

Cluster con Apache Spark

Vanessa K. Sotomayor A.^a Jairo S. Valle R.^b John J. Villavicencio S.^c

^a San Cayetano alto 1101608, Loja, Loja, Ecuador

^b San Cayetano alto 1101608, Loja, Loja, Ecuador

^c San Cayetano alto 1101608, Loja, Loja, Ecuador

Resumen

El presente documento explica de manera detallada como es el proceso de instalación e implementación del cluster Apache Sapark en Ubuntu, también se desarrolla una aplicación tanto en paralelo como en secuencial. Una vez que se tiene desarrollada la aplicación se realiza un plan de pruebas para obtener tiempo de ejecución, aceleración, eficiencia, y coste al ejecutar la aplicación en paralelo y secuencial, con esta información se realizará un análisis de rendimiento.

Key words: Cluster; BigData; Paralelo; Serial; Middelware; Spark; Grid Computing.

1. Introducción

En los últimos años la incorporación de avances de arquitectura en los microprocesadores ha sido significativa. Sin embargo, no podemos depender continuamente de procesadores más rápidos para obtener más eficiencia. Hay límites físicos, como la velocidad de la luz, que eventualmente van a desacelerar la reducción, que se ha visto año tras año, en el tiempo que dura un ciclo (tiempo para ejecutar la operación más básica) de CPU. Dadas las dificultades en mejorar la eficiencia de un procesador, la convergencia en eficiencia entre microprocesadores y los supercomputadores tradicionales, y el relativo bajo costo de los microprocesadores, ha permitido el desarrollo de computadores paralelos viables comercialmente con decenas, cientos y hasta miles de microprocesadores. Un computador paralelo es un conjunto de procesadores capaces de cooperar en la solución de un problema. Esta definición incluye supercomputadores con cientos de procesadores, máquinas con múltiples procesadores, redes de estaciones de trabajo y redes de PCs (Clusters de PCs).

★

Email addresses: vksotomayor@utpl.edu.ec (Vanessa K. Sotomayor A.), jsvalle1@utpl.edu.ec (Jairo S. Valle R.), jjvillavicencio@utpl.edu.ec (John J. Villavicencio S.).

2. Estado del Arte:

2.1. Grid Computing:

Grid se define como un sistema de computación distribuido que permite compartir y gestionar información mediante la colaboración de diferentes nodos, estos nodos de computadoras con diferente hardware y software, las cuales se utilizan para procesar una tarea que demanda una gran cantidad de recursos y poder de procesamiento.[3]

Para el uso de un grid es necesario un software denominado middleware, que asegura la comunicación transparente entre diferentes dispositivos o nodos a ser utilizados

Entre sus ventajas principales tenemos:

- **Gran poder de procesamiento:** Al combinar el poder de varias computadoras se puede procesar más datos.
- **Aprovechamiento de recursos existentes:** Se puede usar computadoras en los momentos en que no están siendo utilizadas para que ayuden con el procesamiento de datos en la grid.
- **No hay límite por espacio físico:** Los equipos que forman la grid pueden estar a gran distancia entre sí. Incluso pueden estar en diferentes continentes.
- **Brinda flexibilidad ante posibles fallos:** Si una máquina que forma parte del grid falla, el sistema lo

reconoce y envía los datos no procesados a otra máquina.

- **Ahorro en costos:** Cuando se necesita una gran capacidad de procesamiento, incurrir en gastos de equipos con ese poder no es una buena idea para medianas y pequeñas compañías por lo que una Grid es una opción viable y considerablemente más barata.

2.2. Middleware (Apache Spark):

El middleware seleccionado para la implementación del cluster es Apache Spark, es una herramienta con un gran potencial, actualmente es muy utilizada en Big Data y Machine Learning. Apache Spark combina un sistema de computación distribuida a través de clusters con ordenadores de una manera sencilla y elegante para escribir programas. Un cluster es un sistema basado en la unión de varios servidores que trabajarán de forma paralela como si se tratara de uno solo, es decir, tenemos un servidor formado por otros servidores. Podemos tener un cluster de 2, 10, 20 ó 100 servidores. Cuantos más servidores tengamos, más potencia de hardware estará disponible. Gracias a que Apache Spark hace un uso eficiente de las máquinas que formen un cluster, el sistema será escalable y realizará un correcto balanceo de carga.

2.3. Arquitectura de un cluster de Apache Spark:

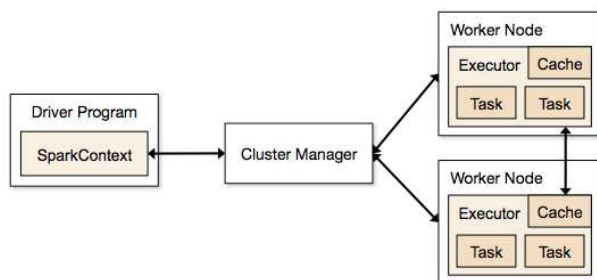


Figura 1. Arquitectura Apache Spark

En la arquitectura de un cluster montado con Apache Spark se puede observar que existen distintos agentes Fig.1. El primero de ellos es el controlador. Tras esto, encontramos el cluster manager y finalmente los workers. De una manera muy resumida, el manager hace las veces de maestro y el worker de esclavo. Conviene conocer algunos términos:[4]

- **Aplicación:** programa desarrollado sobre Spark por un programador.
- **Controlador del programa (driver program):** es el proceso que ejecuta la función main de la aplicación.
- **Cluster manager:** servicio externo utilizado para gestionar los recursos de los workers.
- **Worker:** nodo (servidor) que puede ejecutar código Spark.
- **Task:** operaciones (tareas) a realizar por un worker.

2.4. Modelo de programación:

Un programa típico se organiza de la siguiente manera[1]:

- A partir de una variable de entorno llamada context se crea un objeto RDD leyendo datos de fichero, bases de datos o cualquier otra fuente de información.
- Una vez creado el RDD inicial se realizan transformaciones para crear más objetos RDD a partir del primero. Dichas transformaciones se expresan en términos de programación funcional y no eliminan el RDD original, sino que crean uno nuevo.
- Tras realizar las acciones y transformaciones necesarias sobre los datos, los objetos RDD deben converger para crear el RDD final. Este RDD puede ser almacenado.

Un pequeño ejemplo de código en Python que cuenta el número de palabras que contiene un archivo sería el siguiente:

```

1 myRDD = spark.textFile("hdfs://...")
2 words = myRDD.flatMap(lambda line : line.
    split(" "))
3                 .map(lambda word : (word, 1))
4                 .reduceByKey(lambda a, b : a+b)
5 words.saveAsTextFile("hdfs://...")

```

Spark ofrece más de 80 operadores de alto nivel que hacen que sea fácil de construir aplicaciones paralelas. Y se puede utilizar de forma interactiva desde las conchas Scala, Python y R.

3. Implementación del Cluster

3.1. Estudio a nivel de Hardware, Software y Red

3.1.1. Descripción de Hardware

Modelo	Procesador	Memoria	Ram	Tarjeta de red	Propietario	Rol
Toshiba C45-ASP4311FL	Core i5 3230M (2.60 GHz)	750 GB	6 GB	Ethernet 10/100 LAN	Vanessa Sotomayor	Manger
Dell Inspiron 15r - N5110	2.ª generación de Intel® Core i7-2630QM (2,0 GHz, 6 MB, 4 N)	750 GB	DDR3 SDRAM de 8.192 MB (8GB)	Ethernet 10/100 LAN	John Villavicencio	Worker
Dell Inspiron M501R	Core i5	750 GB	DDR3 SDRAM de 4 GB	Ethernet 10/100 LAN	Jairo Valle	Worker

Figura 2. Descripción de hardware.

3.1.2. Descripción de Software

- **Sistema Operativo:** Se ha elegido usar Ubuntu 16.04 o distribuciones basadas en Ubuntu para la instalación y configuración del middleware Apache Spark.

- Herramientas de monitoreo:

Apache Spark cuenta con un panel de administración en el cual se puede monitorizar como se encuentra funcionando la conexión del cluster y a la vez obtener datos sobre la ejecución de programas en el cluster Fig3.

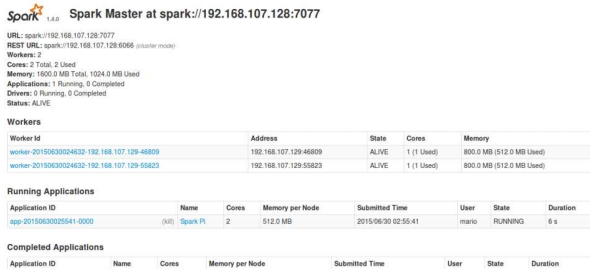


Figura 3. Monitorio Apache Sapark.

3.1.3. Descripción de Red

Para nuestro proyecto trabajaremos con la topología de red en estrella, la misma que reduce la posibilidad de fallo de red conectando todos los nodos a un nodo central. Utilizaremos el protocolo TCP/IP que es un conjunto de protocolos estándar para conectar equipo y crear redes. El servicio de Cluster Server requiere el conjunto de protocolos TCP/IP para las comunicaciones internas y con los clientes. Fig.4 TCP/IP permite establecer comunicación en un entorno heterogéneo, es muy adecuado como protocolo de interconexión de redes corporativas, debido a que proporciona interoperabilidad entre diferentes tipos de sistemas informáticos.

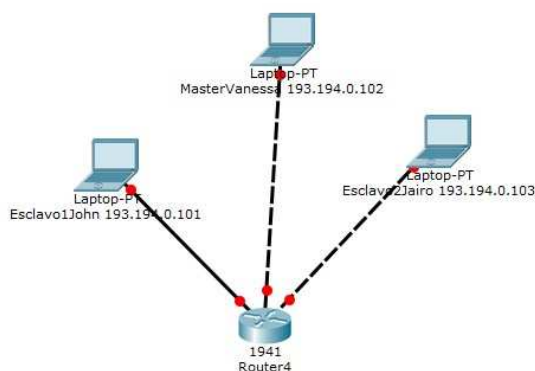


Figura 4. Topología de Red.

3.2. Implementación de la herramienta

3.2.1. Requerimientos de instalación

- a) **Java**, se puede utilizar tanto la versión 6 como la 8.

- b) **Scala**, cualquier versión compatible.
- c) Acceso remoto por SSH entre máquinas.
- d) Apache Spark pre-compilado para Hadoop.
- e) Mínimo dos máquinas virtuales, con el mismo sistema operativo y mismos nombres de usuario

3.2.2. Instalación de la herramienta

3.2.2.1. Instalación y configuración de Java6

```

esclavo1@ubuntu:~$
File Edit View Search Terminal Help
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

esclavo1@ubuntu:~$ sudo add-apt-repository ppa:webupd8team/java
[sudo] password for esclavo1:
Oracle Java (JDK) Installer (automatically downloads and installs Oracle JDK7 /
JDK8 / JDK9). There are no actual Java files in this PPA.

More info (and Ubuntu installation instructions):
- For Oracle Java 7: http://www.webupd8.org/2012/01/install-oracle-java-jdk-7-in-ubuntu-via.html
- For Oracle Java 8: http://www.webupd8.org/2012/09/install-oracle-java-8-in-ubuntu-via-ppa.html

Debian installation instructions:
- Oracle Java 7: http://www.webupd8.org/2012/06/how-to-install-oracle-java-7-in-debian.html
- Oracle Java 8: http://www.webupd8.org/2014/03/how-to-install-oracle-java-8-in-debian.html

Oracle Java 9 (for both Ubuntu and Debian): http://www.webupd8.org/2015/02/install-oracle-java-9-in-ubuntu-linux.html

For JDK9, the PPA uses standard builds from: https://jdk9.java.net/download/ (an

esclavo1@ubuntu:~$
File Edit View Search Terminal Help
esclavo1@ubuntu:~$ sudo apt-get update
Hit:1 http://us.archive.ubuntu.com/ubuntu xenial InRelease
Get:2 http://ppa.launchpad.net/webupd8team/java/ubuntu xenial InRelease [17.6 kB]
Get:3 http://security.ubuntu.com/ubuntu xenial-security InRelease [94.5 kB]
Get:4 http://us.archive.ubuntu.com/ubuntu xenial-updates InRelease [95.7 kB]
Hit:5 http://us.archive.ubuntu.com/ubuntu xenial-backports InRelease
Get:6 http://ppa.launchpad.net/webupd8team/java/ubuntu xenial/main amd64 Package s [2,800 B]
Get:7 http://ppa.launchpad.net/webupd8team/java/ubuntu xenial/main i386 Package s [2,800 B]
Get:8 http://ppa.launchpad.net/webupd8team/java/ubuntu xenial/main Translation-e n [1,260 B]
Fetched 215 kB in 1s (109 kB/s)
Reading package lists... Done
esclavo1@ubuntu:~$ sudo apt-get install oracle-java6-installer
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  gsfontr-xii java-common
Suggested packages:
  binfmt-support visualvm ttf-baeknuk | ttf-unfonts | ttf-unfonts-core
  ttf-kochi-gothic | ttf-sazanami-gothic ttf-kochi-mincho
esclavo1@ubuntu:~$
File Edit View Search Terminal Help
usr/bin/schenagen (schenagen) in auto mode
update-alternatives: using /usr/lib/jvm/java-6-oracle/bin/serialver to provide /
usr/bin/serialver (serialver) in auto mode
update-alternatives: using /usr/lib/jvm/java-6-oracle/bin/wsgen to provide /usr/
bin/wsgen (wsgen) in auto mode
update-alternatives: using /usr/lib/jvm/java-6-oracle/bin/wsimport to provide /u
sr/bin/wsimport (wsimport) in auto mode
update-alternatives: using /usr/lib/jvm/java-6-oracle/bin/xjc to provide /usr/bi
n/xjc (xjc) in auto mode
Oracle JDK 6 installed
update-alternatives: using /usr/lib/jvm/java-6-oracle/jre/lib/amd64/libnpp2.so
to provide /usr/lib/mozilla/plugins/libjavaplugin.so (mozilla-javaplugin.so) in
auto mode
Oracle JRE 6 browser plugin installed
Setting up gsfontr-xii (0.24)
esclavo1@ubuntu:~$ sudo nano /etc/environment
esclavo1@ubuntu:~$ source /etc/environment
esclavo1@ubuntu:~$ java -version
java version "1.6.0_45"
Java(TM) SE Runtime Environment (build 1.6.0_45-b06)
Java HotSpot(TM) 64-Bit Server VM (build 20.45-b01, mixed mode)
esclavo1@ubuntu:~$ echo $JAVA_HOME
/usr/lib/jvm/java-6-oracle/
esclavo1@ubuntu:~$

```


3.2.2.2. Instalación Scala 2.10

The screenshot shows the Scala 2.10.4 download page on www.scala-lang.org/download/2.10.4.html. It lists various download links for different operating systems and architectures. Below the table, there is a section for 'scala' (Programming Language for the JVM) with an 'Install' button. The details section shows the version (2.10.4-400), source (Ubuntu), and size (29.3 MB). A note indicates that the software comes from a 3rd party and may contain non-free components.

Below the download page, a terminal window shows the installation process:

```
esclavo1@ubuntu:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/esclavo1/.ssh/id_rsa):
Created directory '/home/esclavo1/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/esclavo1/.ssh/id_rsa.
Your public key has been saved in /home/esclavo1/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:thQAGLxSPBm1sBux40XFRjA2cnPpSRQWqsj/kfDP8 esclavo1@ubuntu
The key's randomart image is:
+----[RSA 2048]----+
|..+O+..+O*O..|
|..+O+..+O*O..|
|..+O+..+O*O..|
|..+O+..+O*O..|
|..+O+..+O*O..|
|..+O+..+O*O..|
|..+O+..+O*O..|
|..+O+..+O*O..|
|..+O+..+O*O..|
+----[SHA256]-----+
esclavo1@ubuntu:~$
```

The screenshot shows a terminal window where an SSH key is being installed and used to connect to a remote host. The output is as follows:

```
esclavo1@ubuntu:~$ ssh-copy-id -t ~/.ssh/id_rsa.pub jjvillavicencio@192.168.1.12
/bbin/ssh-copy-id: INFO: source of key(s) to be installed: '/home/esclavo1/.ssh/id_rsa.pub'
/bbin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter any that are already installed
/bbin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- If you are prompt
ow it is to install the new keys
llavicencio@192.168.1.124's password:
er of key(s) added: 1
try logging into the machine, with: "ssh 'jjvillavicencio@192.168.1.124'"
check to make sure that only the key(s) you wanted were added.
esclavo1@ubuntu:~$
```

Below this, a terminal window shows the connection to the remote host:

```
jjvillavicencio@NS110:~$
File Edit View Search Terminal Help
esclavo1@ubuntu:~$ ssh jjvillavicencio@192.168.1.124
Welcome to Ubuntu 16.04.1 LTS (GNU/Linux 4.4.0-47-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

Pueden actualizarse 21 paquetes,
0 actualizaciones son de seguridad.
```

3.2.2.4. Instalación de Apache Spark 1.4.0

The screenshot shows the Apache Spark download page on <https://spark.apache.org/downloads.html>. It lists the download links for Spark 1.4.0 (Jun 11 2015) for Hadoop 2.4. The download type is set to 'Direct Download'. The download link is [spark-1.4.0-bin-hadoop2.4.tgz](https://spark.apache.org/download/1.4.0-bin-hadoop2.4.tgz).

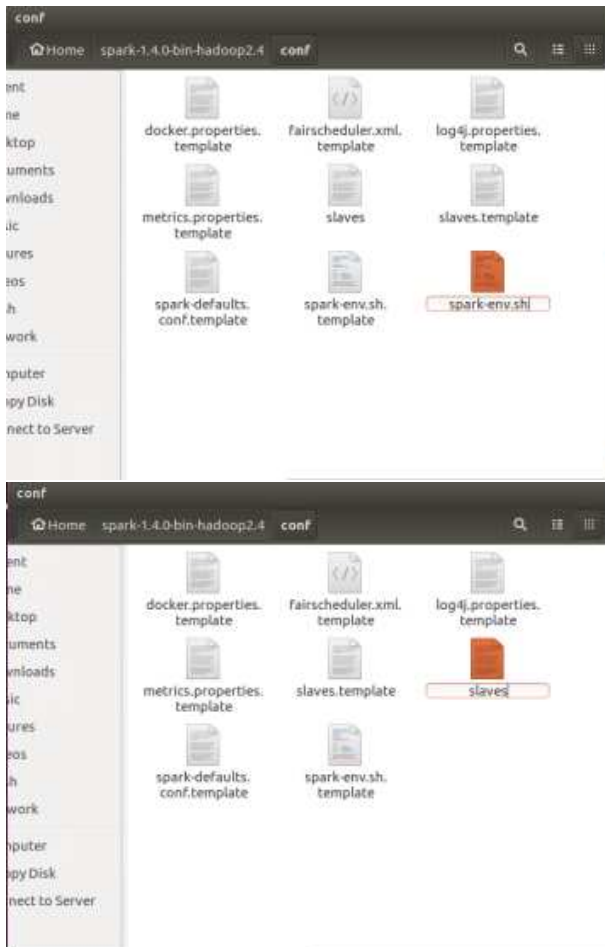
Below the download page, a terminal window shows the installation process:

```
esclavo1@ubuntu:~$ wget https://spark.apache.org/download/1.4.0-bin-hadoop2.4.tgz
esclavo1@ubuntu:~$ tar -xvzf spark-1.4.0-bin-hadoop2.4.tgz
esclavo1@ubuntu:~$ cd spark-1.4.0-bin-hadoop2.4/conf
esclavo1@ubuntu:~/spark-1.4.0-bin-hadoop2.4/conf$ gedit
# Options for the daemons used in the standalone deploy mode
# - SPARK_MASTER_IP, to bind the master to a different IP address or hostname
export SPARK_MASTER_IP=172.16.233.128
# - SPARK_MASTER_PORT / SPARK_MASTER_WEBUI_PORT, to use non-default ports for the master
# - SPARK_MASTER_OPTS, to set config properties only for the master (e.g. "-Dx=y")
# - SPARK_WORKER_CORES, to set the number of cores to use on this machine
export SPARK_WORKER_CORES=1
# - SPARK_WORKER_MEMORY, to set how much total memory workers have to give executors (e.g. 1000m, 2g)
export SPARK_WORKER_MEMORY=800M
# - SPARK_WORKER_PORT / SPARK_WORKER_WEBUI_PORT, to use non-default ports for the worker
# - SPARK_WORKER_INSTANCES, to set the number of worker processes per node
export SPARK_WORKER_INSTANCES=2
# - SPARK_WORKER_DIR, to set the working directory of worker processes
# - SPARK_WORKER_OPTS, to set config properties only for the worker (e.g. "-Dx=y")
# - SPARK_HISTORY_OPTS, to set config properties only for the history server (e.g. "-Dx=y")
# - SPARK_SHUFFLE_OPTS, to set config properties only for the external shuffle service (e.g. "-Dx=y")
```

3.2.2.3. Configuración de acceso SSH entre máquinas

The screenshot shows a terminal window where an SSH key is being generated and used to connect to a remote host. The output is as follows:

```
esclavo1@ubuntu:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/esclavo1/.ssh/id_rsa):
Created directory '/home/esclavo1/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/esclavo1/.ssh/id_rsa.
Your public key has been saved in /home/esclavo1/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:thQAGLxSPBm1sBux40XFRjA2cnPpSRQWqsj/kfDP8 esclavo1@ubuntu
The key's randomart image is:
+----[RSA 2048]----+
|..+O+..+O*O..|
|..+O+..+O*O..|
|..+O+..+O*O..|
|..+O+..+O*O..|
|..+O+..+O*O..|
|..+O+..+O*O..|
|..+O+..+O*O..|
|..+O+..+O*O..|
|..+O+..+O*O..|
+----[SHA256]-----+
esclavo1@ubuntu:~$
```



```

esclavo1@ubuntu: ~/spark-1.4.0-bin-hadoop2.4/bin
File Edit View Search Terminal Help

took 7.092774 s
Pl is roughly 3.14154
16/11/20 14:24:53 INFO SparkUI: Stopped Spark web UI at http://172.16.233.128:4040
16/11/20 14:24:53 INFO DAGScheduler: Stopping DAGScheduler
16/11/20 14:24:53 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
16/11/20 14:24:53 INFO Utils: path = /tmp/spark-f01a5565-4bc8-438f-9447-c099ab8b05a3/blockmgr-fd12010a-82cf-44d6-b20d-fba6f2f7252, already present as root for deletion.
16/11/20 14:24:53 INFO MemoryStore: MemoryStore cleared
16/11/20 14:24:53 INFO BlockManager: BlockManager stopped
16/11/20 14:24:53 INFO BlockManagerMaster: BlockManagerMaster stopped
16/11/20 14:24:53 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
16/11/20 14:24:53 INFO RemoteActorRefProvider$RemotingTerminator: Shutting down remote daemon.
16/11/20 14:24:53 INFO RemoteActorRefProvider$RemotingTerminator: Remote daemon shut down; proceeding with flushing remote transports.
16/11/20 14:24:53 INFO SparkContext: Successfully stopped SparkContext
16/11/20 14:24:53 INFO Utils: Shutdown hook called
16/11/20 14:24:53 INFO Utils: Deleting directory /tmp/spark-f01a5565-4bc8-438f-9447-c099ab8b05a3
esclavo1@ubuntu:~/spark-1.4.0-bin-hadoop2.4/bin$

```

3.2.3. Pruebas de instalación

3.2.3.1. Prueba en nodo máster

```

esclavo1@ubuntu: ~/spark-1.4.0-bin-hadoop2.4/bin
File Edit View Search Terminal Help

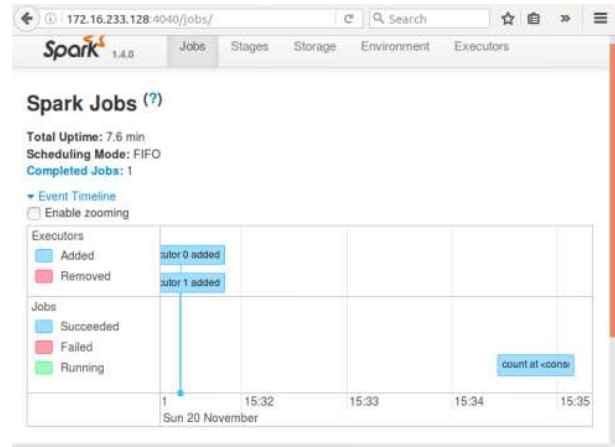
esclavo1@ubuntu:~/spark-1.4.0-bin-hadoop2.4/bin$ ./run-example SparkPi
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
16/11/20 14:24:33 INFO SparkContext: Running Spark version 1.4.0
16/11/20 14:24:34 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
16/11/20 14:24:35 WARN Utils: Your hostname, ubuntu resolves to a loopback address: 127.0.0.1; using 172.16.233.128 instead (on interface ens33)
16/11/20 14:24:35 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
16/11/20 14:24:35 INFO SecurityManager: Changing view acls to: esclavo1
16/11/20 14:24:35 INFO SecurityManager: Changing modify acls to: esclavo1
16/11/20 14:24:35 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(esclavo1); users with modify permissions: Set(esclavo1)
16/11/20 14:24:36 INFO SLF4JLogger: SLF4JLogger started
16/11/20 14:24:37 INFO Remoting: Starting remoting
16/11/20 14:24:37 INFO Remoting: Remoting started; listening on addresses :[akka.tcp://sparkDriver@172.16.233.128:41831]
16/11/20 14:24:37 INFO Utils: Successfully started service 'sparkDriver' on port 41831.
16/11/20 14:24:38 INFO SparkEnv: Registering MapOutputTracker
16/11/20 14:24:38 INFO SparkEnv: Registering BlockManagerMaster
16/11/20 14:24:38 INFO DiskBlockManager: Created local directory at /tmp/spark-f

```

3.2.3.2. Prueba en cluster

The terminal window shows the Spark shell output for a job. It includes logs for the DAGScheduler, TaskScheduler, and TaskSetManager. The job is named 'count at <console>' and is in the 'Running' state. The web browser shows the Spark Jobs page with a table of running applications.

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20161120152945-0000	Spark shell	2	512.0 MB	2016/11/20 15:29:45	esclavo1	RUNNING	3.0 min



4. Implementación de la Aplicación Paralela

4.1. Tema, Problemática y descripción funcionamiento de la Aplicación

Procesamiento de un dataset, el objetivo de esta aplicación es dar una carga de trabajo que exija al cluster, lo que se logrará procesando un archivo de datos y clasificando estos a través de determinados parámetros.[2]

4.2. Algoritmo Secuencial

```
1#!/usr/bin/env python
2# -*- coding: utf-8 -*-
3"""Procesar DataSet en secuencial"""
4
5from time import time
6from collections import Counter
7
8__tiempo_inicial__ = time()
9
10__dataSet__ = open('u.user', 'r')
11__lineas__ = __dataSet__.readlines()
12
13def intervalo_edad(edad):
14    """Intervalos de edad"""
15    if edad < 10:
16        return '0-10'
17    elif edad < 20:
18        return '10-20'
19    elif edad < 30:
20        return '20-30'
21    elif edad < 40:
22        return '30-40'
23    elif edad < 50:
24        return '40-50'
25    elif edad < 60:
26        return '50-60'
27    elif edad < 70:
28        return '60-70'
29    elif edad < 80:
30        return '70-80'
31    else:
32        return '80+'
```



```

33
34 def conversor_edad(data):
35     """pasar edad a un intervalo de
edad"""
36     datos_conv = []
37     for linea in data:
38         datos_conv.append(linea.split("
|"))
39
40     for usuario in datos_conv:
41         usuario[1] = intervalo_edad(int
(usuario[1]))
42     return datos_conv
43
44 def usuarios_coinciden(intervalo, data)
:
45     """Cantidad e usuarios que
coinciden con el intervalo de edad"""
46     coincidencias = []
47     for usuario in data:
48         if usuario[1] == intervalo:
49             coincidencias.append(
usuario)
50     return coincidencias
51
52 def usuarios_profesion(data):
53     """Clasificar usuarios por
profesion"""
54     profesiones = []
55     for usuario in data:
56         profesiones.append(usuario[3])
57     print Counter(profesiones)
58     return Counter(profesiones)
59
60 def extremos(data):
61     """Obtener cantidad de edades
marginales"""
62     extremos_dic = {}
63     extremos_dic['pasados_edad'] = 0
64     extremos_dic['menor_edad'] = 0
65     for usuario in data:
66         if usuario[1] == "70-80":
67             extremos_dic['pasados_edad',
] += 1
68         if usuario[1] == "0-10":
69             extremos_dic['menor_edad']
+= 1
70     return extremos_dic
71
72     __cambiar_edades__ = conversor_edad(
__lineas__)
73     __usuario_filtrados__ =
usuarios_coinciden("20-30",
__cambiar_edades__)
74     __edade_extremos__ = extremos(
__cambiar_edades__)
75
76     print "El total de usuarios es: ", len(
__lineas__)
77     print "Total de usuarios por profesion:
", usuarios_profesion(
__usuario_filtrados__)
78     print "Usuarios entre 0-10: ",
__edade_extremos__['menor_edad']

```

```

79     print "Usuarios entre 70-80: ",
__edade_extremos__['pasados_edad']
80
81     __tiempo_final__ = time()
82     __tiempo_ejecucion__ = __tiempo_final__
- __tiempo_inicial__
83
84     print 'El tiempo de ejecucion fue:',
__tiempo_ejecucion__ #En segundos
85

```

4.3. Algoritmo Paralelo

```

1 from pyspark import SparkContext,
SparkConf
2
3 conf = SparkConf().setAppName('
MyFirstStandaloneApp')
4 sc = SparkContext(conf=conf)
5
6 userRDD = sc.textFile("/usr/lib/
data_backup/opt/spark_usecases/movie/ml
-100k/u.user")
7
8 def parse_N_calculate_age(data):
9     userid, age, gender, occupation,
zip = data.split("|")
10     return userid, age_group(int(age))
, gender, occupation, zip, int(age)
11
12 def age_group(age):
13     if age < 10:
14         return '0-10'
15     elif age < 20:
16         return '10-20'
17     elif age < 30:
18         return '20-30'
19     elif age < 40:
20         return '30-40'
21     elif age < 50:
22         return '40-50'
23     elif age < 60:
24         return '50-60'
25     elif age < 70:
26         return '60-70'
27     elif age < 80:
28         return '70-80'
29     else:
30         return '80+'
31
32 data_with_age_bucket = userRDD.map(
parse_N_calculate_age)
33
34 RDD_20_30 = data_with_age_bucket.filter
(lambda line: '20-30' in line)
35
36 freq = RDD_20_30.map(lambda line: line
[3]).countByValue()
37
38 print "El total de usuarios es: ",
userRDD.count()
39

```

```

40 print "Total de usuarios por profesion
    ", dict(freq)
41
42 Under_age = sc.accumulator(0)
43 Over_age = sc.accumulator(0)
44
45 def outliers(data):
46     global Over_age, Under_age
47     age_grp = data[1]
48     if age_grp == "70-80":
49         Over_age += 1
50     if age_grp == "0-10":
51         Under_age += 1
52     return data
53
54 df = data_with_age_bucket.map(outliers)
55 .collect()
56
57 print "Usuarios entre 0-10: ",
58 Under_age
59 print "Usuarios entre 70-80: ",
60 Over_age

```

4.3.1. Descripción detallada de las funciones y librerías paralela que se utiliza

Acerca del dataset

U.user: Información demográfica de usuarios, separada por barras verticales con los siguientes campos, user id — age — gender — occupation — zip code

Ejemplo:

```

1 1|24|M|technician|85711
2 2|53|F|other|94043
3 3|23|M|writer|32067
4 4|24|M|technician|43537
5 5|33|F|other|15213
6 6|42|M|executive|98101
7 7|57|M|administrator|91344
8 8|36|M|administrator|05201
9 9|29|M|student|01002
10

```

Enlace para descargar dataset:

<http://grouplens.org/datasets/movielens/>

Leer datos con Spark RDD (Resilient Distributed Dataset)

```

1 userRDD = sc.textFile("/usr/lib/
    data_backup/opt/spark_usecases/movie/ml
    -100k/u.user")
2

```

Contar numero de registros

```

1 userRDD.count()
2

```

Vamos a crear una función definida por el usuario para dividir a los usuarios en grupos de edad:

```

1 def parse_N_calculate_age(data):
2     userid, age, gender, occupation, zip =
3     data.split("|")
4     return userid, age_group(int(age)),
5     gender, occupation, zip, int(age)
6
7 def age_group(age):
8     if age < 10:
9         return '0-10'
10    elif age < 20:
11        return '10-20'
12    elif age < 30:
13        return '20-30'
14    elif age < 40:
15        return '30-40'
16    elif age < 50:
17        return '40-50'
18    elif age < 60:
19        return '50-60'
20    elif age < 70:
21        return '60-70'
22    elif age < 80:
23        return '70-80'
24    else:
25        return '80+'

```

```

1 data_with_age_bucket = userRDD.map(
2     parse_N_calculate_age)

```

Ahora, vamos a analizar el grupo de edad "20-30" para su posterior análisis

```

1 RDD_20_30 = data_with_age_bucket.filter
2     (lambda line: '20-30' in line)

```

Vamos a contar el número de usuarios por su profesión en el age_group determinado 2030

```

1 freq = RDD_20_30.map(lambda line: line
2     [3]).countByValue()

```


Ahora, usaremos Acumuladores para la detección de valores aleatorios en el conjunto de datos de película anterior. Supongamos que cualquier persona que caiga en el grupo de edad 70-80 sea marcada como `over_age` y cualquier persona que caiga en el grupo de edad 0-10 es marcado como `under_age`.

```
1 Under_age = sc.accumulator(0)
2 Over_age = sc.accumulator(0)
3
```

```
1 def outliers(data):
2     global Over_age, Under_age
3     age_grp = data[1]
4     if age_grp == "70-80":
5         Over_age += 1
6     if age_grp == "0-10":
7         Under_age += 1
8     return data
9
```

4.3.2. Descripción detallada del funcionamiento de la aplicación

Básicamente la aplicación realizara tres operaciones con un dataset de usuarios:

- **Clasificación de usuarios por edad:**
en el dataset a utilizar cada usuario tiene su edad con la intención de captar y procesar la mayor cantidad de usuarios se los agrupara en intervalos de edad por ejemplo de 20-30
- **Usuarios por profesión:**
Una vez que tenemos los usuarios que coinciden con el intervalo de edad especificado (20-30), contaremos cuantos usuarios pertenecen a determinadas profesiones que se encuentran registras en el dataset.
- **Edades menores y mayores:**
El ultimo calculo a realizar es contar la cantidad de usuarios que tienen edad máxima (70-80) y edad mínima(0-10), esto se lo hará de la totalidad del dataset.

5. Integración del Cluster y la Aplicación

5.1. Plan de Pruebas

El objetivo del plan de pruebas a realizar es comprobar la optimización de recursos cuando se trabaja con algoritmos paralelos bajo una arquitectura grid, en comparación con la versión secuencial desarrollada de igual manera dentro del proyecto

Pruebas de Conexión: Para verificar la conectividad se ha elaborado el siguiente proceso:

- Creación de llaves publicas SSH en los nodos worker para que el nodo master pueda acceder sin ingresar contraseñas.
- Levantar el servicio del cluster `/sbin/start-all.sh` que conecta todos los nodos configurados
- Ejecutar la interfaz web y constatar los nodos worker.

Pruebas de Ejecución: Las pruebas de ejecución se realizan tanto para la versión secuencial del algoritmo como para la paralela.

- En la versión secuencial del programa, se trabajará con el archivo `data.secuencial.py`, en el cual el algoritmo esta programado con funciones normales.
- En la versión paralela se trabajará con el archivo `data.paralelo.py` en el cual el algoritmo esta programado con la librería `pyspark` exclusiva para el cluster.
- La data a utilizar se encuentra en el archivo `u.user` con variaciones de 19803, 39606 y 60292 datos, el mismo que se encontrará disponible en cada nodo

5.2. Comparación mediante los criterios de evaluación de las versión secuencial y paralela

Tiempos de respuesta:

DATOS	SECUENCIAL	PARALELO
19803	0.03461	29.360435
39606	0.07382	19.508621
60292	0.10606	17.20789

Coste secuencial:

$$Tej * \#Proc = 0,64347 * 5 = 3,2173$$

Coste paralelo:

$$Tej * \#Proc = 22,02564 * 5 = 110,12824$$

Tiempo de rendimiento Secuencial:

$$Tf - Ti = 0,10606 - 0,03461 = 0,07145$$

Tiempo de rendimiento Paralelo:

$$Tf - Ti = 17,20789 - 29,360435 = 12,15254$$

Aceleración:

$$Ts / Tp = 0,07145 / 12,15254 = 0,005879$$

Eficiencia:

$$Ts / (p(Tp)) * 100 = 0,07145 / 5(12,15254) = 0,117\%$$

Valor de ejecución:

$$0 \leq Sp < n \quad 0 \leq 0,005879 < 110,1282$$

Eficiencia:

$$0 \leq tf < 1 \quad 0 \leq 0,117\% < 1$$

5.3. Gráficas de la comparación de las dos versiones

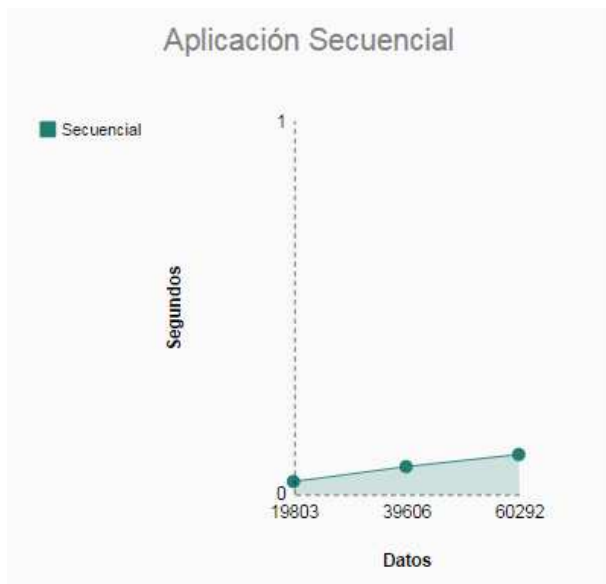


Figura 5. Tiempos respuesta secuencial.



Figura 6. Tiempos respuesta paralelo.

5.4. Análisis del rendimiento

Una vez obtenidos los tiempos de ejecución de la aplicación tanto en paralelo como en secuencial, nos podemos dar cuenta primero, que para trabajar con Apache Spark se debe usar grandes volúmenes de datos debido que esta orientado a BigData, segundo que con la cantidad de datos que se realizó las pruebas la ejecución secuencial obtuvo mejores tiempos que la ejecución paralela, y tercero que se puede notar que mientras aumenta el volumen de datos a procesar Apache Spark va reduciendo su tiempo de respuesta muy contrario a la ejecución en secuencial que va aumentando sus tiempos de respuesta.

6. Conclusiones y Recomendaciones

- Spark es un entorno con múltiples funcionalidades, que simplifica enormemente la gestión de grandes cantidades de datos de forma transparente. La magia de Spark reside en que el programa escrito en el ejemplo, puede ejecutarse tanto en un clúster local, como en un clúster distribuido compuesto por multitud de nodos sin cambiar ni una sola línea de código.
- Apache Spark es una herramienta útil y eficiente para tareas de procesamiento masivo de datos. Está en constante desarrollo y se actualiza frecuentemente. Además, su documentación es muy completa y la comunidad cada vez se hace más grande.
- Se recomienda usar Apache Spark para trabajar con procesamiento de DataSets o grandes volúmenes de información, para aprovechar el cluster.
- Se recomienda familiarizarse con la librería pyspark para realizar programas completamente compatibles con Spark desarrollados en Python.

Referencias

- [1] Mario Pérez Esteso. Apache spark: qué es y cómo funciona, sep 2015.
- [2] <https://www.dezyre.com/>. Pyspark tutorial-learn to use apache spark with python, dec 2015.
- [3] Luis Mora. ¿qué es la computación grid?, aug 2008.
- [4] Francisco Javier Pulido. Spark (iii) - ¿cómo crear y configurar un clúster?, may 2014.