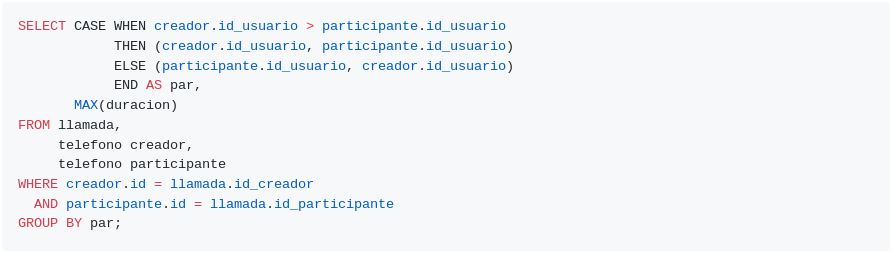
//GraphFrames



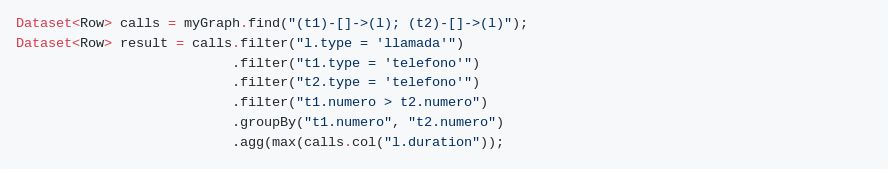
## Q1\_3

*Por cada* ***par de personas*** *(rollup) calcular la* ***máxima duración*** *entre las llamadas en las que ellos participaron,* ***sin diferenciar*** *quien inició la llamada*

//Postgre



/GraphFrames

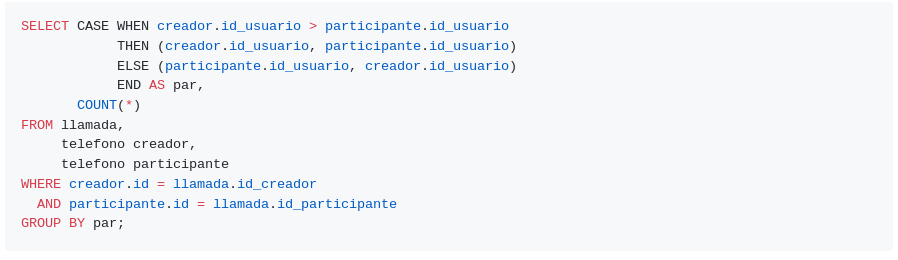


## 

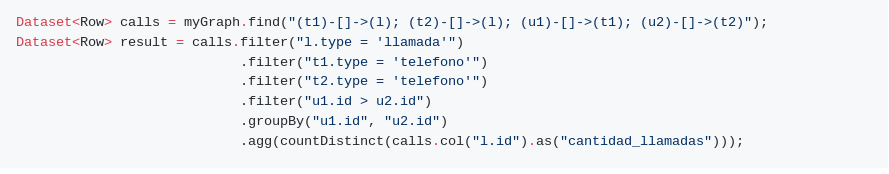
## Q1\_4

*Por cada* ***par de personas*** *(rollup) calcular la* ***cantidad de llamadas*** *en las que ellos participaron,* ***sin diferenciar*** *quien inició la llamada*

//Postgre



//GraphFrames



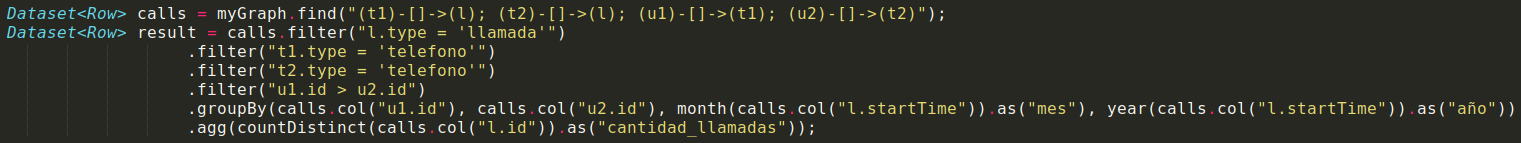
## Q1\_5

*Por cada* ***par de personas*** *(rollup) y* ***por cada mes****, calcular la* ***cantidad de llamadas*** *en las que ellos participaron,* ***sin diferenciar*** *quien inició la llamada*

//Postgre



//GraphFrames



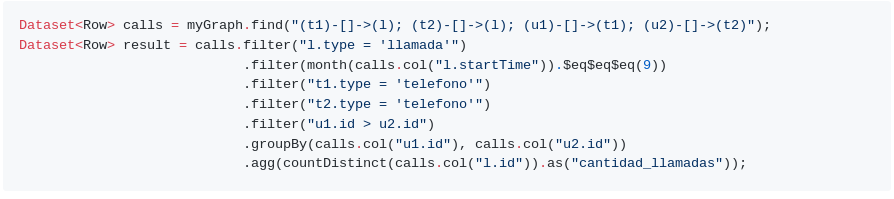
## Q1\_6

*Por cada* ***par de personas*** *(rollup) y* ***sólo para el mes de Abril-2017****, sacar la* ***cantidad de llamadas*** *en las que ellos participaron,* ***sin diferenciar*** *quien inició la llamada*

//Postgre



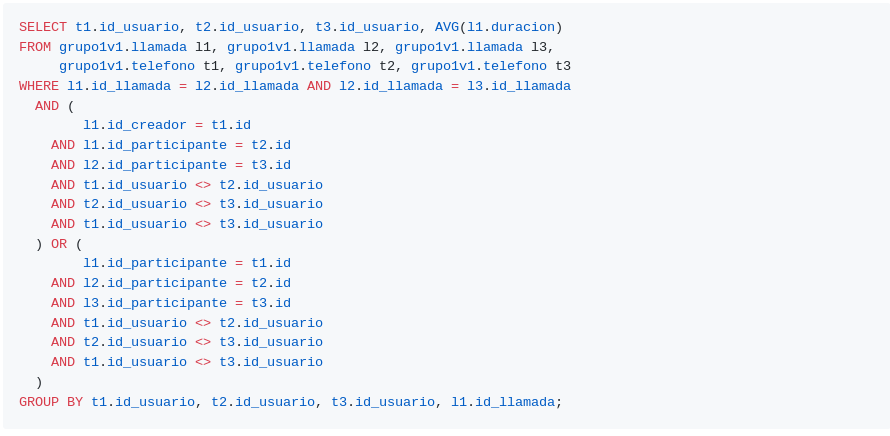
//GraphFrames



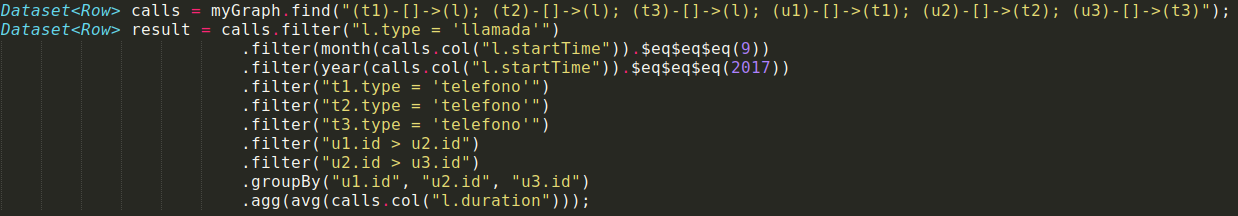
## Q2\_1

*Para* ***Abril del 2017*** *indicar por cada* ***tres personas*** *(rollup) que estuvieron comunicadas, el promedio de la duración de la comunicación. Ignorar quien inició la comunicación. En este caso hay que considerar todos los números de teléfono que poseen cada persona.*

//Postgre



//GraphFrames



# 

# **III. Tiempos de consultas**

Se decidió analizar tanto los tiempos de ejecución de las consultas en POSTGRE como usando GraphFrames. También, se compararon los resultados

**Consideraciones:**

* Tiempo medido en milisegundos

### Tiempo de Ejecución Postgre

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Query** | **Esquema grupo1v3** | | | | | |
| **Corrida 1** | **Corrida 2** | **Corrida 3** | **Corrida 4** | **Corrida 5** | **Promedio** |
| **Q1\_1** | 105852 ms | 106571 ms | 112269 ms |  |  |  |
| **Q1\_2** | 61015 ms | 68559 ms | 61259 ms | 60587 ms | 60466 ms | 62377 ms |
| **Q1\_3** | 181388ms | 181495ms | 181847 ms | 181507 ms | 181325 ms | 181512.4 ms |
| **Q1\_4** | 170380ms | 170613 ms | 170057 ms | 170618 ms | 170097 ms | 170353 ms |
| **Q1\_5** | 187434 ms | 189573 ms | 181070 ms | 181275 ms | 181603 ms | 184191 ms |
| **Q1\_6** | 3403 ms | 3394 ms | 3419 ms | 3399 ms | 3421 ms | 3407.2 ms |
| **Q2\_1** |  |  |  |  |  |  |

### 

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Query** | **Esquema grupo1v2** | | | | | |
| **Corrida 1** | **Corrida 2** | **Corrida 3** | **Corrida 4** | **Corrida 5** | **Promedio** |
| **Q1\_1** | 28275 ms | 36389 ms | 17684 ms | 17625 ms | 17556 ms | 23505.8 ms |
| **Q1\_2** | 4043 ms | 4055 ms | 4300 ms | 6200 ms | 4727 ms | 4665 ms |
| **Q1\_3** | 17458 ms | 19879 ms | 22414 ms | 24262 ms | 22088 ms | 21220.2 ms |
| **Q1\_4** | 9689.209 ms | 9671.168 ms | 9668.081 ms | 9650.771 ms | 9692.031 ms | 9674.252 ms |
| **Q1\_5** | 10567.791 ms | 10570.365 ms | 10636.139 ms | 10494.910 ms | 10524.496 ms | 10559.34 ms |
| **Q1\_6** | 239.258 ms | 236.806 ms | 237.274 ms | 240.662 ms | 236.850 ms | 238.17 ms |
| **Q2\_1** |  |  |  |  |  |  |

### 

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Query** | **Esquema grupo1v1** | | | | | |
| **Corrida 1** | **Corrida 2** | **Corrida 3** | **Corrida 4** | **Corrida 5** | **Promedio** |
| **Q1\_1** | 5801.385 ms | 5883.439 ms | 5785.971 ms | 5801.385 ms | 9543.925 ms | 6563.221ms |
| **Q1\_2** | 1478.804 ms | 1468.506 ms | 1477.688 ms | 1474.273 ms | 1474.499 ms | 1474.754ms |
| **Q1\_3** | 3726.179 ms | 3706.218 ms | 3709.965 ms | 3727.167 ms | 3726.463 ms | 3719.40 ms |
| **Q1\_4** | 1724.134 ms | 3333.693 ms | 1605.346 ms | 1594.223 ms | 1599.646 ms | 1971.4084ms |
| **Q1\_5** | 1799.306 ms | 1765.857 ms | 1783.285 ms | 1768.942 ms | 1762.606 ms | 1776 ms |
| **Q1\_6** | 85.083 ms ms | 51.094 ms | 50.581 ms | 51.257 ms | 50.596 ms | 57.72 ms |
| **Q2\_1** |  |  |  |  |  |  |

### Tiempo de Ejecución GraphFrames

### 

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Query** | **Esquema grupo1v3** | | | | | |
| **Corrida 1** | **Corrida 2** | **Corrida 3** | **Corrida 4** | **Corrida 5** | **Promedio** |
| **Q1\_1** |  |  |  |  |  |  |
| **Q1\_2** |  |  |  |  |  |  |
| **Q1\_3** |  |  |  |  |  |  |
| **Q1\_4** |  |  |  |  |  |  |
| **Q1\_5** |  |  |  |  |  |  |
| **Q1\_6** |  |  |  |  |  |  |
| **Q2\_1** |  |  |  |  |  |  |

### 

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Query** | **Esquema grupo1v2** | | | | | |
| **Corrida 1** | **Corrida 2** | **Corrida 3** | **Corrida 4** | **Corrida 5** | **Promedio** |
| **Q1\_1** |  |  |  |  |  | 1049 ms |
| **Q1\_2** | 1098 ms | 375 ms | 286 ms | 260 ms | 234 ms | 453.2 ms |
| **Q1\_3** | 1059 ms | 337 ms | 268 ms | 246 ms | 211 ms | 424.8 ms |
| **Q1\_4** | 1430 ms | 543 ms | 458 ms | 409 ms | 377 ms | 644.0 ms |
| **Q1\_5** | 1461 ms | 560 ms | 465 ms | 389 ms | 376 ms | 652.2 ms |
| **Q1\_6** | 1472 ms | 560 ms | 490 ms | 505 ms | 439 ms | 693.2 ms |
| **Q2\_1** | 1966 ms | 888 ms | 784 ms | 690 ms | 476 ms | 961.2 ms |

### 

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Query** | **Esquema grupo1v1** | | | | | |
| **Corrida 1** | **Corrida 2** | **Corrida 3** | **Corrida 4** | **Corrida 5** | **Promedio** |
| **Q1\_1** |  |  |  |  |  | 1115 ms |
| **Q1\_2** |  |  |  |  |  | 1140 ms |
| **Q1\_3** |  |  |  |  |  | 1030 ms |
| **Q1\_4** |  |  |  |  |  | 1410 ms |
| **Q1\_5** |  |  |  |  |  | 1470 ms |
| **Q1\_6** |  |  |  |  |  | 1516 ms |
| **Q2\_1** | 2025 ms | 2066 ms | 2032 ms | 2028 ms | 1972 ms | 2025 ms |

# 

# 

# **IV. Análisis de resultados**

Para cantidades de datos pequeñas, el overhead de tener la información distribuida en un cluster hace que una base de datos relacional (PostgreSQL) dé mejores resultados que GraphFrames.

Cuando la cantidad de datos aumenta, el beneficio del procesamiento distribuido y el modelado como grafos se percibe y hace que GraphFrames dé resultados de queries mucho más rápidos en genera.

En el caso de la query Q2\_1, en la base de datos relacional no termina (en dos horas) ni siquiera para el esquema con menor cantidad de datos.

# **V. Alternativas no utilizadas**

* GraphX: Se descartó porque su Java API es poco amigable y requería mucho código para armar definir el modelo del grafo
* JanusGraph: No se utilizó porque al principio no pudimos encontrar buena documentación sobre la API de Java

# **VI. Conclusiones**