Aprendiendo Spark

Nociones basicas y utilización en dana

Quien?

El porqué de Spark

- Hadoop y su modelo de programación, MapReduce, era ineficiente para procesos de algoritmos iterativos o consultas interactivas .
- Spark da soporte en el mismo entorno de ejecución a aplicaciones que, hasta entonces, requerían diversos y separados sistemas distribuidos.
- Spark está diseñado para ser accesible a través de sencillas APIs para lenguajes de programación como Python , Java , Scala , R .
- Spark es compatible con Hadoop y toda su estructura de datos.
- Procesamiento en memoria. Con el fin de mejorar el rendimiento entre operaciones, se permite la persistencia o el almacenamiento en caché de un RDD entre operaciones.
- Las aplicaciones son ejecutadas en Cluster por los nodos y están gestionadas por un maestro
- Escalable y tolerante a fallos

Spark vs. Hadoop

- Diseñado para vencer limitaciones de Hadoop (que escribe datos entre etapas de proceso en disco duro).
- Menor tiempo de procesamiento: Gracias a su procesamiento “in-memory”. 100x más rápido que Hadoop MapReduce (DAG, flujo de datos acíclico).
- Escalabilidad: Posibilidad de ir incrementando nuestro cluster a medida que vamos necesitando más almacenamiento o procesamiento.
- Sencillo: Una completa API, es posible programar complejos hilos de ejecución paralelos en unas pocas líneas de código.

Como funciona Spark

Archivos RDD (Resilient Distribited Datasets)

- Existen dos tipos de operaciones con los RDDs pueden ser transformados, crea un nuevo RDD (Map. Filter, union..) o consultados (acción), nos da un resultado (Reduce, count..)
- Las transformaciones las realiza de forma perezosa (lazy) esto quiere decir que las diferentes transformaciones solicitadas se irán guardando, hasta que se ejecute una acción.

Ecosistema SPARK: Dispone de herramientas de alto nivel

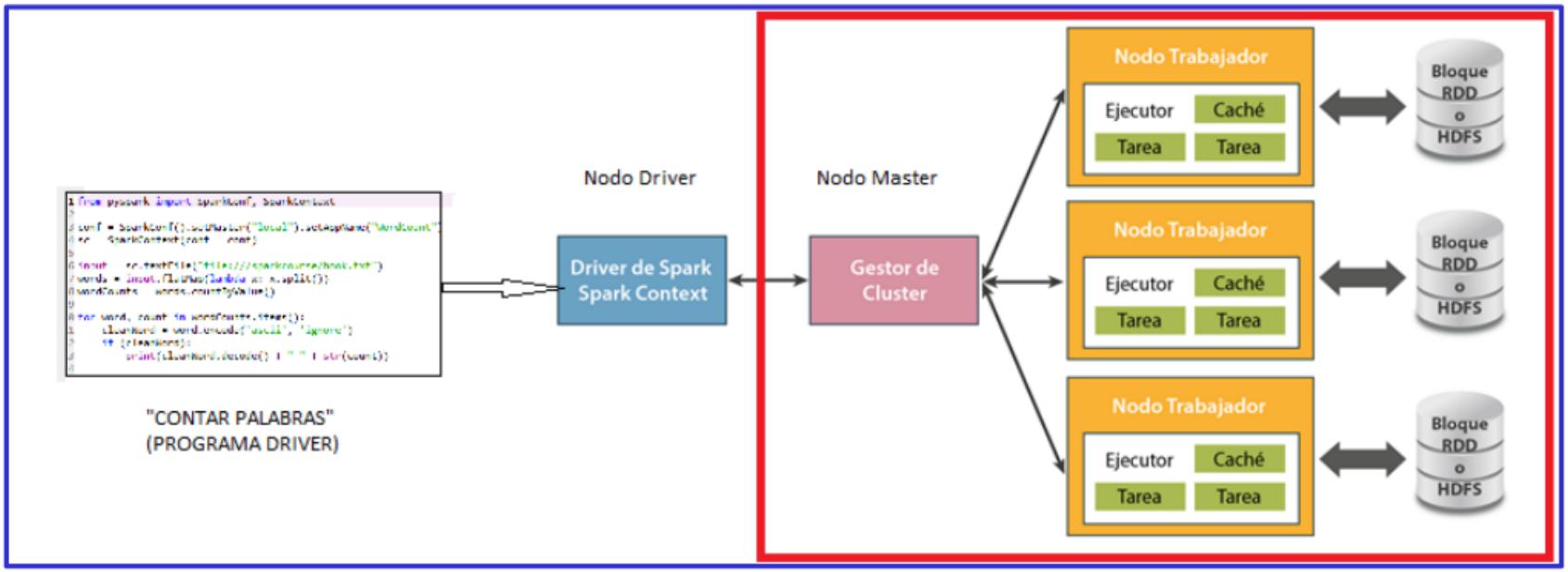
- Spark- Subimt: Un script para ejecutar aplicaciones
- Spark Streaming: Mezclar Batch processing con real time
- Spark -SQL: Un modulo para trabajar con bases de datos estructurados.
- Spark – Mlib: Machine Learning. Aprendizaje automático.

Programa Spark

Se organiza bajo la siguiente estructura

1. A partir de una variable de entorno llamada sc context
2. Se crea un objeto RDD ("base de datos") leyendo de fichero (txt, csv, json, etc), base de datos, colecciones de Python, etc
3. Se realizan transformaciones sobre el RDD inicial. Estas transformaciones generan otros objetos RDD (no se elimina el original). Unificar datos, filtrar, formatear, etc.
4. Se realizan acciones sobre el RDD. Para procesar y extraer información relevante
5. A través de estas transformaciones y acciones se genera un RDD final (resultado)
6. Este RDD puede ser mostrado, almacenado, exportado,..

Spark en modo cluster



El clúster Spark lo componen

- El nodo máster (maestro), que gestionará los recursos del clúster.
- Un conjunto de nodos worker en los que los procesos executors, realizarán las tareas sobre los datos. Los nodos executors pueden ser distintas máquinas de un clúster o diferentes cores de la CPU

Primeros pasos prácticos. Instalación

Descargamos Apache Spark

wget `http://archive.apache.org/dist/spark/spark-1.6.1/spark-1.5.0-bin-hadoop2.6.tgz` tar -xf spark-1.5.1-bin-hadoop2.6.tg

Desempaquetamos

tar -xvf spark_2.6.3.tar.gz

Creamos una carpeta con el nombre de la aplicación

mkdir /opt/spark

Movemos la carpeta de Spark a la nueva ruta

mv Spark/* /opt/spark

Podemos lanzar la aplicación desde la consola

/bin/bash /opt/spark/Spark

Primeros pasos prácticos. Arrancar pyspark

En local

./bin/pyspark

```
ManagerId(driver, localhost, 39996)
17/02/08 02:22:12 INFO BlockManagerMaster: Registered BlockManager
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.1
      /_/

Using Python version 2.7.9 (default, Jun 29 2016 13:08:31)
SparkContext available as sc, HiveContext available as sqlContext.
>>>
```

En dana

./bin/pyspark –master spark://dana:7077 –num-executors 3

```
17/02/08 02:20:41 INFO SparkDeploySchedulerBackend: Granted executor ID app-20170208022041-0021/2 on hostPort 192.168
.134.11:58278 with 4 cores, 1024.0 MB RAM
17/02/08 02:20:41 INFO AppClient$ClientEndpoint: Executor updated: app-20170208022041-0021/1 is now RUNNING
17/02/08 02:20:42 INFO AppClient$ClientEndpoint: Executor updated: app-20170208022041-0021/2 is now RUNNING
17/02/08 02:20:42 INFO AppClient$ClientEndpoint: Executor updated: app-20170208022041-0021/0 is now RUNNING
17/02/08 02:20:42 INFO SparkDeploySchedulerBackend: SchedulerBackend is ready for scheduling beginning after reached
minRegisteredResourcesRatio: 0.0
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.1
      /_/

Using Python version 2.7.9 (default, Jun 29 2016 13:08:31)
SparkContext available as sc, HiveContext available as sqlContext.
```

Primeros pasos prácticos. Ejecutar scripts

Python

./bin/spark-submit < *script.py* >

R

./bin/spark-submit < *script.R* >

Scala

./bin/spark-shell < *script.scala* >

Opciones

```
./bin/spark-submit \
  --class <main-class> \
  --master <master-url> \
  --deploy-mode <deploy-mode> \
  --conf <key>=<value> \
  ... # other options
  <application-jar> \
  [application-arguments]
```

Spark en dana (I)

Ubicación de spark

< *user* >@dana:/srv/nfs4/opt/spark/spark-1.6.1-bin-hadoop2.6$

Ejemplos disponibles

ls examples/src/main/python/

```
guille@dana:/srv/nfs4/opt/spark/spark-1.6.1-bin-hadoop2.6/examples/src/main/python$ ls
als.py                    hbase_inputformat.py    ml                    pi.py              streaming
avro_inputformat.py       hbase_outputformat.py   mllib                 sort.py            transitive_closure.py
cassandra_inputformat.py  kmeans.py               pagerank.py           sql.py             wordcount.py
cassandra_outputformat.py logistic_regression.py  parquet_inputformat.py status_api_demo.py
guille@dana:/srv/nfs4/opt/spark/spark-1.6.1-bin-hadoop2.6/examples/src/main/python$ ls mllib
binary_classification_metrics_example.py     isotonic_regression_example.py          random_rdd_generation.py
correlations.py                              kmeans.py                               ranking_metrics_example.py
decision_tree_classification_example.py      logistic_regression.py                  recommendation_example.py
decision_tree_regression_example.py          multi_class_metrics_example.py          regression_metrics_example.py
fpgrowth_example.py                          multi_label_metrics_example.py          sampled_rdds.py
gaussian_mixture_model.py                    naive_bayes_example.py                  word2vec.py
gradient_boosting_classification_example.py  random_forest_classification_example.py
gradient_boosting_regression_example.py      random_forest_regression_example.py
guille@dana:/srv/nfs4/opt/spark/spark-1.6.1-bin-hadoop2.6/examples/src/main/python$
```

Spark en dana (II)

Arquitectura programa pyspark

```
Random Forest Classification Example.
"""
from __future__ import print_function

import sys

from pyspark import SparkContext
# $example on$
from pyspark.mllib.tree import RandomForest, RandomForestModel
from pyspark.mllib.util import MLUtils
# $example off$

if __name__ == "__main__":
    sc = SparkContext(appName="PythonRandomForestClassificationExample")
    # $example on$
    # Load and parse the data file into an RDD of LabeledPoint.
    data = MLUtils.loadLibSVMFile(sc, 'data/mllib/sample_libsvm_data.txt')
    # Split the data into training and test sets (30% held out for testing)
    (trainingData, testData) = data.randomSplit([0.7, 0.3])

    # Train a RandomForest model.
    #  Empty categoricalFeaturesInfo indicates all features are continuous.
    #  Note: Use larger numTrees in practice.
    #  Setting featureSubsetStrategy="auto" lets the algorithm choose.
    model = RandomForest.trainClassifier(trainingData, numClasses=2, categoricalFeaturesInfo={},
                                         numTrees=3, featureSubsetStrategy="auto",
                                         impurity='gini', maxDepth=4, maxBins=32)

    # Evaluate model on test instances and compute test error
    predictions = model.predict(testData.map(lambda x: x.features))
    labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions)
    testErr = labelsAndPredictions.filter(lambda (v, p): v != p).count() / float(testData.count())
    print('Test Error = ' + str(testErr))
    print('Learned classification forest model:')
    print(model.toDebugString())

    # Save and load model
    model.save(sc, "target/tmp/myRandomForestClassificationModel")
    sameModel = RandomForestModel.load(sc, "target/tmp/myRandomForestClassificationModel")
    # $example off$
```

Spark en dana (III)

Lanzar un programa python en dana

./bin/spark-submit –master spark://dana:7077 –num-executors 2 examples/src/main/python/mllib/random_forest_classification_example.py

```
src/main/python/mllib/random_forest_classification_example.py:50, took 0,066767 s
Test Error = 0.0
Learned classification forest model:
TreeEnsembleModel classifier with 3 trees

  Tree 0:
    If (feature 462 <= 0.0)
     Predict: 0.0
    Else (feature 462 > 0.0)
     Predict: 1.0
  Tree 1:
    If (feature 272 <= 9.0)
     If (feature 235 <= 92.0)
      Predict: 1.0
     Else (feature 235 > 92.0)
      If (feature 629 <= 0.0)
       Predict: 1.0
```

Spark en dana (IV)

Lanzar otro programa python en dana

./bin/spark-submit –master spark://dana:7077 –num-executors 2 examples/src/main/python/mllib/ multi_class_metrics_example.py.py

```
17/02/08 00:44:51 INFO DAGScheduler: ResultStage 26 (collectAsMap at MulticlassMetrics.scala:54) finished in 0,027 s
17/02/08 00:44:51 INFO DAGScheduler: Job 22 finished: collectAsMap at MulticlassMetrics.scala:54, took 0,133994 s
Class 0.0 precision = 0.9
Class 0.0 recall = 0.75
Class 0.0 F1 Measure = 0.818181818182
Class 1.0 precision = 1.0
Class 1.0 recall = 0.944444444444
Class 1.0 F1 Measure = 0.971428571429
Class 2.0 precision = 0.65
Class 2.0 recall = 0.866666666667
Class 2.0 F1 Measure = 0.742857142857
Weighted recall = 0.842105263158
Weighted precision = 0.865789473684
Weighted F(1) Score = 0.846753246753
Weighted F(0.5) Score = 0.856545374358
Weighted false positive rate = 0.0693779904306
```

Spark en dana (V)

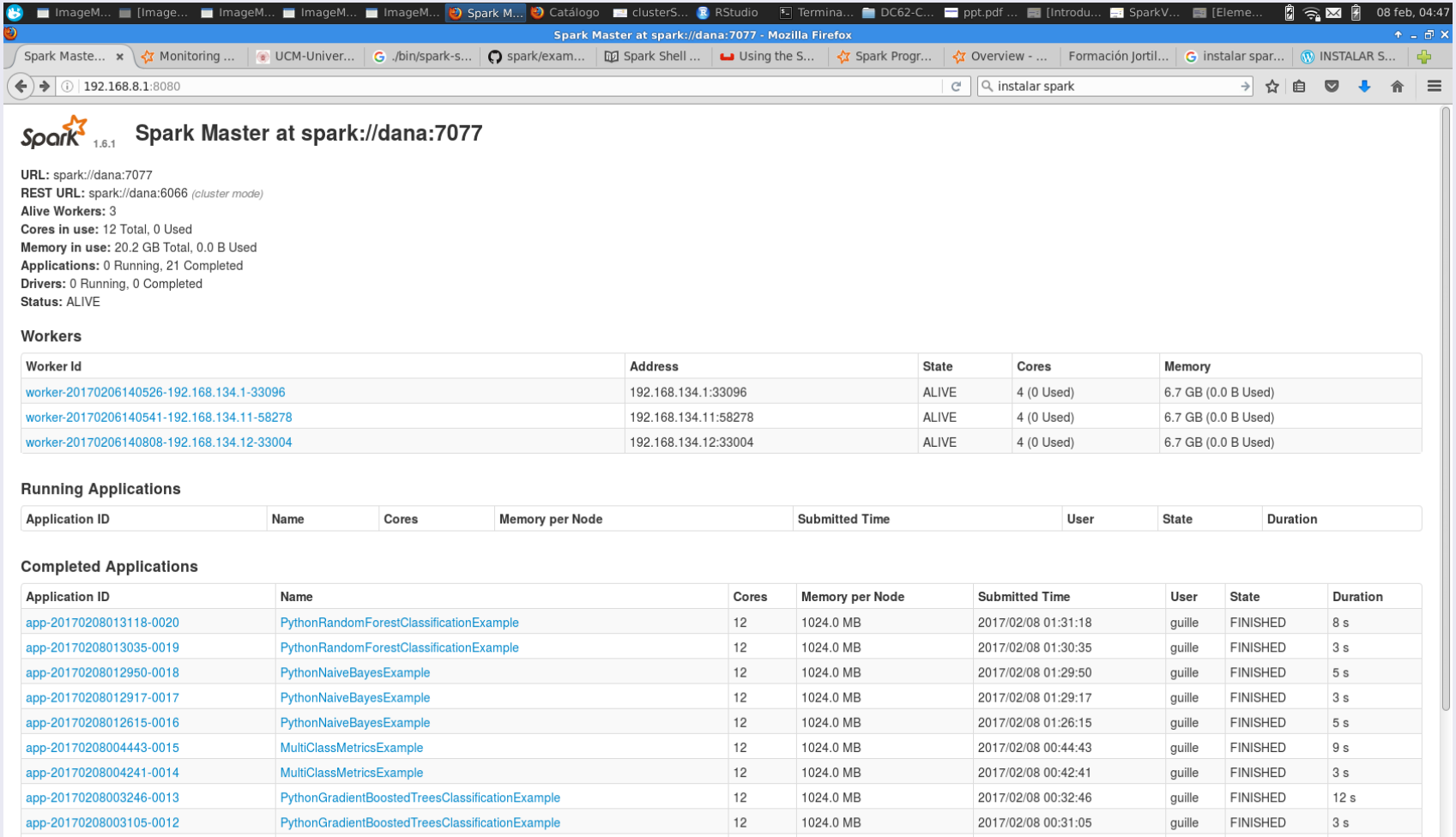
Configuración de spark

```
guille@dana:/srv/nfs4/opt/spark/spark-1.6.1-bin-hadoop2.6$  ls sbin/
slaves.sh                      start-slaves.sh
spark-config.sh                start-thriftserver.sh
spark-daemon.sh                stop-all.sh
spark-daemons.sh               stop-history-server.sh
start-all.sh                   stop-master.sh
start-history-server.sh        stop-mesos-dispatcher.sh
start-master.sh                stop-mesos-shuffle-service.sh
start-mesos-dispatcher.sh      stop-shuffle-service.sh
start-mesos-shuffle-service.sh stop-slave.sh
start-shuffle-service.sh       stop-slaves.sh
start-slave.sh                 stop-thriftserver.sh
guille@dana:/srv/nfs4/opt/spark/spark-1.6.1-bin-hadoop2.6$ ls conf/
docker.properties.template   slaves                 spark-defaults.conf.template
fairscheduler.xml.template   #slaves#               #spark-env.sh#
log4j.properties             slaves~                spark-env.sh
log4j.properties.template    slaves.template        spark-env.sh~
metrics.properties.template  spark-defaults.conf    spark-env.sh.template
```

Monitorización (I)

Control de ejecución en Spark en puerto

- OpenVPN + fichero de configuración
- Visionado en streaming: `http://192.168.8.1:8080/`



Monitorización (II)

Localización resultados

```
guille@dana:/srv/nfs4/opt/spark/spark-1.6.1-bin-hadoop2.6$ ls work/
app-20170206132246-0000  app-20170207201104-0007  app-20170208004443-0015
app-20170207193929-0000  app-20170207201213-0008  app-20170208012615-0016
app-20170207194904-0001  app-20170207201457-0009  app-20170208012917-0017
app-20170207195504-0002  app-20170207202814-0010  app-20170208012950-0018
app-20170207195933-0003  app-20170208002517-0011  app-20170208013035-0019
app-20170207200433-0004  app-20170208003105-0012  app-20170208013118-0020
app-20170207200742-0005  app-20170208003246-0013  app-20170208022041-0021
app-20170207200818-0006  app-20170208004241-0014
guille@dana:/srv/nfs4/opt/spark/spark-1.6.1-bin-hadoop2.6$ ls target/
tmp
guille@dana:/srv/nfs4/opt/spark/spark-1.6.1-bin-hadoop2.6$ ls target/tmp/
myGradientBoostingClassificationModel  myRandomForestClassificationModel
guille@dana:/srv/nfs4/opt/spark/spark-1.6.1-bin-hadoop2.6$ ls target/tmp/myRando
mForestClassificationModel/
metadata
```

Referencias

Sitio ofial de apache

`http://spark.apache.org/docs/latest/index.html`

Tutorial en castellano bastante completo

`http://www.jortilles.com/wp-content/uploads/2015/12/Introducci%C3%B3n_Spark.pdf`

Mi contacto

Para cualquier duda `gvillari@ucm.es`