

Big Data: Implementación de Hadoop 2.7 sobre una placa Raspberry pi v3

César Martín Cruz Salazar[†]

Escuela de Ciencia de la Computación - Facultad de Ciencias.

Universidad Nacional de Ingeniería;

[†]*ccruz@uni.edu.pe*

Recibido el 30 de Septiembre del 2015; aceptado el 14 de Octubre del 2015

Este trabajo de investigación trata del uso de arquitecturas de bajo consumo energético y bajo costo en un sistema tipo clúster y la instalación de Big Data Analytics Framework sobre esta arquitectura para poder garantizar el uso de Big Data. Como parte del trabajo, hicimos una prueba sobre una arquitectura ARM que consistía de un computador compacto de única placa llamada Raspberry pi versión 3 modelo B que formaba parte de un clúster hadoop de un solo nodo SNHC(Single Node Hadoop Cluster) y en la cual se instaló el software Hadoop 2.7.0, se configuró este y se ejecutaron ejemplos, mostrando así una implementación fiable y exitosa de Hadoop sobre un clúster.

Palabras Claves: clúster, raspberry pi versión 3, programación paralela, big data, hadoop versión 2.7, programación distribuida, clúster educativo.

This research is about the use of energy-efficient architectures of low cost in a system cluster type and the installation of Big Data Analytics Framework on this architecture to guarantee the use of Big Data. As part of the work, we did a test on an ARM architecture consisting of a compact computer single board called Raspberry Pi version 3 model B it was part of a Hadoop cluster from a single node SNHC (Single Node Hadoop Cluster) and with Hadoop 2.7.0 installed, this software was configured and examples were executed, showing a reliable and successful implementation of Hadoop on a cluster.

Keywords: cluster, raspberry pi version 3, parallel programming, big data, hadoop version 2.7, distributive programming, educational cluster.

1 Introducción

Sobre la base de las necesidades y demandas cada vez mayores de la ciudadanía acerca de Big Data es que iniciamos esta investigación para proporcionar una mejor arquitectura e infraestructura con bajo consumo de energía y bajo costo, para el análisis de Big Data. El incremento continuo en volumen, variedad y velocidad de Big Data expone a los Centros de Datos a un problema con el consumo de energía[1]. Los servidores de estos Centros de Datos consumen mucha energía, necesitan una gran cantidad de equipos de refrigeración y ocupan espacio de grandes extensiones. En la actualidad, uno de los problemas más grandes que los centros de datos enfrentan es asegurar la eficiencia y la refrigeración barata de todo el hardware con el uso de energía más baja posible. El hardware utilizado en los Centros de Datos es a menudo costoso, hambriento de potencia y con una necesidad de refrigeración. El costo de los equipos de refrigeración a menudo es igual o supera el costo del hardware del servidor. Las empresas y los investigadores están constantemente buscando maneras de mejorar la eficiencia y bajar el costo en los Centros de Datos[2]. Tradicionalmente, los Centros de Datos emplean nodos de servidores x86-64bits con una potencia de decenas a cientos de Watts. Pero últimamente, sistemas de baja potencia originalmente desarrollados para dispositivos móviles han visto una mejora significativa en rendimiento. Es-

tas mejoras podrían conducir a la adopción de tales sistemas pequeños para la implementación de servidores, como han anunciado actores mayores de esta industria. Nos referimos a Dell, HP y AppliedMicro que han lanzado prototipos de servidores basados en procesadores ARM. Aún AMD, que históricamente ha comercializado servidores con procesadores basados en la arquitectura x86, tiene como objetivo lanzar servidores basados en ARM. Naturalmente, esto plantea la pregunta de la viabilidad de servidores ARM de bajo consumo como contendientes para los servidores tradicionales x64 de Intel/AMD para el procesamiento de grandes volúmenes de datos (Big Data). Si un clúster basado en ARM con menor consumo de energía y costo puede igualar el rendimiento de un clúster tradicional de Intel/AMD, esto podría marcar el comienzo de una nueva era de la informática verde(green computing) que puede ayudar a alcanzar nuevos niveles de rendimiento y eficiencia en los costos para el análisis de Big Data. Nuestra experiencia obtenida en la implementación de clústeres de bajo costo y bajo consumo energético[3] nos anima a realizar este nuevo reto. Es así, que en este artículo presentamos además de la instalación y configuración del software Hadoop 2.7.0. resultados de tiempo obtenidos al ejecutar el programa QuasiMonte-Carlo para el cálculo de Pi y el programa WordCount para diferentes tamaños de archivos de texto. También, realizamos comparaciones con otros sistemas de cómputo. El Clúster de único nodo(SNHC, Single Node Hadoop

Cluster) que se muestra en la figura 2 es lo que usamos en esta oportunidad.

2 Especificaciones del clúster

Proporcionamos una descripción de sus componentes de hardware y del software instalado.

2.1 Hardware

Se usó un Raspberry Pi(RPI) versión 3. Esta placa es del tamaño de una tarjeta de crédito ver figura 1. Cada uno cuenta con 1 Gigabyte de RAM, un sistema sobre un Chip(SoC, System on a Chip) Broadcom BCM2837, que integra un procesador quad core ARM Cortex-A53 de 1.2GHz, una arquitectura de 64 bits, un GPU Broadcom VideoCore IV con salida HDMI y salida de vídeo RCA, y un controlador USB. La placa también contiene, un adaptador para puerto Ethernet externo de 10/100 Mbits, y un adaptador de cuatro puertos USB 2.0. Ver tabla 1. El almacenamiento local del sistema operativo y archivos de datos es una tarjeta de memoria microSD de 32 gigabytes que se coloca en una ranura acondicionada en la placa. La placa RPI3 se alimenta con un adaptador de voltaje de 5V a 2.5A.



Figure 1. Raspberry Pi 3 modelo B

SoC	Broadcom BCM2837
CPU	1.2GHz quad-core ARM Cortex-A53
GPU	Dual Core VideoCore
RAM	1GB LPDDR2 SDRAM
Ethernet	Un jack 10/100 RJ45
USB	Cuatro conectores USB 2.0
Memoria	Micro SD card slot

Tabla 1:Especificaciones para Raspberry pi 3 modelo B



Figure 2. Clúster de único nodo hecho con una Raspberry pi versión 3 modelo B .

2.2 Software

Se instaló como sistema operativo Raspbian Jessie, que es una versión optimizada de la distribución Debian GNU/Linux para la Raspberry pi. Para ello se preparó una tarjeta MicroSD con una imagen del Raspbian Jessie, descargada del sitio web de la Fundación Raspberry pi(raspberrypi.org).

2.2.1 Hadoop

La tecnología conocida como Apache Hadoop versión 2 (figura 3) es un framework de software open-source para almacenamiento y procesamiento a gran escala de un conjunto de datos así como para desarrollar aplicaciones distribuidas sobre clusters de computadores de hardware básico acogido por el Apache Software Foundation. Este framework contiene varios módulos: Hadoop Common, Hadoop Distributed File System(HDFS), Hadoop YARN y Hadoop MapReduce.

Uno de los prerequisites para instalar y usar Hadoop es primero instalar Java en el sistema operativo Raspbian Jessie.

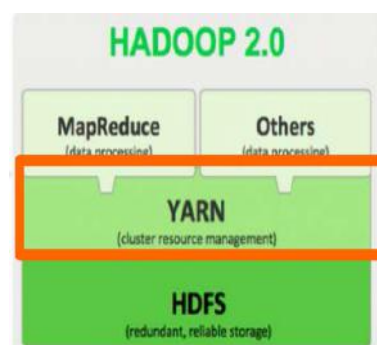


Figure 3. Hadoop versión 2.7.0.

2.2.2 HDFS

HDFS: Hadoop Distributed FileSystem. Es un framework que suministra almacenamiento de datos [?]. Para

el procesamiento de los datos con MapReduce es necesario que los archivos de texto se encuentren almacenados de manera distribuida. Este hecho es posible con HDFS. HDFS ha sido diseñado para ejecutarse sobre clústeres de computadores.

En el uso de Hadoop para ejecutar aplicaciones MapReduce, hay terminologías claves usadas en la tecnología que debemos entender: Namenode(nodo maestro) y Datanode(en el clúster se usa HDFS para almacenar datos replicados).

2.2.3 YARN

Mejora la potencia de un Hadoop Cluster. Suministra gran escalabilidad. La potencia de procesamiento y centros de datos continua creciendo rápidamente porque YARN research manager se enfoca exclusivamente en la calendarización. Puede gestionar aquellos enormes clústeres rápidamente y fácilmente. YARN es completamente compatible con MapReduce, para el seteo de MapReduce daemons. Desde que estamos ejecutando MapReduce usando YARN, el MapReduce jobtracker y tasktrackers son reemplazados con un único resource manager corriendo sobre el namenode.

2.2.4 MapReduce

Es un modelo de programación desarrollado por Google en primer lugar para permitir procesamiento simple distribuido de grandes conjuntos de datos. En un MapReduce el programa consta de dos pasos :El paso Mapa y la etapa de reducción. El paso mapa realiza filtraje y clasificación. La etapa de reducción, realiza más cálculos en la salida de los mapas, y esto suele ser una operación de recorte de datos. Dependiendo del programa del mapa y/o reducción las tareas pueden ser paralelizadas. Los programadores encuentran el sistema fácil de usar: mas de 10 mil distintos programas MapReduce han sido implementados internamente en Google en los cinco años pasados, y un promedio de cien mil tareas MapReduce son ejecutados sobre clústeres de Google cada día, procesando un total de mas de veinte petabytes de datos por día [4].

3 Instalación y Configuración

3.1 HADOOP 2.7.0

Para ver la instalación y configuración de Hadoop 2.7.0 consultar el Apéndice.

```
pi@pi3Master:~$ hadoop version
Hadoop 2.7.0
Subversion https://git-wip-us.apache.org/repos/asf/hadoop.git -r d4c8d4d4d203c934e8074b31289a28724c0842cf
Compiled by jenkins on 2015-04-10T18:40Z
Compiled with protoc 2.5.0
From source with checksum a9e90912c37a35c3195d23951fd18f
This command was run using /usr/local/hadoop/share/hadoop/common/hadoop-common-2.7.0.jar
```

Figure 4. Versión de Hadoop en el Clúster de un solo nodo.

Para empezar a usar los servicios de Hadoop. Se ejecutan antes algunos scripts: El script shell start-dfs.sh y start-yarn.sh.

```
pi@pi3Master:~$ start-dfs.sh
16/08/13 14:03:09 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Starting namenodes on [localhost]
localhost: starting namenode, logging to /usr/local/hadoop/logs/hadoop-pi-namenode-pi3Master.out
localhost: starting datanode, logging to /usr/local/hadoop/logs/hadoop-pi-datanode-pi3Master.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-pi-secondarynamenode-pi3Master.out
16/08/13 14:03:40 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
```

Figure 5. Ejecución de script start-dfs.sh.

```
pi@pi3Master:~$ start-yarn.sh
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn-pi-resourcemanager-pi3Master.out
localhost: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-pi-nodemanager-pi3Master.out
```

Figure 6. Ejecución de script start-yarn.sh.

Con el comando jps chequeamos si todos los servicios estan corriendo como se espera. Ver figura 7.

```
pi@pi3Master:~$ jps
1393 NodeManager
1700 Jps
1142 SecondaryNameNode
843 NameNode
941 DataNode
1295 ResourceManager
```

Figure 7. Ejecución de jps.

Para detener los servicios, se ejecutan los scripts shell en este orden: stop-yarn.sh y stop-dfs.sh. Con nuestros servicios corriendo, podemos ahora empezar a ejecutar ejemplos.

3.2 Programa QuasiMonteCarlo para cálculo de Pi

```
pi@pi3Master:~$ hadoop jar /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.0.jar pi 5 50
```

Resultado obtenido: Job Finished in 175.156 seconds Estimated value of Pi is 3,16800000000000000000

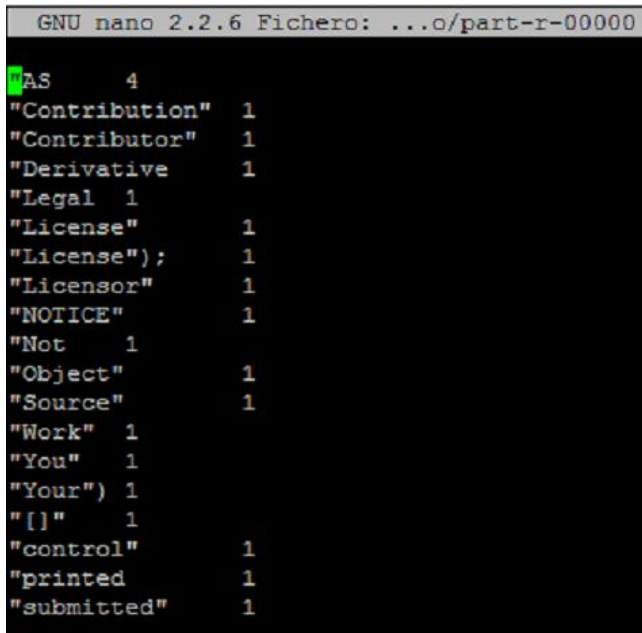
3.3 Programa WordCount

El código fuente del programa WordCount se encuentra en el apéndice de este artículo. Su propósito es tomar un archivo de entrada y retornar otro archivo con las cuentas de cada palabra localizada en el archivo de entrada. Ejecutamos este programa con diferentes tamaños de archivos de texto para probar el clúster de un solo nodo.

3.3.1 Archivo: LICENSE.txt de 16kb

```
pi@pi3Master:~ $ hdfs dfs -copyFromLocal \
LICENSE.txt /license.txt
16/08/13 22:35:19 WARN util.NativeCodeLoader: \
Unable to load native-hadoop library for your \
platform... using builtin-java classes where\
applicable
pi@pi3Master:~ $ time hadoop jar /usr/local/ \
hadoop/share/hadoop/mapreduce/hadoop- \
mapredexamples-2.7.0.jar wordcount /license.txt \
/license-resultado
```

Tiempo obtenido: 1m9,770s. Del archivo obtenido se visualiza la primera de 14 páginas(1 de 14):



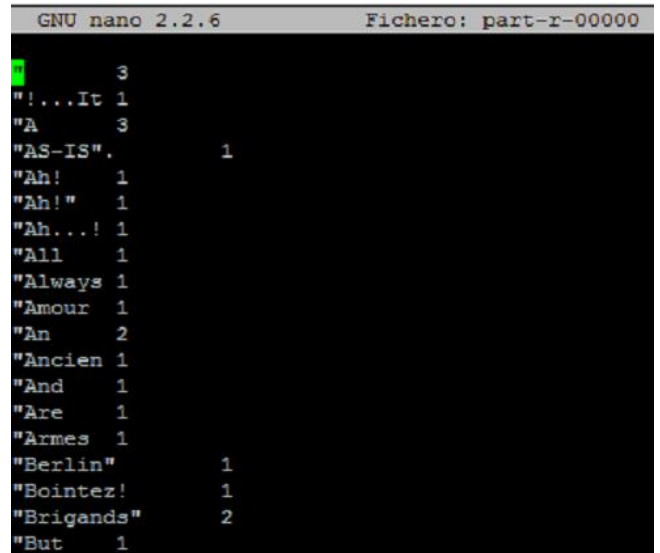
```
GNU nano 2.2.6 Fichero: ...o/part-r-00000
"AS" 4
"Contribution" 1
"Contributor" 1
"Derivative" 1
"Legal" 1
"License" 1
"License");" 1
"Licensor" 1
"NOTICE" 1
"Not" 1
"Object" 1
"Source" 1
"Work" 1
"You" 1
"Your")" 1
"[]" 1
"control" 1
"printed" 1
"submitted" 1
```

Figure 8. Resultado del contador de palabras del archivo LICENSE.txt.

3.3.2 Archivo: smallfile.txt de 2 Megabytes

```
pi@pi3Master:~ $ time hadoop jar /usr/local/ \
hadoop\share\hadoop\mapreduce\hadoop-mapreduce\
-examples-2.7.0.jar wordcount /smallfile.txt \
/smallfile-resultado
```

Tiempo obtenido: 1m44,596s. Se visualiza la primera página del archivo obtenido:



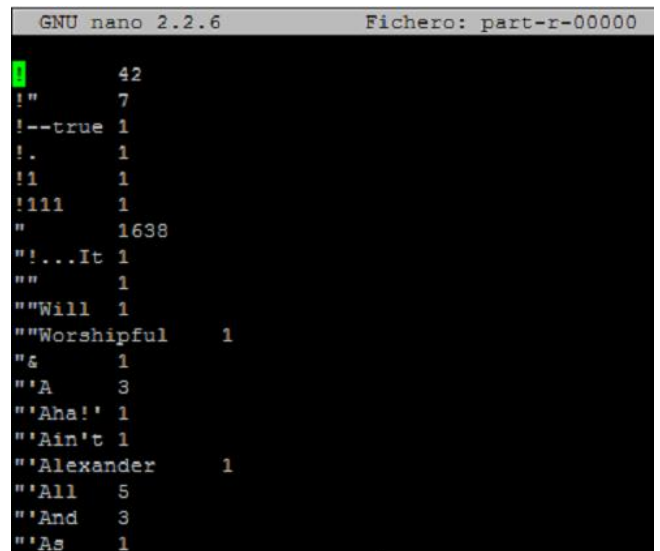
```
GNU nano 2.2.6 Fichero: part-r-00000
"!" 3
"!...It" 1
"A" 3
"AS-IS" 1
"Ah!" 1
"Ah!" 1
"Ah...!" 1
"All" 1
"Always" 1
"Amour" 1
"An" 2
"Ancien" 1
"And" 1
"Are" 1
"Armes" 1
"Berlin" 1
"Bointez!" 1
"Brigands" 2
"But" 1
```

Figure 9. Resultado del contador de palabras del archivo smallfile.txt.

3.3.3 Archivo: mediumfile.txt de 35Mb

```
pi@pi3Master:~ $ time hadoop jar /usr/local/ \
hadoop/share/hadoop/mapreduce/hadoop-mapreduce\
-examples-2.7.0.jar wordcount /mediumfile.txt\
/mediumfile-resultado
```

Tiempo obtenido: 6m4,984s. En la figura 10, se visualiza la primera página del archivo obtenido.



```
GNU nano 2.2.6 Fichero: part-r-00000
"!" 42
!--true 1
!. 1
!1 1
!111 1
" 1638
"!...It" 1
"" 1
""Will" 1
""Worshipful" 1
"€" 1
"'A" 3
"'Aha!" 1
"'Ain't" 1
"'Alexander" 1
"'All" 5
"'And" 3
"'As" 1
```

Figure 10. Resultado del contador de palabras del archivo mediumfile.txt.

3.3.4 Página Web para monitoreo:

<http://192.168.0.130:8088>.

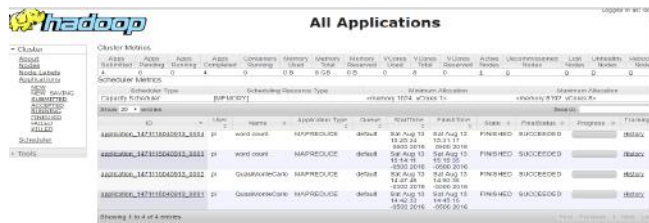


Figure 11. Visualización de los procesos ejecutados.

Clúster	Tamaño archivo	Tiempo(s)
RPI v2 de 3 Nodos	2MB	1m41s ref. [6]
SNHC	2MB	1m44.596s
RPI v2 de 3 Nodos	35MB	2m:22s ref. [6]
RPI mB 8 nodos	35MB	8m35s [5]
SNHC	35MB	6m4.984s
Standard Laptop	35MB	18.07s [5]
Standard Desktop	35MB	14.50s [5]

Tabla 3: Clúster de un solo nodo(SNHC) vs otros Clústeres

3.3.5 Página Web:

http://192.168.0.130:50070.

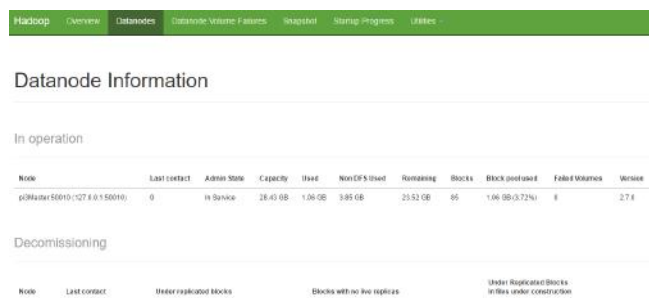


Figure 12. Visualización de los nodos activos.

4 Análisis de tiempos comparativo de Clústeres

Empezamos ante con una visualización de las especificaciones de las máquinas involucradas en las comparaciones.

4.1 Especificaciones de máquinas

La referencia en la tabla se dan las especificaciones de las máquinas usadas.

Computador	Núcleos	Procesador	RAM
Desktop	4	Intel i7-3.5GHz	4 GB
Laptop	2	Intel i7-1.8GHz	4 GB
RaspberryPi	1	ARM11-700MHz	512MB

Tabla 2: Especificaciones de máquinas[5]

4.2 Tabla de resultados de tiempos

La tabla 3 muestra los resultados de tiempos obtenidos para diferentes tamaños de archivos de texto y diferentes tipos de clústeres.

5 Conclusiones

En un clúster de un solo nodo de Raspberry Pi 3 se instaló exitosamente Hadoop versión 2.7.0. Este hecho tiene un gran significado porque nos permite con un costo de ~\$70 ejecutar software de Big Data. Este precio no incluye los precios de monitor y teclado. Así mismo, este equipo cumple muy bien el rol educativo ya que permite instalar y probar diversos softwares libres que son tendencia.

6 Trabajo Futuro

Como una continuidad de este trabajo se instalará en un futuro cercano Hadoop 2.7.2 en un Clúster multinodo basado en varias placas Raspberry pi 3 u otro tipo de placa de mejores características.

7 Agradecimientos

Agradecemos a Dios, a la UNI y a la Facultad de Ciencias.

8 Apéndice

8.1 Instalación y configuración de Hadoop 2.7.0 en un clúster de un solo nodo

```
$ sudo apt-get update

$ java -version

$ sudo apt-get install rsync

$ ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa

$ cat ~/.ssh/id_dsa.pub >> ~/.ssh \
/authorized_keys

$ wget -c http://apache.mirrors.lucidnetworks.\
net/hadoop/common/hadoop-2.7.0/ \
hadoop-2.7.0.tar.gz

$ tar zxvf hadoop-2.7.0.tar.gz

$ sudo mv hadoop-2.7.0 /usr/local/hadoop
```

```

$ update-alternatives --config java

$ sudo nano ~/.bashrc

#Hadoop Variables
export JAVA_HOME=/usr/lib/jvm/ \
jdk-8-oracle-arm32-vfp-hflt
export HADOOP_HOME=/usr/local/hadoop
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=\
$HADOOP_HOME/lib/native
export HADOOP_OPTS="-Djava.library.path= \
$HADOOP_HOME/lib"

$ source ~/.bashrc

$ cd /usr/local/hadoop/etc/hadoop

$ sudo nano hadoop-env.sh

#The java implementation to use.
export JAVA_HOME="/usr/lib/ \
jvm/jdk-8-oracle-arm32-vfp-hflt "

$ sudo nano core-site.xml
<configuration>
<property>
<name>fs.defaultFS</name>
<value>hdfs://localhost:9000</value>
</property>
</configuration>
$ sudo nano yarn-site.xml
<configuration>
<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>
<property>
<name>yarn.nodemanager.aux-services.mapreduce. \
shuffle.class</name>
<value> org.apache.hadoop.mapred. \
ShuffleHandler</value>
</property>
</configuration>

$ sudo cp mapred.site.xml.template \
mapred-site.xml

$ sudo nano mapred-site.xml

<configuration>
<property>
<name>mapreduce.framework.name</name>
<value>yarn</value>
</property>

```

```
</configuration>
```

```
$ sudo nano hdfs-site.xml
```

```
<configuration>
```

```
<property>
```

```
<name>dfs.replication</name>
```

```
<value>1</value>
```

```
</property>
```

```
<property>
```

```
<name>dfs.namenode.name.dir</name>
```

```
<value>file:/usr/local/hadoop/hadoop_data \
/hdfs/namenode</value>
```

```
</property>
```

```
<property>
```

```
<name>dfs.datanode.data.dir</name>
```

```
<value>file:/usr/local/hadoop/hadoop_data \
/hdfs/datanode</value>
```

```
</property>
```

```
</configuration>
```

```
$ cd
```

```
$ mkdir -p /usr/local/hadoop/hadoop_data \
/hdfs/namenode
```

```
$ mkdir -p /usr/local/hadoop/hadoop_data \
/hdfs/datanode
```

```
$ sudo chown pi:pi -R /usr/local/hadoop
```

```
$ hdfs namenode -format
```

```
$ start-dfs.sh
```

```
$ start-yarn.sh
```

```
$ jps
```

```
http://192.168.0.130:8088/
```

```
http://192.168.0.130:50070/
```

8.2 Programa WordCount.java

```

package org.myorg;
import java.io.IOException;
import java.util.*;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;

public class WordCount {

public static class Map extends \
MapReduceBase implements Mapper<LongWri$
private final static IntWritable \
one = new IntWritable(1);
private Text word = new Text();

```

```

public void map(LongWritable key, \
Text value, OutputCollector<T$
String line=value.toString();
StringTokenizer tokens = new \
StringTokenizer(line);
while (tokens.hasMoreTokens()) {
    total += values.next().get();
}
output.collect(key, new IntWritable(total));
}
}
public static void main(String[] args) \
throws Exception{

String input = args[0];
String output = args[1];
String startupmsg = String.format\
("Word Count job started getti$
System.out.println(startupmsg);

JobConf configuration = new JobConf \
(WordCount.class);

configuration.setJobName("wordcount");
configuration.setOutputKeyClass(Text.class);
configuration.setOutputValueClass( \
IntWritable.class);

configuration.setMapperClass(Map.class);
configuration.setCombinerClass \
(Reduce.class);
configuration.setReducerClass \
(Reduce.class);
configuration.setInputFormat \
(TextInputFormat.class);
configuration.setOutputFormat \
(TextOutputFormat.class);

FileInputFormat.setInputPaths \
(configuration,new Path(input));
FileOutputFormat.setOutputPath \
(configuration,new Path(output));
JobClient.runJob(configuration);
}
}

```

-
1. Dumitrel Loghin, Bogdan Marius Tudor, Hao Zhang, Beng Chin Ooi, Yong Meng Teo. A performance study of big data on small nodes. Proceedings of the VLDB Endowment, Vol. 8, No. 7, 2015.
 2. Nick Schot. Feasibility of raspberry pi 2 based micro data centers in big data applications, 23th Twente Student Conference on IT, June, 2015,.
 3. Cesar Martin Cruz Salazar. Medidas de rendimiento y comparacion entre el cluster Cruz I y el cluster Cruz II. *REVCUNI*, 17(1):9-16, 8 2014.
 4. Vijayakumar S, Dr.M.Balamurugan, Ranjani K. Big data: Hadoop cluster deployment on arm architecture. *IJAR-CCE*, Vol.4, 1, 2015.
 5. Shreyyas Vanarase, Tommy Mark. Distributed computing systems with raspberry pi. Georgia Institute of Technology-CX 4140, 2014.
 6. Raspberry pi 2 hadoop 2 cluster. http://www.widriksson.com/raspberry-pi-2-hadoop-2-cluster/#Performance_comparison_to_Raspberry_PI_1_Model_B.512mb
 7. Cesar Martin Cruz Salazar. Taller de big data y hadoop, UNI, 2015.