UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingenierías

Funcionalidad y eficiencia de Hadoop en clúster de Raspberry Pi

Sistematización de experiencias prácticas de investigación y/o intervención..

Galo Ernesto Rosero Abad

Ingeniería en Sistemas

Trabajo de titulación presentado como requisito
para la obtención del título de
ingeniero en sistemas

Quito, 8 de mayo de 2018

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

COLEGIO CIENCIAS E INGENIERÍAS

HOJA DE CALIFICACIÓN
DE TRABAJO DE TITULACIÓN

Funcionalidad y eficiencia de Hadoop en clúster de Raspberry pi

Galo Ernesto Rosero Abad

Calificación: _____

Nombre del profesor, Título académico: Fausto Pasmay , MS

Firma del profesor: _____

Quito, 8 de mayo de 2018

Derechos de Autor

Por medio del presente documento certifico que he leído todas las Políticas y Manuales de la Universidad San Francisco de Quito USFQ, incluyendo la Política de Propiedad Intelectual USFQ, y estoy de acuerdo con su contenido, por lo que los derechos de propiedad intelectual del presente trabajo quedan sujetos a lo dispuesto en esas Políticas.

Asimismo, autorizo a la USFQ para que realice la digitalización y publicación de este trabajo en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Firma del estudiante: _____

Nombres y apellidos: Galo Ernesto Rosero Abad

Código: 00111293

Cédula de Identidad: 1716284425

Lugar y fecha: Quito, 8 de mayo de 2018

RESUMEN

Este trabajo presenta como crear y utilizar un clúster de computadores de placa simple, específicamente de Raspberry Pi 3 y probar su desempeño para manejar Big Data en entornos de pocos recursos financieros o con propósitos educativos. Para probar el desempeño de este clúster se configuraron 3 nodos y se utilizó el framework Hadoop de Apache para realizar tareas asociadas a MapReduce, el cual es una técnica muy utilizada en la minearía de datos en general y permite medir el desempeño de una forma concisa del sistema distribuido y probar su escalabilidad. A lo largo del documento se presenta la información clave de configuración e instalación de Hadoop para Raspberry pi, sobre todo el manejo de memoria y recursos, los cuales son limitados siendo esta la parte más importante del manejo del sistema. Se llegó a la conclusión que si es posible manejar Hadoop y MapReduce en los Raspberry pi, pero para que sea viable usar Raspberry pi para manejar BigData se debería agregar mayor cantidad de nodos, permitiendo así comparar el desempeño al de sistemas más grandes sin tener que hacer grandes gastos de dinero. Las pruebas de desempeño se realizaron con TeraSort y TestDFSIO.

Palabras clave: Raspberry pi 3, Hadoop MapReduce, SBC, TeraSort, TestDFSIO.

ABSTRACT

This work presents how to create and use a cluster of single board computers, and test its performance to handle Big Data on environments of low financial resources or for educational purposes. The cluster was conformed by Raspberry Pi 3 boards. To test the performance of this cluster, 3 nodes were configured and the Hadoop framework form Apache was used to preform task related to Map Reduce which is a very used technique in the data mining in general. This technique allows the measurement of the performance of the distributed system as well as to test its scalability. Trough this document key information about the configuration and installation of Hadoop for Raspberry pi, emphasizing the resource and memory management. These resources are limited and therefore the key to the successful management of the system. It was concluded that it is indeed possible to handle Hadoop and MapReduce in the Raspberry pi, but to make it viable to use the Raspberry pi to handle big Data there should be added nodes, letting it narrow the gap against the performance of bigger systems without having to make big money expenses. The tests were realized with TeraSort and TestDFSIO.

Key words: Raspberry pi 3, Hadoop, MapReduce, SBS, TeraSort, TestDFSIO.

TABLA DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

Con el gran incremento de flujo de datos y la importancia de Big Data, se ha abierto las puertas hacia un nuevo enfoque de entendimiento y toma de decisiones basada en datos. Este avance en la tecnología es utilizada para describir enormes cantidades de datos (estructurados, no estructurados y semi estructurados) que tomaría demasiado tiempo y sería muy costoso cargarlos a una base de datos relacional para su análisis (Barranco, 2012). El uso de herramientas tradicionales no permite el procesamiento de toda esta información de una forma correcta, por eso, el uso de herramientas como Hadoop, el cual es un framework que permite el procesamiento distribuido de grandes cantidades de datos a través de clústers de computadoras mediante el uso de una técnica llamada MapReduce (Hadoop, 2017), permite facilitar este trabajo en una gran escala.

Para poder usar esta herramienta y sacar su máximo potencial se necesitan tener varias computadoras o servidores en los cuales se pueda distribuir el procesamiento de los datos deseados. Para grandes empresas, construir grandes data centers no es un problema, pero si se trata de empresas pequeñas o proyectos pequeños personales construir data centers o tener varias computadoras no siempre es una opción por su alto costo, con un promedio de $200 a $500 por computador. Por este motivo, he desarrollado la idea de preparar y armar un clúster de Raspberry pi 3, el cual es un computador pequeño de placa simple o SBC (por sus siglas en inglés) con una arquitectura ARM y de bajo costo, de $35 por unidad. Estas características lo vuelven un candidato perfecto para elaborar un clúster en el cual manejar Big Data en situaciones en las que grandes presupuestos no están disponibles.

Parte de este trabajo es poder elaborar correctamente un clúster de SBC y medir la eficiencia que este demuestra, por medio de benchmarks, para poder determinar su

aplicabilidad en un entorno real y si realmente pueden remplazar a computadores de gran escala.

1.1 Funcionamiento de Hadoop

Para entender el funcionamiento de Hadoop se lo estudia en dos partes principales: el almacenamiento de datos llamado HDFS y el componente procesamiento de datos, siendo este específicamente la parte del método de MapReduce como tal.

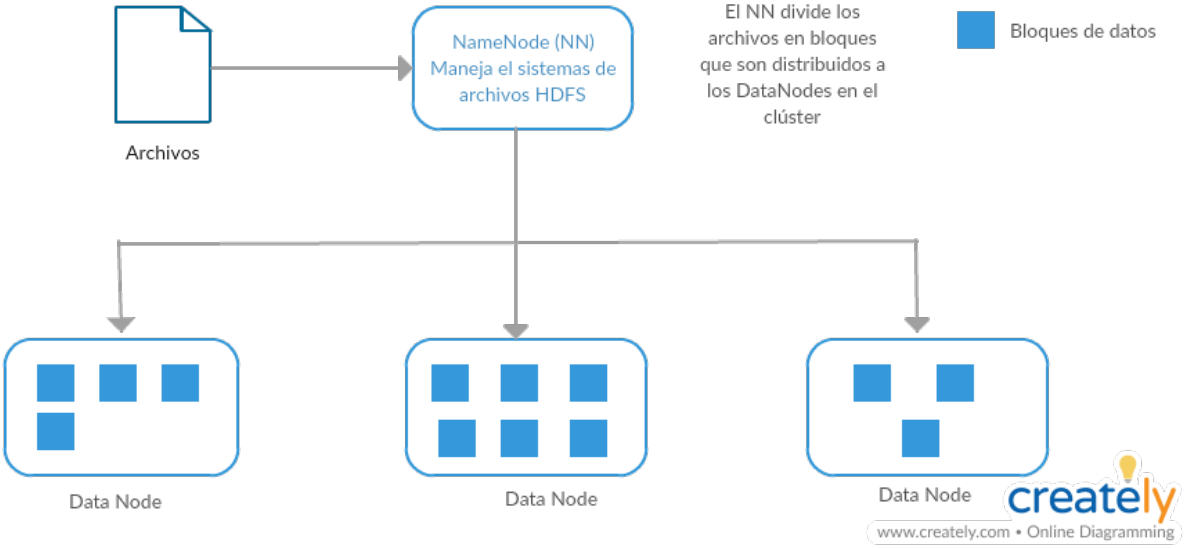
1.1.1 HDFS- Hadoop Distributed File System

Como su nombre lo indica, es el sistema de archivos distribuidos de Hadoop. Este fue construido y diseñado para ser altamente tolerante a fallas y pueda ser implementado en hardware de bajo costo, es decir no demanda muchos recursos para su funcionamiento, diferenciándolo altamente de otros sistemas de archivos distribuidos(Hadoop, 2103). HDFS provee acceso de alto rendimiento a los datos, y es adecuado para aplicaciones que tienen grandes cantidades de datos, de aquí su uso para manejar Big Data en MapReduce.

HDFS tiene una arquitectura de maestro – esclavo, en el que consiste de un único nodo llamado NameNode, el cual hace la función de servidor maestro que maneja los sistemas de archivos y regula el acceso a los archivos por parte de los clientes. Por otra parte, hay un número indefinido de DataNodes pero normalmente es un DataNode por cada nodo del clúster físico de la estructura el cual maneja el almacenamiento propio en el que corre dicho nodo. Cuando entra un archivo, este es dividido en uno o más bloques los cuales son mapeados, distribuidos y almacenados en los distintos DataNodes. El NameNode ejecuta las operaciones del sistema de archivos y determina el mapeo de los bloques dentro de los

DataNodes donde estos se encargan de servir los pedidos de lectura y escritura que tenga el cliente (Hadoop, 2013).

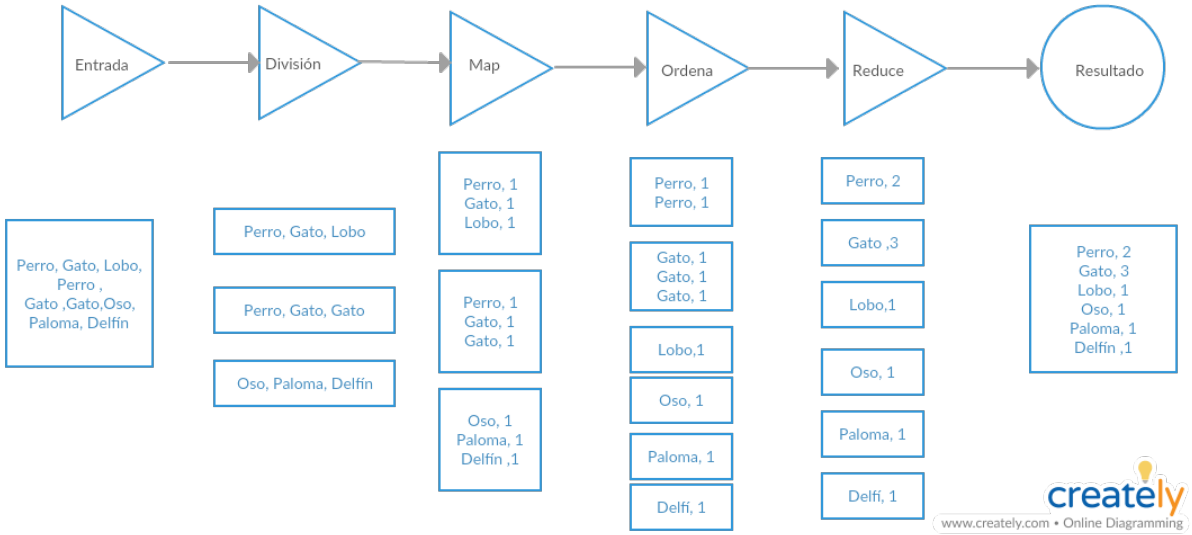
Figura 1.- Estructura de HDFS



1.1.2 MapReduce

Este es un framework escrito en Java utilizado en aplicaciones para que puedan procesar una gran cantidad de datos. Así como HDFS, este fue diseñado para ser tolerante a fallas y para trabajar en ambientes de clúster. MapReduce tiene la capacidad de dividir los datos de entrada en trabajos más pequeños, llamados los trabajos de mapeo que pueden ser ejecutados en procesos paralelos (Hadoop, 2013). Una vez que cada trabajo es mapeado estos pasan a su fase de reducción donde normalmente son almacenados en el sistema de archivos, de ahí el nombre MapReduce(Mapeo y Reducción). Un ejemplo de su funcionamiento es el sistema de conteo de palabras. Este toma un archivo de texto como entrada, lo divide en partes más pequeñas y cuenta cada una de las palabras de cada sub archivo y las ordena. Una vez que cada parte terminó su trabajo, se une el resultado de cada sub trabajo y se genera un solo archivo de salida con el conteo de palabras del archivo de entrada (IBM, 2018).

Figura 2.- Flujo MapReduce



1.1.3 YARN

YARN es un framework para Hadoop presente desde la versión 2.0, la cual tiene como objetivo de dividir y distribuir las funcionalidades del manejo de recursos y los trabajos de monitoreo y de scheduling en diferentes daemons (servicios). Este manejador de recursos tiene dos componentes principales, el Scheduler y ApplicationsManager(Hadoop, 2018).

El scheduler es responsable de distribuir los recursos a las distintas aplicaciones que se ejecutan dependiendo del límite de capacidades, colas, entre otros que este puede presentar. Pero este scheduler no realiza ninguna actividad de monitoreo o seguimiento del estado de la aplicación, tampoco ofrece ninguna garantía de reiniciar trabajos fallidos en ningún caso, por ejemplo en fallos de la aplicación o en fallo de hardware. Es decir, el scheduler solo realiza su trabajo basado únicamente en los requerimientos de recursos de la aplicación. Por este motivo se lo emparejo con el ApplicationsManager que se presenta a continuación (Hadoop, 2018).

El ApplicationsManager, por otro lado es responsable de aceptar las solicitudes de trabajos, negociar el contenedor designado a ejecutar la aplicación ApplicationMaster específica, y provee el servicio para reiniciar el contenedor del ApplicationMaster en caso de

falla. Esta aplicación paralela llamada ApplicationMaster tiene la responsabilidad de negociar los contenedores apropiados para los recursos del scheduler, seguir su status, y monitorear su progreso(Hadoop, 2018).

2.DESARROLLO DEL TEMA

2.1 Configuración de Clúster

Para este trabajo se utilizó tres Raspberry pi 3 modelo B. Este SBC es el modelo más nuevo de los Raspberry pi y es el que cuenta con mayor cantidad de recursos comparado a sus modelos predecesores. Cuenta con un procesador ARM de 4 núcleos a 1.2 GHz y 1GB de memoria RAM.

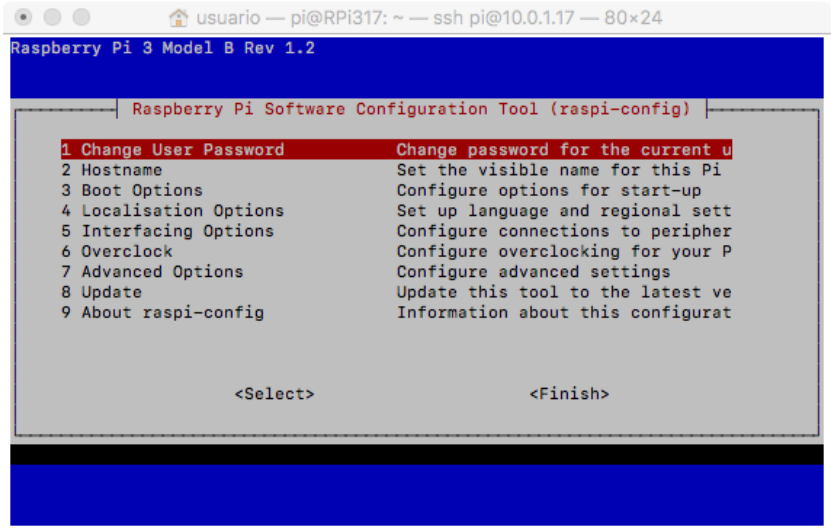
2.2 Creación del nodo maestro

Para empezar el trabajo se instaló el último sistema operativo disponible en la página oficial de Raspberry. Este es el sistema operativo Raspbian-Stretch-Lite de 64 bits, se eligió la versión lite dado que no se requiere tener una interfaz gráfica para manejar el sistema, y de esta forma hace más ligero al sistema operativo y optimiza el uso de recursos.

Se utilizó la herramienta Etcher para poder grabar el sistema operativo descargando directamente a una tarjeta micro SD de clase 10 que actúa como disco duro en este sistema, y que hace posible ser ejecutado por el Raspberry pi. Es importante que la tarjeta SD tenga como mínimo 8GB de capacidad y que sea clase 10, dado que es la tarjeta SD con mayor velocidad de escritura/lectura que existe en el mercado en el momento. En este caso se utilizaron tarjetas Kingston de 32GB de capacidad de clase 10.

Una vez instalado el sistema operativo se procede a su configuración inicial. Para esto, una vez iniciado, se ejecuta el comando *raspi-config* el cual nos llevará a una ventana de configuración gráfica como se muestra en la siguiente figura.

Figura 3 .- Ventana de configuración inicial Raspberry pi



En este panel de configuraciones, dentro de las opciones avanzadas, lo primero que se debe realizar es expandir el sistema de archivos con la opción Expand filesystem. De este modo el Raspberry pi puede utilizar la capacidad total de la tarjeta SD para sus necesidades y asegurar que no se desperdicie almacenamiento que en otras circunstancias se ignorarían por el sistema operativo. A continuación se debe ir a la opción de Memory Split, esta configuración permite determinar cuanto de la memoria RAM disponible se distribuirá para el GPU y cuanto se quedara para el sistema. Como en Hadoop no se requiere de procesamiento gráfico se elegirá 16MB destinados para GPU, el cual es el mínimo permitido por el sistema. De esta forma daremos la mayor cantidad posible de RAM para el uso de Hadoop. Dentro de estas opciones también es posible aumentar la velocidad del CPU, pero para esta configuración inicial se mantendrá con sus valores por defecto. Una vez realizadas estas configuraciones se debe reiniciar el sistema para que los cambios surjan efecto.

A continuación, se recomienda cambiar el nombre de usuario y la contraseña del usuario root. Como este es un entorno cerrado de pruebas se decidió dejar con los valores pre configurados de usuario y contraseña. Una vez dentro del usuario root de Raspberry, se procede a realizar las configuraciones de red que tendrá el clúster. Como este es el nodo

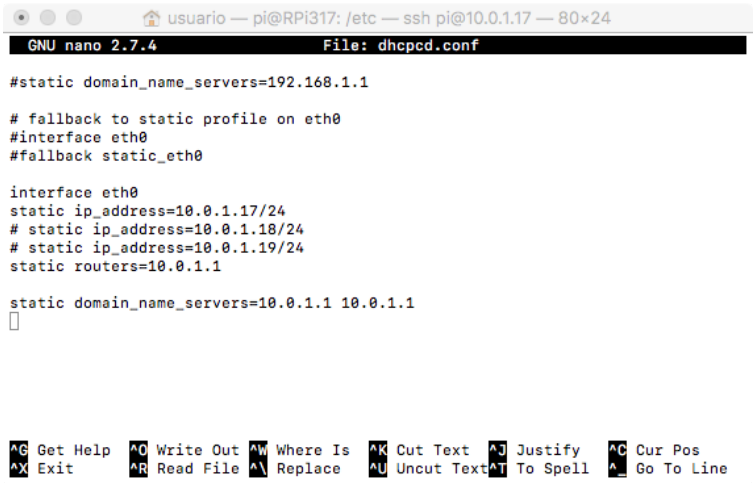
maestro debe contener la información de los demás nodos para poder conectarse a los mismos y controlarlos. Para esto nos dirigimos al directorio /etc y buscamos el archivo de configuración de red *dhcpcd.conf* y utilizamos una herramienta como nano para editar su contenido. Dentro de esta especificaremos que vamos a utilizar la interface eth0 la cual corresponde al puerto Ethernet del Raspberry pi y designamos una serie de direcciones ip estáticas que vayan de acuerdo a nuestro ambiente de red. Para nuestra configuración de clúster se utilizarán direcciones ip correspondientes al ambiente de red 10.0.1.1 y como la configuración inicial del clúster será de 3 nodos, 1 maestro y dos esclavos, se designarán las siguientes direcciones ip mostradas en la tabla 1.

Tabla 1.- Configuración de red

Dirección ip	Nodo
10.0.1.17/24	Maestro
10.0.1.18/24	Esclavo 1
10.0.1.19/24	Esclavo 2

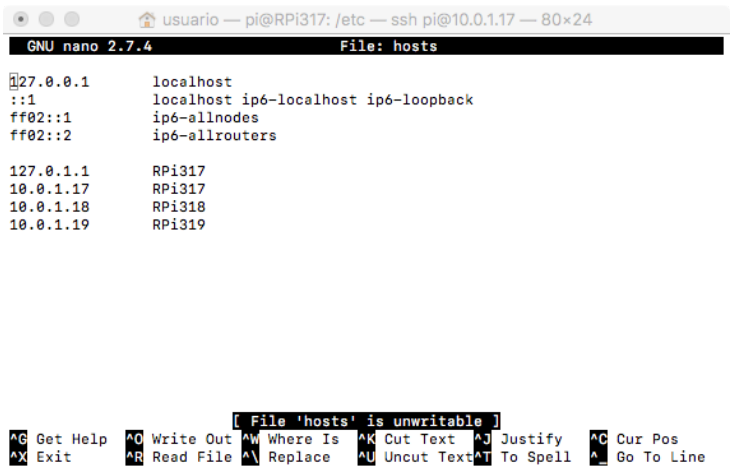
El archivo de configuración dhcpcd.conf quedará de la siguiente forma.

Figura 4.- archivo dhcpcd.conf



Posterior a tener configurada las direcciones de red, se debe crear el nombre de host para cada Raspberry pi. Esto se lo realiza igualmente dentro de el directorio /etc pero dentro del archivo *hostname.* Se utilizaron nombres representativos para poder identificar correctamente a cada nodo siguiendo el formato siguiente: RPi3xx donde xx representa los dos últimos valores de la dirección ip asignada a cada nodo. De esta forma nuestro nodo maestro se llamará RPi317 y de la misma forma para los subsiguientes nodos. Por último, para no depender de un servidor DNS, se modifica el archivo *hosts* donde se especificará que dirección ip corresponde a que hostname como se muestra en la siguiente figura:

Figura 5.- Archivo de configuración de hosts



A continuación se instalaran dependencias que requiere el sistema y Hadoop para su funcionamiento. En primer lugar, Hadoop es un programa escrito en JAVA por lo tanto debemos instalar el mismo en el sistema. Utilizaremos la versión 8 del mismo la cual es la última versión estable disponible. También se debe instalar una utilidad llamada execstack utility. Esta es importante dado que hay un bug reportado en el cual al momento de utilizar Hadoop no reconoce correctamente las librerías nativas asociadas a java y no funciona correctamente. Con esta utilidad nos aseguramos que al momento de ejecutar Hadoop todo funcione correctamente.

Por último, se debe configurar un usuario de Hadoop, este usuario es el que controlará y ejecutará todo el sistema de Hadoop y debe ser distinto al usuario root por defecto.

A nuestro usuario Hadoop le llamaremos hduser el cual debe pertenecer al grupo sudo y al grupo Hadoop. Una vez creado este usuario se debe crear el par de llaves ssh con una contraseña en blanco. Esto permitirá que los nodos se comuniquen entre ellos dentro del clúster.

2.3 Instalación de Hadoop

Para este punto hay dos opciones, o compilar nativamente Hadoop o instalar uno de los binarios pre compilados de Hadoop. Se decidió utilizar la versión de Hadoop 2.7 dado que tiene extensa documentación y reportes e bugs los cuales resultan muy útiles dado que Raspberry pi no es un sistema para el que fue diseñado específicamente Hadoop, por lo tanto se necesita tener acceso a toda la información de uso y solución de errores posibles para garantizar su funcionamiento. Por lo tanto, se procede a descargar la versión mencionada de la página oficial de Hadoop. A continuación se lo debe extraer y dar permisos al usuario hduser que creamos anteriormente para que tenga acceso a todo el directorio de Hadoop. Como se menciono hay un bug reportado para el cual se necesita execstack, para corregir este error se debe ejecutar el siguiente comando:

sudo execstack -c /opt/hadoop/lib/native/libhadoop.so.1.0.0

Una vez ejecutado ese comando la advertencia seguirá saliendo pero no afectará a al funcionamiento de Hadoop. Con esto podemos proceder a configurar las variables de entorno de Hadoop añadiendo los siguientes comandos dentro del archivo *bash.bashrc* que se encuentra en el directorio /etc.

```
export JAVA_HOME=$(readlink -f /usr/bin/java | sed "s:jre/bin/java::")
export HADOOP_INSTALL=/opt/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL
export HADOOP_HOME=$HADOOP_INSTALL
```

Por último, se debe específicar la hubicación home de java en las variables de hadoop. Esto se lo realiza en el archivo *hadoop-env.sh* que se encuentra dentro de los directorios de instalación de hadoop en la sección con el mismo nombre, es decir, *export JAVA_HOME=$.* Con esto Hadoop ya está instalado en el sistema y se puede probar su funcioameinto entrando en el usuario hduser y correr el comando *hadoop version.* Si la instalación se realizó correctamente debe salir en mensaje indicando la información de la versión instalada. Si aparece que no se reconoce el comando es porque Hadoop no se instaló.

2.4 Configuración de recursos y memoria para MapReduce y YARN

Esta configuración es necesaria para especificar el funcionamiento de Hadoop en nuestro sistema. Como este es un sistema reducido, atípico del uso estándar en el que se ejecuta Hadoop, las configuraciones de memoria y recursos deben ser correctamente establecidas de lo contrario Hadoop no funcionará en absoluto. en específico los archivos que se deben modificar son los de: *core-site.xml, hdfs-site.xml, mapred-site.xml* y *yarn-site.xml.* Estos últimos en especial son los que manejan los recursos y como se maneja el clúster. Dentro de los anexos se puede encontrar el archivo completo que se creo para las configuraciones de Hadoop.

2.4.1 MapReduce

La configuración de los recursos y de memoria para MapReduce se los realiza dentro de los archivos fuente de Hadoop que son de formato xml, específicamente en el archivo mapred-site.xml. A continuación se expondrá una tabla especificando los cambios que se realizaron y comparándolo con su valor por defecto.

Tabla 2.- Cambios en Valores de Map Reduce

Propiedad	Valor por defecto	Nuevo valor	Descripción
*.framwork.name	local	Yarn	El framework específico para manejar los trabajos de MapReduce. Puede ser local, clásico o Yarn.
*.map.memory.mb	1024	256	Cantidad de memoria a pedir del Scheduler para cada trabajo del mapeo
*.map.java.opts	-Xmx1024M	-Xmx204M	Máximo tamaño de heap de JVM para cada trabajo, debe ser menor que el punto anterior con recomendación de una proporción de 0.8
*.map.cpu.vcores	1	2	Número de CPU disponibles para trabajos de mapeo
*.reduce.memory.mb	1024	102	Memoria máxima para trabajos de reducción
*.reduce.java.opts	-Xmx2560M	-Xmx102M	Máximo tamaño de heap para los trabajos de reducción, debe ser menor al punto anterior con recomendación de

			una proporción de 0.8
*.reduce.cpu.vcores	1	2	Número de CPU disponibles para trabajos de reducción
yarn.app.mapreduce.am.resource.mb	1536	128	Memoria máxima para AppMAster
*.mapreduce.am.command-opts	-Xmx1024m	-Xmx102m	Máximo tamaño de heap para AppMAster
*mapreduce.am.resource.cpu-vcores	1	1	Máximo número de CPU para AppMAster
mapreduce.job.maps	2	4	Número de trabajos de mapeo por trabajo
mapreduce.job.reduces	1	1	Número de trabajos de reducción por trabajo

(Hadoop, s/f)

2.4.2 YARN

La configuración de los recursos y de memoria para YARN se los realiza dentro de los archivos fuente de Hadoop que son de formato xml, específicamente en el archivo yarn-site.xml. A continuación se expondrá una tabla especificando los cambios que se realizaron y comparándolo con su valor por defecto.

Tabla 3.- Cambios en Valores de YARN

Propiedad	Valor por defecto	Nuevo valor	Descripción
yarn.nodemanager.resource.memory-mb	8192	768	Cantidad de memoria física en MB que puede ser distribuida por los contenedores
yarn.scheduler.minimum-allocation-mb	1024	64	Mínimo de memoria que YARN distribuirá para un contenedor. Una aplicación debe pedir por lo menos esta cantidad de memoria o mas.
yarn.scheduler.maximum-allocation-mb	8192	256	Maxima memoria que YARN distribuirá para un contenedor
yarn.nodemanager.vmem-check-enabled	true	true	Determina si se establece un límite de memoria virtual
Yarn.nodemanager.vmem-pmem-ratio	2.1	2.1	Relación de memoria virtual a física cuando se aplica limite para los contenedores
yarn.nodemanager.pmem-check-enabled	true	true	Determina si se establece un límite de memoria física

(Hadoop, s/f)

Estas configuraciones de memoria son esenciales para el correcto funcionamiento de Hadoop en la configuración de Raspberry pi. Como se mencionó este sistema cuenta con un límite de 1024MB de memoria RAM física y se debe configurar el sistema de acuerdo a esta limitación. En el caso de que no se establezca correctamente el uso de memoria al ejecutar se eliminará al contenedor y no se podrá ejecutar MapReduce.

Tabla 4.- Distribución de memoria

RAM total (1024mb)			
Yarn nodemanager reduce (768mb)			Sistema(256mb)
Map(2x256)	Reduce(1x128)	Master(128)	

La idea de esta configuración es usar la mayor cantidad de RAM y capacidad del CPU posible del Raspberry pi. En los anexos se puede encontrar el xml completo con los cambios que se realizaron para YARN y MapReduce.

Por último, se debe crear el sistema de archivos de Hadoop, es decir el HDFS. Para esto se debe crear un nuevo directorio en el que trabajará HDFS y se debe dar permisos de acceso a hduser. Una vez creado este directorio se ejecuta el siguiente comando que formateará el sistema de archivos y lo prepara para su utilización:

hdfs namenode –format

Con esto Hadoop esta listo para usarse, solo se necesita ejecutar 2 comandos que darán por inicio al sistema de Hadoop. Estos son:

start-dfs.sh

start-yarn.sh

Para probar que todos los servicios de Hadoop estén activos se ejecuta como hduser el comando *jps* y debe mostrar el nombre de los 6 servicios que deben estar activos de Hadoop, los cuales son: DataNode, Jps, ResourceManager, SecondaryNameNode, NodeManager y NameNode. Si aparecen estos 6 servicios todo inicio correctamente y esta listo para usarse el nodo maestro.

2.5 Creación de los nodos esclavos.

Para crear los nodos esclavos existen dos opciones: clonar la tarjeta SD del nodo maestro o replicar los pasos antes mencionados pero con ligeros cambios. Para asegurar que no hayan errores y se pueda ir probando el funcionamiento se recomienda replicar los pasos de creación de un nodo más no simplemente clonar la tarjeta SD del nodo maestro. Los

cambios que se deben realzar al momento de crear los nuevos nodos es que en estos el hostname debe cambiar, respectivamente es decir el nodo dos se llamará RPi318 y el nodo 3 RPi319, además de configurar sus direcciones ip estáticas como correspondan. El cambio más importante al momento de crear cualquier nodo extra que se agregue, es de extraer los archivos de creación del par de llaves ssh creadas para el usuario hduser en el nodo maestro. Si se crean nuevas llaves, Hadoop en el nodo maestro no podrá ingresar automáticamente al sistema de los nodos y no podrá funcionar. Por esto, simplemente se copia las llaves creadas en el nodo maestro y se les coloca en la carpeta /.ssh de los nodos esclavos. Una vez completado todos los pasos de creación para los nodos, se puede volver a iniciar los servicios de Hadoop en el nodo maestro. Para comprobar que los nodos estén funcionando en conjunto se vuelve a ejecutar el comando *jps* en cada uno de los nodos. En el nodo maestro deben salir los mismos seis servicios mencionados anteriormente, pero en los nodos esclavos solo deben aparecer los 3 siguientes: NodeManager, DataNode y Jps. Esto demuestra que no están funcionando como sistemas independiente sino como un solo sistema en conjunto a lo que llamamos nuestro clúster.

2.6 Benchmarks y Resultados

En Primer lugar para probar el funcionamiento del clúster y de MapReduce se obtuvo unos archivos de texto de ejemplo a los cuales se les realizará un conteo de palabras (wordcount) que está dentro de las herramientas de Hadoop. Para empezar se creo un directorio llamado textfiles dentro del directorio home de hduser. Una vez creado este directorio se puede colocar aquí los archivos de texto con los que se quiere trabajar. Hadoop necesita que los archivos estén presentes dentro de su propio sistema de distribución de archivos. Para esto, se crea un directorio dentro de HDFS con el siguiente comando:

hadoop fs -mkdir /textfiles

Con esto el directorio textfiles ya estará disponible en HDFS para que Hadoop lo utilice. Ahora se procede a copiar los archivos de texto del directorio textfiles externo al nuevo directorio textfiles de HDFS. Para esto se utiliza el siguiente comando:

hadoop fs -put ~/textfiles/.txt /textfiles*

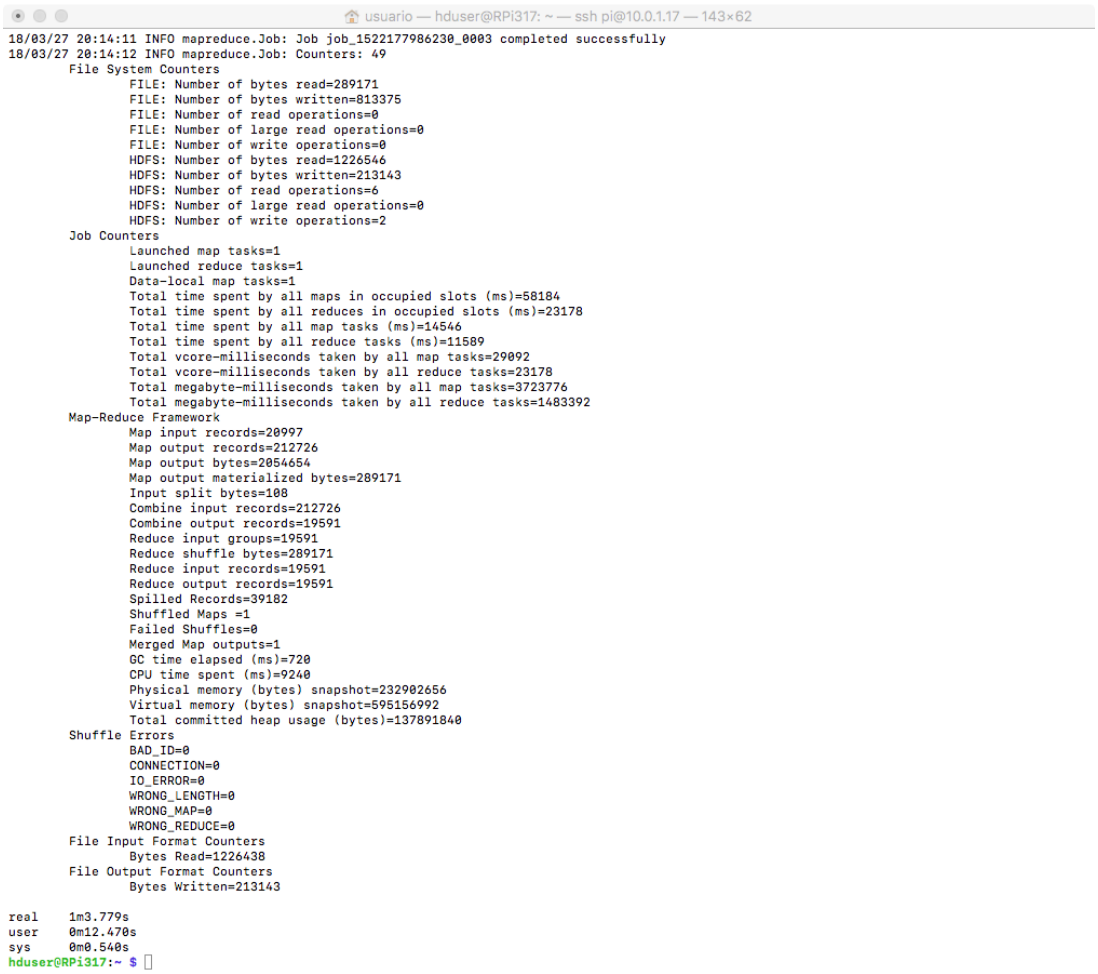
Los archivos específicos con los que Hadoop trabajará ya estarán disponibles además de tener un directorio de trabajo donde de igual manera colocará los archivos de salida cuando ejecuta en trabajo.

Como se mencionó, Hadoop ya tiene las herramientas de trabajo dentro de su sistema. La sintaxis para ejecutar sus herramientas solo se necesita especificar el paquete de herramientas, el tipo de herramienta específico del paquete y los argumentos específicos de cada herramienta. Para el caso del conteo de palabras en nuestro ejemplo se utiliza el comando de la siguiente forma para un texto pequeño y un texto mediano.

time hadoop jar /opt/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.2.jar wordcount /textfiles/smallfile.txt /smallfile-out

Como se puede observar el paquete de herramientas en este caso es Hadoop-mapreduce-examples-2.7.2.jar, la herramienta es wordcount y se especificó que el archivo con el que se trabajará es smallfile.txt y el archivo de salida con los resultados será smallfile-out. Con nuestro sistema configurado como está dio la salida mostrada en la figura 6. El archivo elegido es un libro de aproximadamente 5MB titulado The Project Gutenberg Etext of The Memoirs of General the Baron de Marbot. Este fue obtenido de la biblioteca virtual de la organización Gutemberg.(Project Gutemberg, s/f).

Figura 6.- Resultado del archivo pequeño



Como se puede observar en la figura 6 el tiempo de ejecución de este trabajo fue de 1 minuto y 3 segundos. Este tiempo para analizar un libro completo es relativamente corto aunque un poco trivial tomando en cuenta que se lo considera un archivo pequeño. En estos casos Hadoop suele recurrir a no distribuir el trabajo a los nodos esclavos dado que puede resultar ser perjudicial para el desempeño general del sistema el tener que mandar la información a los nodos y volver a retomar la información en lugar de procesarlo el mismo nodo.

Para el segundo ejemplo se tomo un extracto de 35 obras de Shakespeare con un tamaño aproximado de 40MB volviéndolo más significativo para su análisis de igual forma

de la biblioteca virtual de la organización Gutemberg (Project gutemberg, s/f). Al correr la misma herramienta sobre este texto se obtuvo el siguiente resultado:

Figura 7.- Proceso de MapReduce

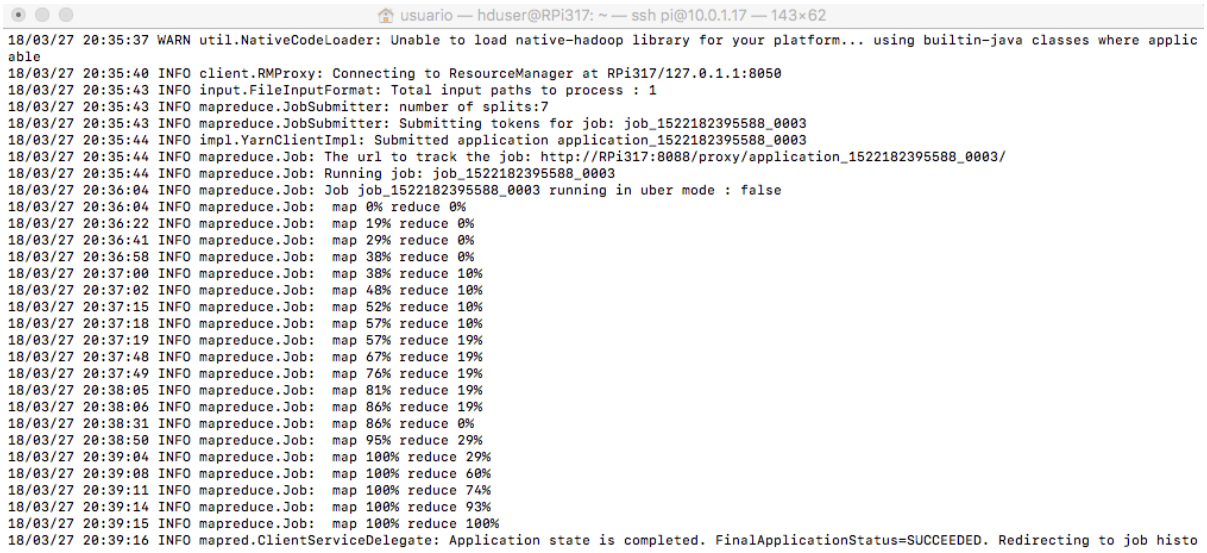
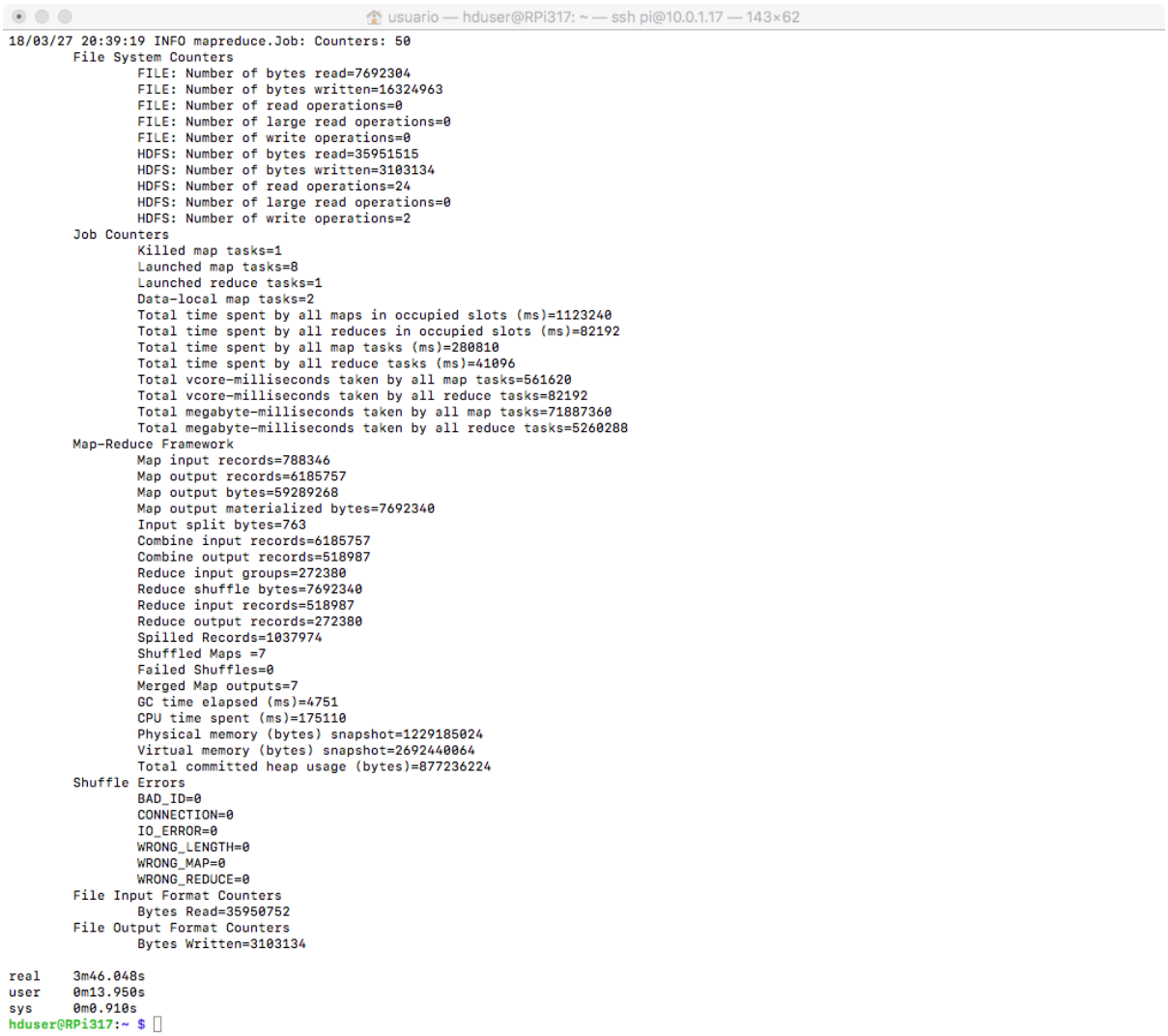


Figura 8.- Resultado del archivo mediano

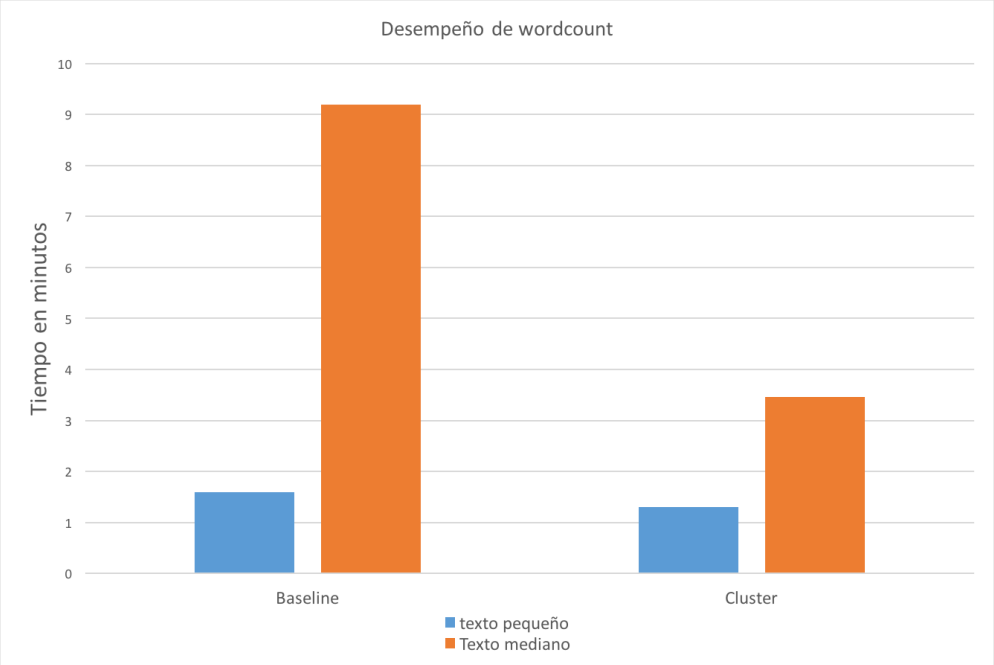


Como se puede observar en la figura 7, en este caso como el archivo es de mayor tamaño el proceso de mapeo y reducción se lo realiza en paralelo dando funcionalidad al uso de un clúster en comparación contra un solo ordenador. La tarea de mapeo y reducción se lo va realizando de acuerdo van acabando los trabajos. No es necesario que se termine de realizar el mapeo en todos los nodos para que se empiece a realizar la tarea de reducción optimizando el trabajo del mismo. Gracias a esto, como se puede observar en la figura 8, le tomo al sistema 3 minutos y 46 segundos en completar todo el trabajo. Es decir le tomo un total de 2 minutos y 43 segundos más en comparación de un trabajo pequeño como en el

anterior ejemplo. El tamaño del archivo es 8 veces mayor al anterior, sin embargo no le tomó 8 veces más en completar el trabajo.

Para tener una mejor visualización de los resultados se comparó a estos contra un línea base basado generada en un sistema de un solo computador. En la siguiente figura se presentará de forma visual la comparación de desempeño de ambos.

Figura 9.- Desempeño wordcount



En la figura 9 se observa que en el sistema base para el texto pequeño casi no hay diferencia por la trivialidad de hacer un clúster para trabajos tan pequeños, sin embargo para un texto más grande o en nuestro caso en el texto mediano se se puede notar una diferencia. En el sistema base hacer el conteo de palabras mediano tomo 9 minutos y 19 segundos en completar mientras que en el clúster de Raspberry pi tomo 3 minutos y 46 segundos.

Estos dos ejemplos realizados son para casos a pequeña escala. Cuando se trata de analizar mayor cantidad de datos es cuando se entra al mundo de lo que es Big Data como tal. Para estudiar el desempeño en casos del tipo de Big Data se utilizará dos herramientas de benchmark oficiales de Hadoop las cuales son TeraSort y TestDFSIO. La primera prueba, es decir TeraSort, lo que hace es generar una cantidad específica de datos y luego ordenarlas

mediante MapReduce y validar su desempeño. En el caso de TestDFSIO, nos ayudará a medir el desempeño para lecturas y escrituras para el sistema HDFS. Sirve para hacer pruebas de estrés al sistema y descubrir los potenciales cuellos de botella en la red que afecten el desempeño en general.(Lynch, 2017)

Al igual que la herramienta de wordocunt, TeraSort y TestDFSIO son parte del complemento de Hadoop y están listas para ser utilizadas en el entorno bajo la misma sintaxis.

Para poder utilizar TeraSort, en primer lugar se debe o tener grandes archivos de texto de ejemplo o utilizar la herramienta conjunta llamada TeraGen con la cual podemos generar cualquier cantidad de datos para su posterior análisis. En este caso utilizamos la herramienta TeraGen para generar un archivo con 1GB de datos para su análisis y poder medir el desempeño del clúster. Esto se logra con el siguiente comando:

hadoop jar /opt/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.2.jar teragen 10000000 /benchmarks/terasort-input

TeraGen funciona haciendo correr un trabajo de mapeo para generar datos pero no correrá ningún trabajo de reducción. El número de trabajos de mapeo por defecto esta definido por el parámetro especificado en el xml de nuestro sistema Hadoop, siendo en este caso 4. El único propósito es el de generar 1GB de datos aleatorios siguiente el siguiente formato: 10 bytes de llave | 2 bytes de salto | 32 bytes ASCII/hex |4 bytes de salto | 48 bytes de relleno | 4 bytes de salto (Lynch, 2017)

Una vez se termine de generar el 1GB de datos se procede a correr la herramienta de benchmark TeraSort con el siguiente comando:

hadoop jar /opt/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.2.jar *terasort /benchmarks/terasort-input /benchmarks/terasort-output*

Esta herramienta creara una serie de trabajos de mapeo para ordenar los datos ASCII generados en TeraGen. Posteriormente habrá un trabajo de mapeo por cada bloque de datos de HDFS. El número de trabajos de reducción esta ligado al parámetro especificado en el xml de nuestro sistema Hadoop, siendo en este caso 1.(Lynch, 2017)

Al momento de correr TeraSort, no pudo completar el trabajo por falta de memoria física terminando así los contenedores que ejecutan los trabajos. Este demuestra ser un gran problema al momento de tener un clúster de bajo costo. De acuerdo a Tröger, de la universidad de Chemnitz, dentro de un clúster solo una pequeña parte de la memoria física es realmente compartida entre todos los nodos.(Tröger, 2015). Dado que en nuestra distribución solo contamos con 3 nodos de prueba es normal que al tratar de analizar 1GB de datos el sistema se quede sin memoria física. Esto puede ser solucionado con la adición de más nodos al sistema y dado el bajo costo de los Raspberry pi esto no es un gran problema.

En el caso de TestDFSIO, para realizar las pruebas se ejecuto dos trabajos uno de escritura para generar los datos y otro de lectura y en ambos se obtiene resultados en unidades de Mb/s para la desempeño entrada y salida y el throughput general. Para empezar con el trabajo de escritura se ejecuta el siguiente comando:

hadoop jar /opt/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-client-jobclient-2.7.2-tests.jar TestDFSIO -write -nrFiles 2 -fileSize 10MB

Como se observa en los argumentos del comando, se debe especificar que es el trabajo de escritura, el numero de archivos a generar y el tamaño de cada archivo. Al igual que en TeraGen por el tamaño del clúster de pruebas la memoria RAM se agota rápidamente pero TestDFSIO es más flexible que TeraGen al manipular los datos con los que se trabaja y permite tener una idea más general del desempeño del sistema más que el desempeño

exclusivo de Hadoop. Por este motivo se eligió la distribución de 2 archivos de 10MB cada uno. Este test dio como resultado lo expuesto en la siguiente figura:

Figura 10.- TestDFSIO escritura

```
18/03/28 22:18:30 INFO fs.TestDFSIO: ----- TestDFSIO ----- : write
18/03/28 22:18:30 INFO fs.TestDFSIO:            Date & time: Wed Mar 28 22:18:30
 UTC 2018
18/03/28 22:18:30 INFO fs.TestDFSIO:        Number of files: 2
18/03/28 22:18:30 INFO fs.TestDFSIO: Total MBytes processed: 20.0
18/03/28 22:18:30 INFO fs.TestDFSIO:      Throughput mb/sec: 3.5106196243637
18/03/28 22:18:30 INFO fs.TestDFSIO: Average IO rate mb/sec: 3.516913414001465
18/03/28 22:18:30 INFO fs.TestDFSIO:  IO rate std deviation: 0.14877134218959367
18/03/28 22:18:30 INFO fs.TestDFSIO:     Test exec time sec: 55.962
18/03/28 22:18:30 INFO fs.TestDFSIO:
hduser@RPi317:~ $
```

Como se puede observar en la figura 10 el sistema provee de un throughput de 3.51 mega bytes por segundo y una velocidad de entrada y salida similar de 3.51 mega bytes por segundo.

Posterior a esta prueba se debe realizar el trabajo de lectura utilizando el mismo comando para la lectura con la diferencia que se debe especificar que este será el trabajo de lectura en sus argumentos.

hadoop jar /opt/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-client-jobclient-2.7.2-tests.jar TestDFSIO -read -nrFiles 2 -fileSize 10MB

Figura 11.- TestDFSIO lectura

```
18/03/28 22:21:33 INFO fs.TestDFSIO: ----- TestDFSIO ----- : read
18/03/28 22:21:33 INFO fs.TestDFSIO:            Date & time: Wed Mar 28 22:21:33
 UTC 2018
18/03/28 22:21:33 INFO fs.TestDFSIO:        Number of files: 2
18/03/28 22:21:33 INFO fs.TestDFSIO: Total MBytes processed: 20.0
18/03/28 22:21:33 INFO fs.TestDFSIO:      Throughput mb/sec: 10.576414595452142
18/03/28 22:21:33 INFO fs.TestDFSIO: Average IO rate mb/sec: 10.788280487060547
18/03/28 22:21:33 INFO fs.TestDFSIO:  IO rate std deviation: 1.511841579050321
18/03/28 22:21:33 INFO fs.TestDFSIO:     Test exec time sec: 52.815
18/03/28 22:21:33 INFO fs.TestDFSIO:
hduser@RPi317:~ $
```

En la figura 11, podemos observar que el sistema se desempeña mucho mejor en un entorno de lectura con un throughput de 10.57 mega bytes por segundo y velocidad de entrada y salida de 10.78 mega bytes por segundo. Sin embargo, como en un sistema no se depende solo de la lectura la velocidad general estaría predeterminada por el trabajo de

escritura que seria el cuello de botella. Por otra parte, este desempeño no es constante, depende del tamaño del archivo y la cantidad de archivos que se ejecuten al momento de probar. En el trabajo de Lynch, se demuestra que trabajar con 64 archivos de 16GB cada uno da un desempeño 7 veces mejor que trabajar con 4 archivos de 256GB cada uno a pesar de ser la misma cantidad de datos.(Lynch, 2017). Este cambio se da porque siempre hay un solo reductor que se ejecuta cuando todos los trabajos de mapeo se han completado. Como se sabe, el reductor es el responsable de generar el set de resultado uniendo el resultado de cada trabajo de mapeo. La indexación de datos es más lenta en un archivo de mayor tamaño que en varios archivos de menor tamaño. Por otro lado, en el caso de 64 archivos habrán 16 trabajos de mapeo por cada nodo frente a 1 trabajo de mapeo en el caso de 4 archivos y cada trabajo de mapeo termina más rápido permitiendo al trabajo reductor tener un mejor desempeño.(Lynch, 2017)

Según la HCC mientras más nodos tenga Hadoop mejor desempeño de entrada y salida y tiempo de procesamiento(HCC, 2016). En específico menciona que pasar de 3 nodos a 5 nodos aumenta el desempeño en un 36% .(HCC, 2016) Es decir, a nuestro sistema de pruebas se le puede aumentar su desempeño si se le agrega mayor cantidad de nodos.

Hadoop no se limita a trabajos de conteo de palabras o minería de datos como tal. Otra función que se le puede dar a un clúster de Hadoop es el análisis numérico. En este caso en específico se ejecutará un trabajo de multiplicación de matrices mediante métodos de mapeo y reducción. Cabe notar que esta herramienta no es propia del sistema de herramientas de Hadoop, pero dado su gran facilidad de inclusión de código nuevo escrito en JAVA, se han creado estos complementos, por la comunidad de Hadoop para expandir sus funcionalidades.

Supongamos que se tiene una matriz M con elementos de fila *i* y columna *j* denotados por *mij* y una matriz N con elementos de fila *j* y columna *k* denotados por *njk*. Por lo tanto el producto P = M*N será una nueva matriz con elementos de fila *r* y columna *k* denotados por *Prk* donde *P(r,k) = mij*Njk.*

Para que una matriz pueda ser procesada por MapReduce, debe seguir el siguiente estándar. Se representa a la matriz M como M(I,J,V) con tuplas(*i, j, mij*) y la matriz N como N(J,K,kW), con tuplas (*j,k,njk*). En la vida real, la mayoría de matrices son dispersas, por lo tanto gran cantidad de celdas tienen valor cero. Para optimizar el almacenamiento y la ejecución, cuando se representan matrices en esta forma, no se necesita mantener entradas para estas celdas de valor cero.

Una vez que se tengan los datos de entrada siguiendo el formato el cual el Hadoop reconocerá se procede a realizar su multiplicación con MapReduce.

Trabajo de Mapeo en Multiplicación de Matrices

el trabajo de mapeo se encarga de generar un par de datos que contiene una llave y un valor correspondiente a la llave para cada tupla de cada matriz. Una vez que se genera el par de datos de cada matriz, se combina este par de datos en una sola, donde las entradas con el mismo valor de llave para cada matriz son agrupadas devolviendo un nuevo par de datos del tipo <llave, valor> (Aytekin, 2016).

Trabajo de Reducción En multiplicación de Matrices

El trabajo de reducción, toma el par de datos del tipo <llave, valor> como entrada y procesa una llave a la vez y lo divide en dos listas, una para M y otra para N que posteriormente serán ordenadas. Una vez que se ordenan las listas para que el formato de la matriz coincida, se suma el producto de las dos listas siguiendo el índice ordenado dando así el resultado de la multiplicación de la matriz (Aytekin, 2016).

Como esta no es una herramienta estándar de la biblioteca de Hadoop, existen distintas versiones de códigos para realizar esta operación por lo que su eficiencia puede variar dependiendo de la optimización que se haya realizado en el código que se utilice. Para nuestro ejemplo se utilizará la herramienta de multiplicación de matrices creado por Aytekin. Como entrada se utilizará una matriz M de tamaño 1000 x 100 y una matriz N de 100 x 1000 con una dispersión de 0.3. Esto significa que cada matriz tendrá por lo menos 30000 datos. Al correr esta multiplicación de matriz se obtuvo resultado en aproximadamente 1 minuto 27 segundos. El cual es un resultado aceptable para nuestro sistema

Según Eyas, Ayyoub y Abu. En general, el uso de sistemas distribuidos para la multiplicación de matrices reduce el tiempo de ejecución de $O(n^3)$ a $O(n^2)$ representando un gran incremento en el rendimiento de estas operaciones. Pero que el beneficio de agregar más nodos se va reduciendo hasta un punto en el que agregar más nodos no representa ninguna mejora en tiempos de ejecución(Eyas, Ayyoub, & Abu, 2004).

3. CONCLUSIONES

Este trabajo fue realizado para mostrar la capacidad de tener un sistema distribuido de bajo costo con las SBC Raspberry pi. Basado en las pruebas realizadas, un sistema de pocos nodos como el utilizado para realizar este, no puede manejar grandes cantidades de datos volviéndolo inviable. Este problema de tener pocos nodos fue en su mayor parte por la cantidad limitada de memoria RAM que disponen estos sistemas, solo 1GB, y por su incapacidad de expansión de la misma por métodos tradicionales. Debido a esto los contenedores de Hadoop se quedan sin memoria rápidamente deteniendo el trabajo que estén realizando ya sea de mapeo o reducción. En específico, en esta configuración con archivos únicos de más de 100MB el clúster no podía trabajar. Según la página oficial de Raspberry pi se podría utilizar una memoria USB externa como swap para tener un incremento de memoria virtual. Sin embargo el desempeño de esta no es la misma que tener más RAM dentro de la palca madre de los Raspberry pi y su configuración puede resultar difícil y presentar muchos errores. (Raspberrypi, 2015)

La configuración ideal para un sistema Hadoop esta compuesto por servidores con las siguientes características:

- 2 CPUs quad /octo cores
- 64 GB de RAM (O'Dell, 2013)

Servidores con estas características cuestan cada uno entre $1500 y $7000 dólares por cada sistema, dependiendo de otras configuraciones entre ellas el almacenamiento y otros.(DELL, 2018) Lo cual para proyectos relativamente pequeños o para propósitos educativos sería una inversión demasiado grande e inaccesible. Por este motivo, incluso si se escalara el tamaño de este trabajo a 20 nodos representaría un costo total de aproximadamente

$800 y un gran incremento de su desempeño permitiendo trabajar con mayor cantidad de datos aproximándose más a situaciones reales sin la necesidad de hacer inversiones tan exorbitantes. Los SBC nunca podrán tener el mismo desempeño que los servidores de gama alta, pero como se menciono anteriormente para propósitos educativos o proyectos pequeños es una gran alternativa con un gran potencial, además de que cada vez aparecen más opciones de SBC con mejores características y a precios moderados.

A pesar de los relativamente bajos recursos que tienen los Raspberry pi , las aplicaciones que se les puede dar a estos son casi infinitas. Con el incremento del internet de las cosas (IoT) el uso de SBC se vuelve más práctico y da paso al crecimiento de los mismos. El tener todo un sistema funcional del tamaño de una tarjeta de crédito abre las puertas a muchas más aplicaciones y servicios de los que contamos hoy en día. Según la Universidad de Southampton, a pesar de el tipo de arquitectura que utiliza, que para los estándares de hoy en día no es convencional, muchos aspectos de su diseño como el uso de hardware y software Opensource, los procesadores de bajo consumo de poder y las grandes aplicaciones para el almacenamiento flash hará que usar estos sistemas se vuelvan una tendencia en el futuro. Algunas universidades importantes ya están realizando proyectos con cientos de módulos de Raspberry y prueban que su capacidad de escalabilidad es casi ilimitada.

Por otro lado, Hadoop, demostró ser una gran herramienta por su amplia utilidad . Desde las tareas de conteo de palabras para minería de datos hasta aplicaciones matemáticas como el cálculo de pi , mediante integración numérica o la multiplicación de matrices. Dado que Hadoop es construido sobre JAVA y es Opensource, da la facilidad de crear nuevas aplicaciones para el mismo que sigan la filosofía de MapReduce o que se beneficien del uso de trabajos en paralelo en distintos nodos. Todo esto sin la necesidad de pagar por licencias o necesitar de software avanzado para su modificación y mejoramiento continuo.

Entre las dificultades encontradas al realizar este proyecto fue la compilación de Hadoop. El sistema operativo utilizado es una distro simplificada de Linux por lo que para hacerlo funcionar correctamente se debió instalar una gran cantidad de complementos para hacer posible la compilación de Hadoop y sin embargo se presentaban ciertos errores. Si bien compilar el código fuente directamente en el Raspberry pi le adapta al sistema operativo, instalar el binario pre compilado es la mejor opción. Estas versiones ya contienen las últimas correcciones de errores parches y herramientas disponibles además de la simplificación de su actualización. A parte de este, en general manejar un sistema distribuido de Raspberry pi es bastante sencillo y fácil de escalar.

4. REFERENCIAS BIBLIOGRÁFICAS

Aytekin, M. (2016, Febrero 16). Matrix Multiplication with MapReduce. Recuperado en Abril 22, 2018, de https://lendap.wordpress.com/2015/02/16/matrix-multiplication-with-mapreduce/

Barranco, R. (2012, Junio 18). ¿Qué es Big Data? Recuperado en Agosto 17, 2017, de https://www.ibm.com/developerworks/ssa/local/im/que-es-big-data/index.html

DELL. (2018). Servidores Dedicados Dell. Recuperado en Marzo 28, 2018, de http://www.dell.com/ec/empresas/p/servers

Eyas, E., Ayyoub, A., & Abu, N. (2004). Quick Matrix Multiplication on Clusters of Workstations. Vilna: Institute of Mathematics and Informatics. Recuperado en Abril 24, 2018, de https://www.mii.lt/informatica/pdf/INFO549.pdf

Hadoop. (2018, Marzo 16). Apache Hadoop YARN. Recuperado en Marzo 25, 2018, de https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

Hadoop. (2013, Abril 08). HDFS Architecture Guide. Recuperado en Febrero 5, 2018, de https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

Hadoop. (s.f.). Mapred-Default. Recuperado en Marzo 20, 2018, de https://hadoop.apache.org/docs/r2.7.2/hadoop-mapreduce-client/hadoop-mapreduce-client-core/mapred-default.xml

Hadoop. (2013, Abril 08). MapReduce Tutorial. Recuperado en Febrero 5, 2018, de https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

Hadoop. (2017). What is Apache Hadoop? Recuperado en Agosto 17, 2017, de http://hadoop.apache.org/

Hadoop. (s.f.). Yarn-Default. Recuperado en Marzo 20, 2018, de https://hadoop.apache.org/docs/r2.7.2/hadoop-yarn/hadoop-yarn-common/yarn-default.xml

HCC. (2016, Agosto 25). More Hadoop nodes = faster IO and processing time? Recuperado en Marzo 25, 2018, de https://community.hortonworks.com/articles/53264/more-hadoop-nodes-faster-io-and-processing-time.html

IBM. (2018). Apache MapReduce. Recuperado en Febrero 5, 2018, de https://www.ibm.com/analytics/hadoop/mapreduce

Lynch, D. (2017, Noviembre 20). Running TeraSort MapReduce Benchmark. Recuperado en Marzo 20, 2018, dehttps://discuss.pivotal.io/hc/en-us/articles/200927666-Running-TeraSort-MapReduce-Benchmark

MAPR. (2014). TeraSort Benchmark Comparison for YARN. Recuperado en Septiembre 15, 2017, de https://mapr.com/resources/terasort-benchmark-comparison-yarn/

O'Dell. (2013, Agosto 28). How-to: Select the Right Hardware for Your New Hadoop Cluster. Recuperado en Marzo 28, 2018, de https://blog.cloudera.com/blog/2013/08/how-to-select-the-right-hardware-for-your-new-hadoop-cluster/

Project Gutemberg. (s.f.). Project Gutenberg's Etext of Shakespeare's First Folio/35 Plays. Recuperado en Febrero 20, 2018, de http://www.gutenberg.org/ebooks/2270

Project Gutemberg. (s.f.). The Memoirs of General Baron de Marbot. Recuperado en Febrero 20, 2018, de http://www.gutenberg.org/ebooks/2401

Raspberrypi. (2017). Raspberry pi - about us. Recuperado en Agosto 17, 2017, de https://www.raspberrypi.org/about/

Raqspberrypi.(2015).USB RAM. Recuperado en Marzo 28, 2017, de https://www.raspberrypi.org/forums/viewtopic.php?t=97898

Tröger, P. (2015). To what extent is RAM shared in large parallel computers? Recuperado en Marzo 23, 2018, de https://www.researchgate.net/post/To_what_extent_is_RAM_shared_in_large_parallel_computers

5. ANEXO A: ARCHIVOS XML

Yarn-site.xml

```
<configuration>
    <property>
        <name>yarn.resourcemanager.resource-tracker.address</name>
        <value>RPI317:8025</value>
    </property>
    <property>
        <name>yarn.resourcemanager.scheduler.address</name>
        <value>RPi317:8035</value>
    </property>
    <property>
        <name>yarn.resourcemanager.address</name>
        <value>RPI317:8050</value>
    </property>
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
    <property>
        <name>yarn.nodemanager.resource.cpu-vcores</name>
        <value>4</value>
    </property>
    <property>
        <name>yarn.nodemanager.resource.memory-mb</name>
        <value>768</value>
    </property>
    <property>
        <name>yarn.scheduler.minimum-allocation-mb</name>
        <value>64</value>
    </property>
    <property>
        <name>yarn.scheduler.maximum-allocation-mb</name>
        <value>256</value>
    </property>
    <property>
        <name>yarn.scheduler.minimum-allocation-vcores</name>
        <value>1</value>
    </property>
    <property>
        <name>yarn.scheduler.maximum-allocation-vcores</name>
        <value>4</value>
    </property>
    <property>
        <name>yarn.nodemanager.vmem-check-enabled</name>
        <value>true</value>
    </property>
    <property>
        <name>yarn.nodemanager.pmem-check-enabled</name>
        <value>true</value>
    </property>
    <property>
        <name>yarn.nodemanager.vmem-pmem-ratio</name>
        <value>2.1</value>
    </property>
</configuration>
```

Mapred-site.xml

```
<configuration>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
    <property>
        <name>mapreduce.map.memory.mb</name>
        <value>256</value>
    </property>
    <property>
        <name>mapreduce.map.java.opts</name>
        <value>-Xmx204m</value>
    </property>
    <property>
        <name>mapreduce.map.cpu.vcores</name>
        <value>2</value>
    </property>
    <property>
        <name>mapreduce.reduce.memory.mb</name>
        <value>128</value>
    </property>
    <property>
        <name>mapreduce.reduce.java.opts</name>
        <value>-Xmx102m</value>
    </property>
    <property>
        <name>mapreduce.reduce.cpu.vcores</name>
        <value>2</value>
    </property>
    <property>
        <name>yarn.app.mapreduce.am.resource.mb</name>
        <value>128</value>
    </property>
    <property>
        <name>yarn.app.mapreduce.am.command-opts</name>
        <value>-Xmx102m</value>
    </property>
    <property>
        <name>yarn.app.mapreduce.am.resource.cpu-vcores</name>
        <value>1</value>
    </property>
    <property>
        <name>mapreduce.job.maps</name>
        <value>4</value>
    </property>
    <property>
        <name>mapreduce.job.reduces</name>
        <value>1</value>
    </property>
</configuration>
```

Core-site.xml

```
<configuration>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/hdfs/tmp</value>
  </property>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://RPi317:54310</value>
  </property>
</configuration>
```

Hdfs-site.xml

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <name>dfs.blocksize</name>
    <value>5242880</value>
  </property>
</configuration>
```