Apache Spark
Introducción

Taller de Big Data

Agenda

- Apache Spark
- Estructura de datos: Spark RDD
- Operaciones: Transformaciones y Acciones
- Estructura de datos: Spark DataFrames

Apache Spark

¿Qué es Apache Spark?

- Un motor general para procesamiento a gran escala de datos.
- Ejecuta en Apache YARN, *standalone*, en Amazon EC2 o sobre Apache Mesos.
  - Aquí sólo veremos el funcionamiento *standalone*
- Soporta varios lenguajes de programación
  - Python, Scala, Java y recientemente R.

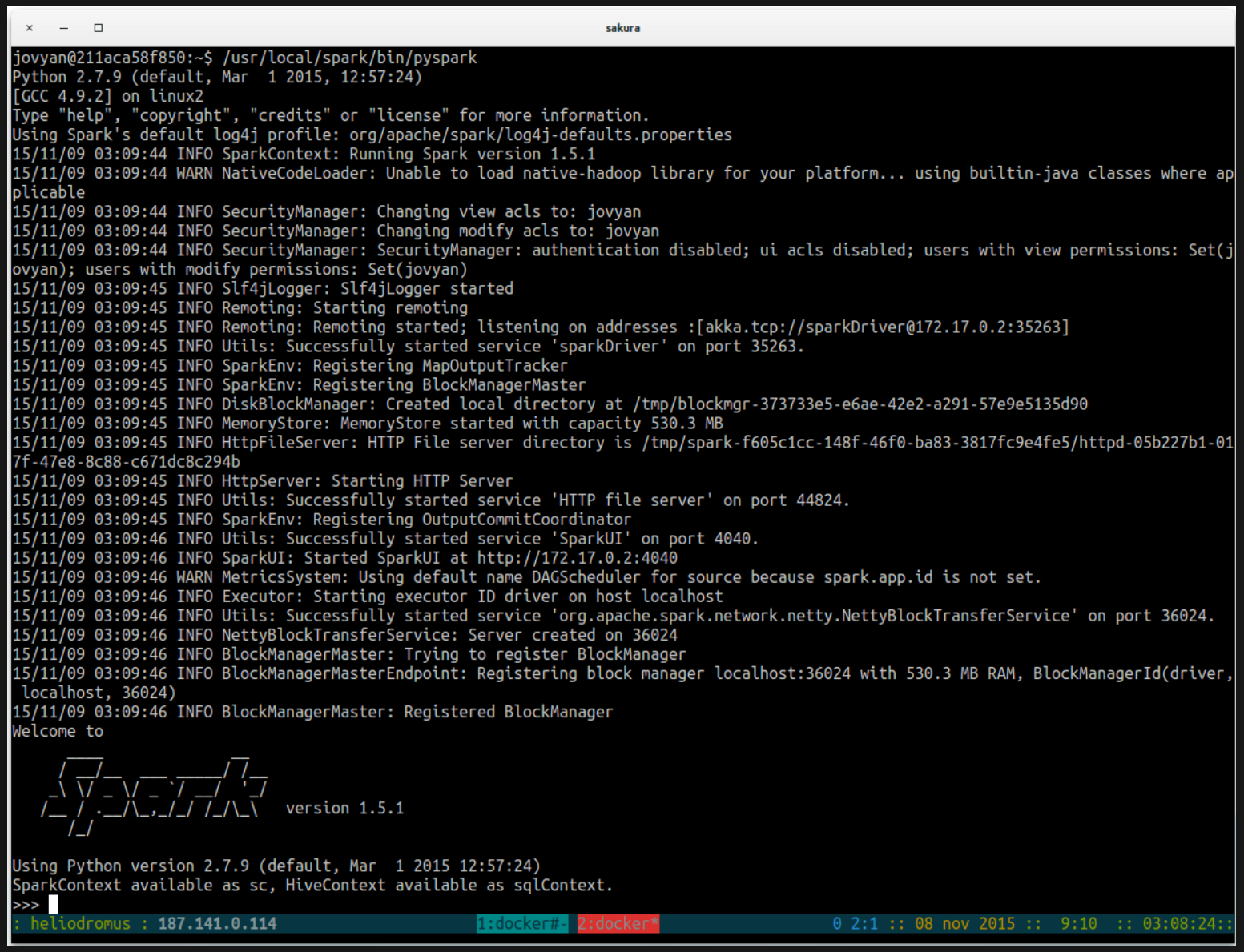
¿Por qué Apache Spark?

- Soporta Algoritmos Iterativos
  - Como algoritmos de grafos
- Exploración Iterativa
- Puede ejecutar en *batch.*
- Soporta *streaming.*

Historia

- Todo empezó con el proyecto **Mesos**, un *framework* distribuido de ejecución, realizado para una clase en UC Berkeley en 2009.
- **Spark** fué creado para probar **Mesos.**
- Fué tal su éxito que se abrió el código en 2010.

Funcionamiento



Modelo de Ejecución

Ciclo de vida

- Crear un RDD a partir de datos externos
- Transformarlo *lazily* en nuevos RDDs usando **transformaciones.**
- Usa `cache()` en los RDDs que vayas a reutilizar
- Utiliza **acciones** para iniciar la ejecución en paralelo

Spark RDD

Spark RDD

- *Resilient Distributed Dataset*
- Abstracción que representa una colección de objetos de sólo lectura particionada a lo largo de varias máquinas.

Spark RDD: Ventajas

- Pueden ser reconstruidas gracias a su *linage.*
- Pueden ser manipuladas en paralelo.
- Están *cached* en memoria para su uso inmediato.
- Son almacenadas de manera distribuida.
- Contienen cualquier tipo de dato, incluidos los definidos por el programador.

Spark RDD: Operaciones

- Soportan dos tipos de operaciones:
  - **Transformaciones**
  - **Acciones**
- Las *transformaciones* construyen un nuevo RDD a partir del anterior.
  - El cual queda guardado en el *linage graph* (DAG)
- Las *acciones* calculan el resultado basado en el RDD.
- La diferencia es que las transformaciones son computadas de manera *lazy* y sólo son ejecutadas hasta la acción.

Operaciones

Transformaciones

Demo

Acciones

Demo

Spark DataFrames

Spark SQL

- Agregado recientemente (2015) a Apache Spark
- Ejecuta *queries* `SQL` o `HiveQL`.
- Agrega una capa de abstracción encima de los RDDs al convertirlos en **Data Frames**, análogos a los usados en R y Pandas.
- Define un `sqlContext` análogo a `sc`

Spark SQL

- Unifica la interfaz para escribir/leer en varios formatos
- `df = sqlContext.read \`
  `.format(“json”)\`
  `.load(“….”)`
- `df = sqlContext.write \`
  `.format(“parquet”)\`
  `.mode(“append”)`
  `.saveAsTable(“….”)`

Spark SQL

- Seleccionar columnas y filtrar
- Realizar `joins`
- Agregaciones
- Funciones analíticas
- Graficación
  - usando `pandas` ó `ggplot`

Spark DataFrames

- Soporta una gran cantidad de formatos de datos y sistemas de almacenamiento.
  - Json, parquet, avro, csv, etc.
- Optimización con Spark SQL Catalyst
  - De hecho son más eficientes que los RDD
- APIs en Python, Java, Scala, R.
- Integrados con toda la infraestructura de Spark.
  - ML, *Streaming*, GraphX, etc.

Spark DataFrames

- Al estar construidas sobre Spark RDD, puedes usar operaciones de RDD sobre un DataFrame, pero no es recomendable, ya que obtendrías un RDD de regreso.
- Los DataFrames (debido al optimizador Catalyst) son mucho más rápidos que los RDDs, además de que esa velocidad ganada es igual en todos los lenguajes.
  - Esto no sucede con los RDDs, un RDD en Scala es más rápido que un RDD en Python.

Spark DataFrames

- Por *default* Spark utiliza el `SQLContext` nativo, pero, si estás conectado a un *cluster* con Apache Hive, puedes utilizar el `HiveContext` en su lugar.
- Esto te permitirá usar todo el `HiveQL`, sus `udf's` y escribir/leer a tablas en Hive.
- En este taller **NO** tenemos un `HiveContext` `- :(`

Spark DataFrames

- Los DataFrames tienen esquema.
  - Se puede ver con `printSchema()`.
  - Spark se encarga de extraerlo (`Parquet`, tabla) o inferirlo (`json`).
- Un DataFrame `Column` es una abstracción, ya que no todas las fuentes están en formato columnar.
- A pesar de la sintaxis, es importante recordar que una `Column` **NO** es igual a un DataFrame

Spark DataFrame

Demo

Spark ML

Spark ML

- Apareció en Spark 1.2
  - `spark.ml`
- Su objetivo es estandarizar una API para algoritmos de aprendizaje de máquina, de tal manera que sea más fácil combinarlos en un *pipeline* o *workflow*.
- Usa Spark DataFrames
  - **IMPORTANTE**: `spark.mlib` usa RDDs, `spark.ml` usa DataFrames.
  - No veremos `spark.mlib`

Spark ML: Transformers

- Es un algoritmo que transforma un DataFrame en otro DataFrame, generalmente agregando una o más columnas.
- Su método más importante es `transform()`
- Incluye *features transformers* y *learning models.*

Spark ML: Transformers

Feature Transformer

- Toma un dataset
- Lee una columna
  - `texto`
- La convierte en una nueva columna de *features* y la agrega al dataset
  - `vectorización del texto`
- Devuelve un dataset nuevo

Learning model

- Toma un dataset
- Lee la columna que contiene los *features*
- Predice las etiquetas para cada vector de *features*
- Agrega esa columna de etiquetas
- Devuelve un dataset nuevo

Spark ML: Estimators

- Es un algoritmo que puede ser ajustado en un DataFrame para producir un **Transformer.**
- Un algoritmo de aprendizaje de máquina es un **Estimator** que entrena en el dataset para obtener un modelo.
- Implementan el método `fit()`.
- La regresión logística es un Estimator e invocando `fit()` entrena un modelo de regresión logística que es un Transformer.

Spark ML: Pipeline

- En aprendizaje de máquina, es común ejecutar los algoritmos en una secuencia para procesar y aprender de los datos.
- Spark ML abstrae este *workflow* en un Pipeline, el cual consiste en una secuencia de Transformers y Estimators.

Spark ML

Demo

Spark Streaming

Spark Streaming

Demo

Spark GraphX

Spark GraphX

Demo