Universidad
Rey Juan Carlos

Escuela Técnica Superior
de Ingeniería Informática

Grado en Ingeniería de Computadores

Curso 2021-2022

Trabajo de Fin de Grado

Comparativa entre las API de Spark de Scala y Python

Autor: Oscar Nydza Nicpoñ
Tutor: Juan Manuel Serrano Hidalgo

Indice

- Introducción
- Objetivos
- Descripción técnica
  - Programación de queries
  - Visualización de resultados
  - Migración de código
- Comparativa
- Conclusión

Introducción

¿Qué es Spark?

De Wikipedia:

"Apache Spark se puede considerar un sistema de computación en clúster de propósito general y orientado a la velocidad. Proporciona APIs en Java, Scala, Python y R."

Introducción

¿Qué API usamos?

Dos categorías:

- Nativas: Scala y Java
- No nativas: Python y R

Compararemos las API
de Scala y Python

Objetivos

Principal: Comparar el rendimiento entre APIs

Derivados:

- Programar queries en ambas APIs
- Visualizar los resultados obtenidos de las consultas
- Migrar queries de una API a otra

Objetivos

Orden a seguir:

1. Programar queries en ambas APIs
2. Visualizar los resultados obtenidos de las consultas
3. Migrar queries de una API a otra
4. Comparar el rendimiento entre APIs

Descripción técnica: Conjunto de datos

Se va a utilizar un conjunto de datos de la Fórmula 1

Las cuestiones sobre las que buscamos respuestas son:

- Piloto más consistente en la temporada 2012
- Mejor piloto de la historia
- Fabricante más dominante en un periodo de tiempo
- Análisis de temporada por piloto
- Temporada más interesante para el espectador

Descripción técnica: Programación

Ejemplo de código en Scala:

```scala
val lastRace = Window.partitionBy("year")

val lastRaces = spark.read.format("csv")
    .option("header", "true")
    .option("sep", ",")
    .load("../data/races.csv")
    .where(col("year") >= 1990 && col("year") <= 1999)
    .withColumn("round", col("round").cast(IntegerType))
    .withColumn("max", max(col("round")).over(lastRace))
    .where(col("round") === col("max"))
    .select("raceId", "year")
```

Descripción técnica: Programación

Ejemplo de código en Scala:

```scala
val constructors = Window.partitionBy("constructorId")

val constructorWinners = spark.read.format("csv")
    .option("header", "true")
    .option("sep", ",")
    .load("../data/constructor_standings.csv")
    .join(lastRaces, Seq("raceId"), "right")
    .where(col("position") === 1)
    .select("constructorId", "wins", "year")
    .withColumn("totalChampWins", count(col("constructorId")).over(constructors))
    .withColumn("totalRaceWins", sum(col("wins")).over(constructors).cast(IntegerType))
    .drop("wins")
```

Descripción técnica: Programación

Ejemplo de código en Python:

```python
lastRace = Window.partitionBy("year")

lastRaces = spark.read.format("csv")\
    .option("header", "true")\
    .option("sep", ",")\
    .load("../data/races.csv")\
    .where((F.col("year") >= 1990) & (F.col("year") <= 1999))\
    .withColumn("round", F.col("round").cast(T.IntegerType()))\
    .withColumn("max", F.max(F.col("round")).over(lastRace))\
    .where(F.col("round") == F.col("max"))\
    .select("raceId", "year")
```

Descripción técnica: Visualización

Utilizaremos Plotly como herramienta de visualización

- Uso sencillo
- Disponible para Python y Scala
- Podemos hacerlo funcionar con datos obtenidos de Spark

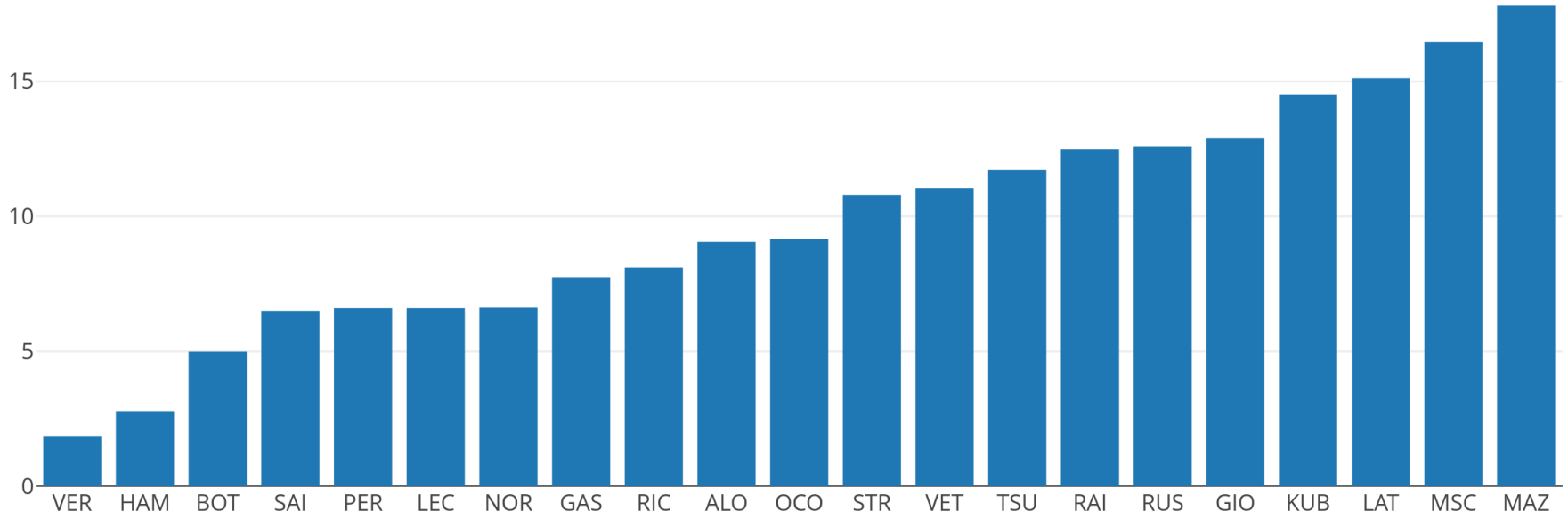
Descripción técnica: Visualización

Procedimiento:

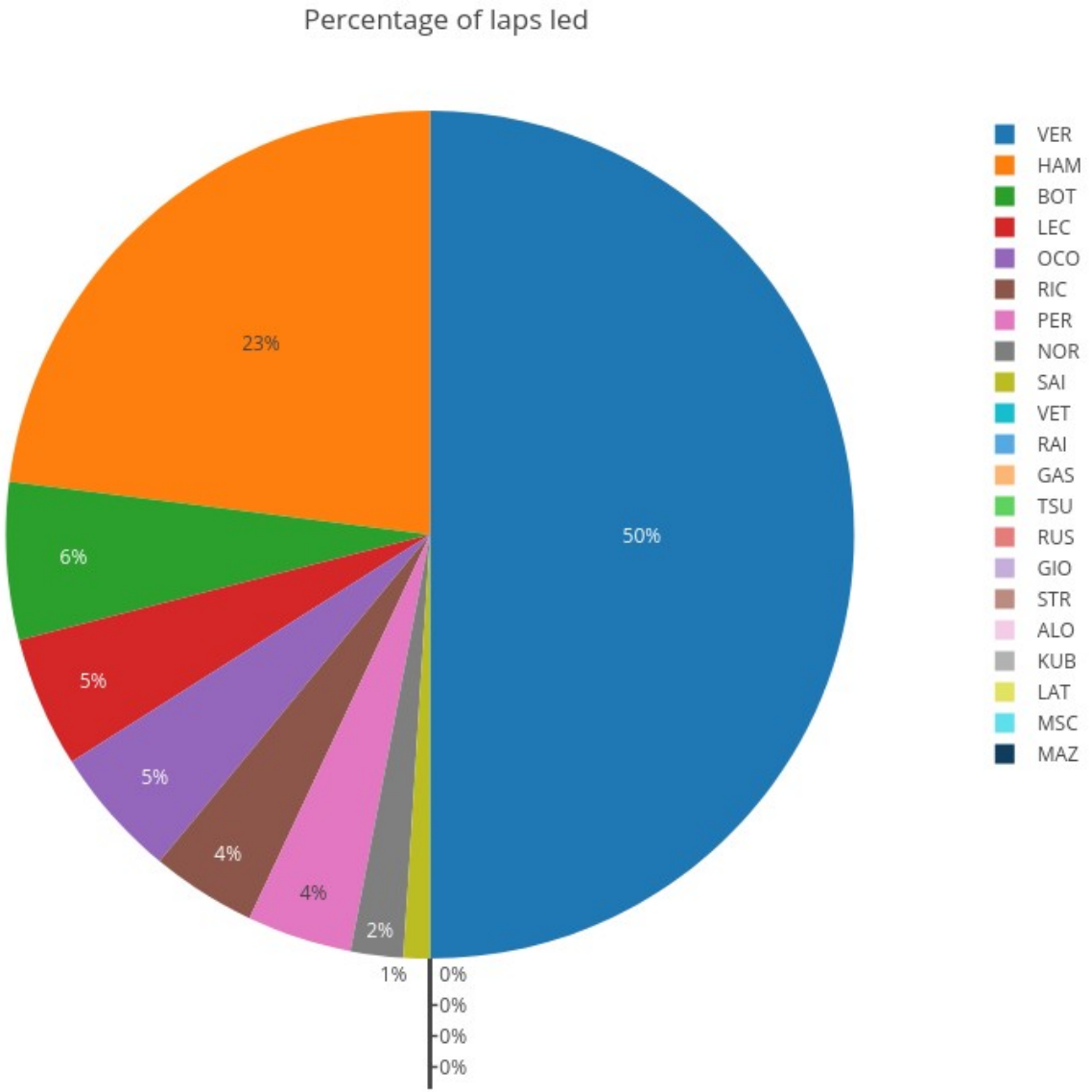
- Ordenar el DataFrame
- Conseguir ambos ejes como lista
- Crear objetos Bar y Layout
- Imprimir por pantalla

```
val driverCodes = results
    .sort("avgPosition")
    .select("code")
    .as[String]
    .collect()
    .toList

val avgPosition = results
    .sort("avgPosition")
    .select("avgPosition")
    .as[Double]
    .collect()
    .toList

val data = Seq(
  Bar(
      driverCodes,
      positionDelta
  )
)

val layout = Layout(
  barmode = BarMode.Group
)

plot(data, layout)
```

Descripción técnica: Visualización



Descripción técnica: Visualización



Descripción técnica: Migración

Objetivo principal: ver diferencias a nivel de programación

```scala
val lastRace = Window.partitionBy("year")

val lastRaces = spark.read.format("csv")
    .option("header", "true")
    .option("sep", ",")
    .load("../data/races.csv")
    .where(col("year") >= 1990 && col("year") <= 1999)
    .withColumn("round", col("round").cast(IntegerType))
    .withColumn("max", max(col("round")).over(lastRace))
    .where(col("round") === col("max"))
    .select("raceId", "year")
```

Scala

Python

```python
lastRace = Window.partitionBy("year")

lastRaces = spark.read.format("csv")\
    .option("header", "true")\
    .option("sep", ",")\
    .load("../data/races.csv")\
    .where((F.col("year") >= 1990) & (F.col("year") <= 1999))\
    .withColumn("round", F.col("round").cast(T.IntegerType()))\
    .withColumn("max", F.max(F.col("round")).over(lastRace))\
    .where(F.col("round") == F.col("max"))\
    .select("raceId", "year")
```

Descripción técnica: Migración

Automatización del proceso: primer paso

Implementación de expresiones regulares para sustitución

Ejemplos:

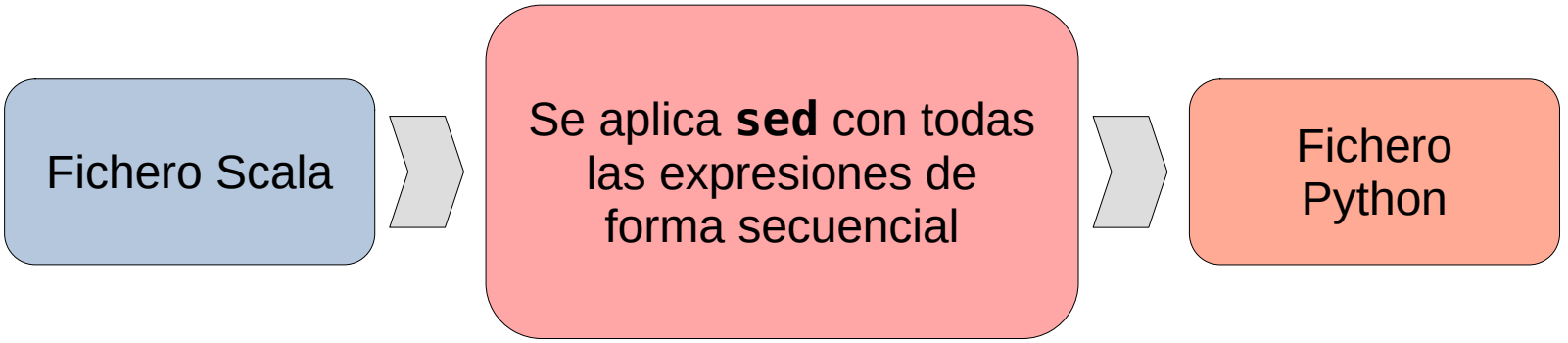
- Eliminación de `var` y `val`:
  `(val|var)[ ]+?(?=[a-zA-Z])` → <cadena vacía>
- Añadir carácter `\` a los finales de línea:
  `(\))(\n)` → `$1\\$2`
- Intercambiar parámetros de `na.replace()`:

```
na\.replace\([ |\n]*(.*)[ |\n]*,[ |\n]*(.*)[ |\n]*\)
na.replace($2, $1)
```

Descripción técnica: Migración

Automatización del proceso: segundo paso

Implementación de un script de Bash que aplique las expresiones regulares automáticamente.



Otra opción:

Exportar el plan de Spark de una API y usarla en otra

Requiere investigación

Pendiente como trabajo futuro

Comparativa

Recolección de métricas

Necesitamos obtener métricas sobre:

- Gestión del proceso
- Uso de memoria
- Tiempos de ejecución

La Spark UI nos dará lo que necesitamos

Usaremos:

- Trabajos completados
- Tareas completadas
- Tareas saltadas
- Etapas completadas
- Tiempo de tarea
- Total leído
- Total leído en operaciones de shuffle
- Total escrito en operaciones de shuffle