Trabajo de Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Clúster Hadoop sobre Rapberry Pi con datos de acelerómetro, giroscopio y sensor de temperatura

Autor: Hernán Adrián Gómez González
Tutor: Antonio Jesús Sierra Collado

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo de Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Clúster Hadoop sobre Rapberry Pi con datos de acelerómetro, giroscopio y sensor de temperatura

Autor:
Hernán Adrián Gómez González

Tutor:
Antonio Jesús Sierra Collado

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Trabajo Fin de Grado: Clúster Hadoop sobre Rapberry Pi con datos de acelerómetro, giroscopio y sensor de temperatura

Autor: Hernán Adrián Gómez González

Tutor: Antonio Jesús Sierra Collado

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

A mi familia

A mis amigos

Agradecimientos

A mis padres, por todo lo que han hecho por mí. Gracias por haberme apoyado en mis decisiones, por confiar en mí y por darme todo lo que he necesitado. Durante todos estos años como estudiante siempre habéis estado a mi lado, animándome a seguir adelante y gracias a vosotros, soy quien soy hoy en día. Nunca encontraré forma de agradecéroslo.

A mi hermano, por los consejos y cosas que siempre me has enseñado. Gracias a ti he descubierto incontables cosas que me han permitido ver el mundo desde otra perspectiva, además de haberme aconsejado siempre que lo he necesitado.

A mis amigos de siempre, por soportarme durante todos estos años que no han sido nada fáciles. Siempre habéis estado ahí, escuchándome y apoyándome, a pesar de los dramas y del estrés que hemos vivido. No podría tener amigos mejores.

A los amigos y personas que conocido durante la etapa universitaria. Gracias por todos los momentos increíbles y por la gran cantidad de conocimiento y valores que me habéis aportado. Sin vosotros nada de esto habría sido posible.

Hernán Adrián Gómez González

Sevilla, 2020

Resumen

Vivimos en un mundo conectado donde la cantidad de información que generamos es mayor día a día. Toda esta cantidad de datos contiene información muy valiosa que antes estaba oculta, pero ahora, podemos encontrarla y usarla en el progreso de la humanidad. La herramamienta que nos permite almacenar, procesar y analizar estos datos es Big Data. La relevancia de esta tecnología es cada vez mayor, y sus posibilidades son casi infinitas. Lo podemos encontrar en las redes sociales, internet, investigaciones científicas, empresas y otros cientos de lugares. Big Data está en todas partes, y sus ventajas son considerables.

Una de las tecnologías más usadas en el mundo de Big Data es Apache Hadoop. Hadoop nos permitirá realizar las tareas de Big Data de forma muy cómoda, ofreciendo un amplio ecosistema. Por una parte, con Hadoop podremos almacenar los datos de un manera totalmente fiable y tolerante a fallos. Por otra parte, Hadoop nos permite procesar todos esos datos de una forma sencilla. Cuando hablamos de procesar datos, nos referimos a que nos permite procesar terabytes de información, por ello es una herramienta muy potente. Para permitir todo esto, Hadoop posee tres componentes principales que son HDFS, MapReduce y Yarn.

Por tanto, además de explicar qué es Big Data y ver algunas de sus aplicaciones, en este trabajo nos centraremos en Hadoop. Veremos en detalle cada uno de sus componentes y cómo funcionan. Además de explicar cómo funciona teóricamente, también realizaremos una demostración práctica. Para ello instalaremos Hadoop en un clúster de Raspberry Pi. Este clúster estará compuesto por cuatro Raspberry Pi 3 model B+ que permitirán ejecutar Hadoop sin problema. Detallaremos los componentes necesarios, cuál ha sido el proceso de instalación, así como las herramientas que nos ofrece.

Hadoop posee un amplio ecosistema en el que cada una de sus diversas aplicaciones ofrece una solución para los diferentes problemas de Big Data. Una de esas herramientas es Apache Pig. Pig nos permite realizar las tareas MapReduce de una forma sencilla y rápida. MapReduce es el framework usado por Hadoop para el procesamiento de datos, por lo que su importancia es vital. De esta forma, también instalaremos Pig en nuestro clúster Hadoop para realizar las tareas MapReduce. Veremos su funcionamiento y profundizaremos en el lenguaje que utiliza Pig para el procesamiento de los datos, el cual es Pig Latin. Estudiaremos todas las operaciones más importantes de Pig Latin y realizaremos un ejemplo con cada una de estas operaciones usando el clúster Hadoop.

No podremos usar Hadoop ni Pig sin unos datos que poder almacenar y procesar. Para ello se usará un sensor MPU6050. Este sensor ofrece un acelerómetro, giroscopio y sensor de temperatura. Los datos capturados por este sensor serán almacenados en un fichero de texto el cual podremos almacenar en Hadoop y procesar con Pig.

Por tanto, en este trabajo, demostraremos que es posible instalar Hadoop en uno de los ordenadores más baratos y sencillos que existen, como son las Raspberry Pi. Almacenaremos los datos obtenidos durante un período de tiempo por un sensor MPU6050 usando HDFS, el cual es el sistema de almacenamiento de Hadoop. Y, por último, procesaremos estos datos usando Pig, con lo que veremos las operaciones más importantes que posee y su funcionamiento.

Abstract

We live in a connected world where the information that we generate is greater day by day. This amount of data contains a very valuable information that was hidden, but now, we can find it and use it in the progress of humanity. The Big Data is the tool that allows us store, process and analyze this data. The relevance of this technology is increasing, and its possibilities are almost endless. It can be found in social networks, the internet, scientific researches, companies and hundreds of other places. Big Data is everywhere, and its advantages are considerable.

One of the most used technologies in the Big Data world is Apache Hadoop. Hadoop will allow us to carry out Big Data tasks in a very confortable way, offering a wide ecosystem. On the one hand, Hadoop will allow us storing data in a totally reliable and fault-tolerant way. On the other hand, Hadoop allows us to process all this data in a simple way. When we talk about processing data, we mean that it allows us to process terabytes of information, this is why it is a very powerful tool. Hadoop has three main components to manage all of this, those are three components which are HDFS, MapReduce and Yarn.

Therefore, in addition to explaining what Big Data is and studying some of its applications, we will focus on Hadoop in this work. We will see in detail each of its components and how they work. In addition to explaining how it works theoretically, we will also do a practical demonstration. We will install Hadoop on a Rasperry Pi cluster for this. This cluster will be composed of four Raspberry Pi 3 model B+ that will allow Hadoop to run without any problem. We will detail the necessary components, the installation process, as well as the tools it offers us.

Hadoop has a wide ecosystem in which each of its components offers a solution for different Big Data problems. One of those tools is Apache Pig. Pig allows us to perform MapReduce tasks in a simple and fast way. MapReduce is the framework used by Hadoop for data processing, so its importance is vital. In this way, we will also install Pig on our Hadoop cluster to perform MapReduce tasks. We will see how it works and we will study the language that Pig uses for data processing, which is Pig Latin. We will study all the most important Pig Latin operations and we will run an example of each of these operations using the Hadoop cluster.

We will not be able to use Hadoop or Pig without any data to store and process. A MPU6050 sensor will be used for this. This sensor offers and accelerometer, gyroscope and a temperature sensor. The data captured by this sensor will be stored in a text file which we can store in Hadoop and process with Pig.

Therefore, in this work, we will demonstrate that it is posible to install Hadoop on one of the cheapest and simplest computers that exist, such as the Raspberry Pi. We will sotre the data obtained over a period of time by the MPU6050 sensor using HDFS, which is the Hadoop storage system. And finally, we will process this data using Pig, studying the most importat operations it has and how they work.

ÍNDICE

Agradecimientos ix

Resumen xi

Abstract xiii

ÍNDICE xv

Índice de Ilustraciones xvii

Índice de Tablas xxi

Notación xxiii

1 Objetivos 26

2 Introducción 29

2.1 Introducción a Big Data 29

2.1.1 Datos estructurados y no estructurados 31

2.1.2 Las V's de Big Data 31

2.1.3 Algunos casos de uso 32

2.1.4 Problemas sociales 33

3 Hadoop 34

3.1 ¿Qué es Hadoop? 34

3.1.1 HDFS 35

3.1.2 MapReduce 39

3.1.3 Yarn 41

3.1.4 Versiones de Hadoop 43

3.1.5 Ecosistema Hadoop 44

3.1.6 Distribuciones Hadoop 48

4 Descripción de la solución 51

4.1 Equipo necesario 51

4.1.1 Ordenador portátil MSI GE60 2PC Apache 51

4.1.2 Raspberry Pi 3 Model B+ 51

4.1.3 Switch TP-Link TL-SG108 V3.0 52

4.1.4 Tarjetas microSDHC SanDisk Ultra 32GB 52

4.1.5 Sensor MPU6050 52

4.2 Recursos software 54

4.2.1 NOOBS 54

4.2.2 SD Card Formatter 54

4.2.3 Hadoop 55

4.2.4 Pig 55

4.3 Diseño de la solución 55

5 Desarrollo, implantación y prueba de la solución 59

5.1 Toma y almacenamiento de datos 59

5.2 Hadoop e interfaz gráfica *60*
5.3 Pig y procesamiento de datos *62*
5.2.1 Funcionamiento de los operadores de Pig Latin 62

6 Conclusiones y líneas futuras **81**
6.1 Conclusiones *81*
6.2 Líneas futuras *82*

Referencias **84**

Anexos **88**

ÍNDICE DE ILUSTRACIONES

Ilustración 1: Esquema básico del escenario planteado 28
Ilustración 2: Evolución coste por almacenaje de un GB de información desde 1980 hasta 2015[1] 30
Ilustración 3: Evolución del tamaño de los transistores desde 1971 hasta la actualidad [3] 30
Ilustración 4: Logo de Hadoop 34
Ilustración 5: Representación gráfica de la estructura de HDFS [20] 36
Ilustración 6: Proceso de "checkpointing" en HDFS [20] 38
Ilustración 7: Ejemplo de MapReduce 40
Ilustración 8: Representación gráfica de la estructura de YARN [22] 41
Ilustración 9: Flujo de ejecución de una tarea en YARN [23] 42
Ilustración 10: Diferencias entre la arquitectura de Hadoop 1.x y Hadoop 2.x 43
Ilustración 11: Representación gráfica del método "Erasure coding" [26] 44
Ilustración 12: Ecosistema Hadoop [20] 45
Ilustración 13: Ecosistema Hadoop Cloudera [37] 49
Ilustración 14: Switch TP-Link TL-SG108 V3.0 [39] 52
Ilustración 15: Ejes en el MPU6050 [40] 52
Ilustración 16: Representación gráfica de un sistema MEMS [41] 53
Ilustración 17: Movimiento angular 54
Ilustración 18: Raspberry Pi en soporte y numeración dada [45] 56
Ilustración 19: Pines GPIO en la Raspberry Pi 3 model B+ [47] 56
Ilustración 20: Escenario completo 57
Ilustración 21: Esquema de conexiones completo del escenario planteado 58
Ilustración 22: Ejemplo de datos del MPU6050 59
Ilustración 23: Página principal de la interfaz web de Hadoop 61
Ilustración 24: Pestaña "Datanodes" de la interfaz gráfica de Hadoop 61
Ilustración 25: Réplica de "datosMPU.txt" 62
Ilustración 26: Resultado de la operación DUMP 63
Ilustración 27: Resultado de la operación FILTER 64
Ilustración 28: Resultado de la operación DISTINCT 65
Ilustración 29: Resultado de la operación DUMP 66
Ilustración 30: Fallo de la operación ASSERT 67
Ilustración 31: Resultado de la operación LIMIT 67

Ilustración 32: Resultado de la operación CROSS 68
Ilustración 33: Primer resultado de la operación DUMP 68
Ilustración 34: Segundo resultado de la operación DUMP 69
Ilustración 35: Resultado de la operación GROUP 69
Ilustración 36: Resultado de la operación FOREACH con bloque 70
Ilustración 37: Resultado de la operación FOREACH anidado 71
Ilustración 38: Resultado de la operación (inner) 72
Ilustración 39: Resultado de la operación JOIN (outer) 72
Ilustración 40: Resultado de la operación CUBE 73
Ilustración 41: Resultado de la operación CUBE con ROLLUP 74
Ilustración 42: Resultado de la operación ORDER BY 74
Ilustración 43: Resultado de la operación RANK sin DENSE 75
Ilustración 44: Resultado de la operación RANK con DENSE 75
Ilustración 45: Resultado de la operación UNION 76
Ilustración 46: Resultado de la ejecución de la macro "macroFiltro" 77
Ilustración 47: Macro "temperaturaMedia" 78
Ilustración 48: Resultado de la ejecución de la macro "temperaturaMedia" 78
Ilustración 49: Resultado de la operación STREAM 79
Ilustración 50: SD Card Formatter 88
Ilustración 51: Pantalla inicial de NOOBS 89
Ilustración 52: Menú de configuración de Rasperry Pi OS 89
Ilustración 53: Selección país, idioma y zona horaria en Rasperry Pi OS 90
Ilustración 54: Establecimiento de contraseña para usuario “pi” en Raspbian OS 90
Ilustración 55: Selección red wifi en Raspbery Pi OS 91
Ilustración 56: Selección bordes negros en Raspberry Pi OS 91
Ilustración 57: Actualización Raspberry Pi OS 92
Ilustración 58: Modificación en el fichero “/boot/cmdline.txt” 92
Ilustración 59: Configuración IP estática en Rasperry Pi 1 93
Ilustración 60: Configuración IP estática en Windows 10 93
Ilustración 61: Conexión SSH desde el ordenador portátil a Raspberry Pi 2 94
Ilustración 62: Configuración del fichero “/etc/hosts/” en la Raspberry Pi “master” 94
Ilustración 63: Creación del usuario “hduser” 95
Ilustración 64: Configuración del fichero “~/.ssh/config” 95
Ilustración 65: Definición de las variables de entorno de Java y Hadoop en el fichero “~/.bashrc” 96
Ilustración 66: Selección de Java 8 por defecto 96
Ilustración 67: Configuración del fichero “/opt/hadoop/etc/hadoop/slaves” 97
Ilustración 68: Configuración del fichero “core-site.xml” de Hadoop 97
Ilustración 69:Configuración del fichero “hdfs-site.xml” de Hadoop 98
Ilustración 70: Configuración del fichero “yarn-site.xml” de Hadoop 99

Ilustración 71: Configuración del fichero “mapred-site.xml” de Hadoop 99
Ilustración 72: Configuración en el fichero “hadoop-env.sh” 99
Ilustración 73: Ejecución del comando “/opt/hadoop/bin/hdfs dfsadmin -report” 101
Ilustración 74: Definición variables de entorno de Pig en el fichero “~/.bashrc” 102
Ilustración 75: Ejecución del comando “pig -version” 102
Ilustración 76: Selección de “Interfacing Options” 103
Ilustración 77: Selección de “I2C” 103
Ilustración 78: Selección de “Yes” para habilitar I2C 104
Ilustración 79: Script "mpu.py" [59] 105

ÍNDICE DE TABLAS

Tabla 1: Ejemplo de datos estructurados 31
Tabla 2: Operadores Pig Latin [34] 47

Notación

IoT	Internet of Things, Internet de las cosas
HDFS	Hadoop Distributed File System
YARN	Yet Another Resource Negotiator
POSIX	Portable Operating System Interface for X
IP	Internet Protocol
TCP	Transmission Control Protocol
HTTP	Hypertext Transfer Protocol
GB	Gigabyte
KB	Kilobyte
MB	Megabyte
TB	Terabyte
UDF	User Defined Function
MHz	Megahercio
GHz	Gigahercio
CPU	Central Processing Unit
RAM	Random Access Memory
SDA	Serial Data
SCL	Serial Clock
GND	Ground
VCC	Voltaje en corriente directa
dV	Derivada de la velocidad
dt	Derivada del tiempo
dΘ	Derivada del desplazamiento angular
bps	Baudios por Segundo
QoS	Quality of Service, Calidad del servicio.
MEMS	MicroElectroMechanical Systems
DMP	Digital Motion Processor
V	Voltio
A	Amperio
GPIO	General Purpose Input/Outpout
ºC	Grados centígrados

1 OBJETIVOS

La tecnología avanza cada vez más rápido. Este avance nos ha traído una cantidad de opciones enorme para poder realizar cualquier tipo de tarea, tanto para las personas individuales como empresas, gobiernos, entre otros. El uso que hacemos hoy en día de todas estas herramientas ha provocado que entremos en una nueva era, y esta era es la era de los datos. Todo lo que hacemos en internet, la información que generan sensores, así como otros muchos casos, generan una cantidad descomunal de datos que hasta hace relativamente poco tiempo no podíamos procesar. Aquí es donde entra Big Data.

Big Data abarca desde el almacenamiento de grandes cantidades de datos al procesamiento y análisis de estos. Gracias a esta tecnología, podemos sacar nuevas conclusiones y hacer nuevos descubrimientos que antes estaban ocultos entre todos estos datos que, en principio, no podían aportarnos nada. El término Big Data realmente no es algo nuevo, ya que el análisis de datos ha sido constante en nuestra historia. Pero ha sido en los últimos años, en los que los avances en procesamiento, internet o la disminución de costes ha permitido demostrar el verdadero potencial.

Desde siempre me ha gustado la tecnología y descubrir los avances que se producen en ella. No fue hasta hace poco tiempo cuando descubrí en qué consistía Big Data y lo que realmente nos puede aportar. Es realmente increíble la cantidad de cosas que pueden descubrirse gracias a las técnicas de Big Data, ayudadas de técnicas de Machine Learning o Inteligencia Artificial. Es por este motivo que he decidido realizar este proyecto, y poder transmitir lo realmente útil que nos puede ser esta tecnología.

Como vimos en el resumen, en este proyecto, usaremos uno de los componentes más importantes que existen en el mundo de Big Data. Este componente es Apache Hadoop. Hadoop nos ofrece tres componentes principales, que son HDFS, MapReduce y Yarn. Gracias a estos tres componentes, podremos almacenar y procesar TB de información sin problema. Por tanto, explicaremos en detalle Hadoop y sus componentes. Pero no nos quedaremos en la parte teórica, también realizaremos una demostración práctica. Para ello, instalaremos Hadoop en un clúster de Raspberry Pi 3 Model B+. Una de las mayores ventajas que nos ofrece Hadoop, es la posibilidad de usarlo y realizar el almacenamiento y procesamiento de datos en hardware básico, es decir, el mismo hardware que usan los ordenadores de nuestras casas. Las Raspberry Pi son ordenadores muy económicos y que tienen un hardware no muy potente, pero suficiente para poder ejecutar Hadoop.

Como mencionamos, Hadoop posee un amplio ecosistema para ofrecer distintas soluciones ante los distintos problemas que surgen con el Big Data. De este ecosistema, hemos decidido usar apache Pig, que posee un lenguaje que nos permite realizar las tareas MapReduce de una forma mucho más sencilla y cómoda. Estudiaremos las operaciones más relevantes de Pig Latin ofreciendo un ejemplo para cada una de ellas, usando nuestro clúster Hadoop. Los datos que usaremos en este caso, serán los datos obtenidos por un sensor MPU6050. Este sensor posee un acelerómetro, un giroscopio y un sensor de temperatura. Los datos capturados por este sensor, pueden ser almacenados en un fichero de texto para su posterior almacenamiento en Hadoop y procesamiento con Pig.

Ahora que sabemos que vamos a realizar en este trabajo, vamos a detallar los objetivos del mismo en la siguiente lista:

- 1.- Introducción al mundo de Big Data y explicación de sus posibilidades: Para iniciar el trabajo, primeramente, nos adentraremos en el mundo de Big Data. Realizaremos una definición del mismo, estudiaremos por qué es tan importante y expondremos algunos casos de uso reales en los que se ha

aplicado esta tecnología.

- 2.- Explicación de Hadoop, sus componentes y su ecosistema: Una vez que tengamos claro qué es el concepto de Big Data, estudiaremos Hadoop en profundidad. Haremos una introducción al mismo, y veremos en detalle cada uno de sus tres componentes. Además, estudiaremos su ecosistema resumiendo las posibilidades que nos ofrece.
- 3.- Montaje del clúster de Raspberry Pi 3 Model B+: Para poder instalar Hadoop en el clúster de Rapsberry Pi, primero tendremos que montarlo y configurarlo. Para ello, tendremos que explicar cada uno de los componentes que se han usado en el clúster, como pueden ser las Raspberry Pi 3 Model B+, un conmutador, cables ethernet, adaptadores de corriente, entre otros. Además, tendremos que instalar el sistema operativo en cada una de estas Raspberry Pi, así como relizar la configuración necesaria para el correcto funcionamiento del clúster.
- 4.- Instalación de Hadoop en el clúster de Raspberry Pi: Una vez tengamos montado el escenario del clúster de Raspberry Pi, tendremos que instalar Hadoop. Como se mencionó, Hadoop permite ser usado en hardware básico, de esta forma, instalándolo en estas Raspberry Pi, podremos demostrar que es posible. Por tanto, explicaremos en detalle el proceso y configuraciones necesarias para que Hadoop pueda funcionar correctamente.
- 5.- Instalación de Pig en el clúster Hadoop: El siguiente paso, será instalar Pig en el clúster Hadoop previamente montado y configurado. Tendremos que explicar los pasos y configuraciones necesarias para poder realizar la instalación.
- 6.- Conexión del MPU6050 a la Raspberry Pi y toma de datos del mismo: Un paso necesario será realizar la conexión del sensor MPU6050 a una de las Raspberry Pi para su posterior toma de datos. Para ello podremos usar los pines GPIO que poseen las Raspberry Pi, así como una "protoboard" para realizar las conexiones. Una vez correctamente conectado, veremos cómo podemos tomar los datos obtenidos por el sensor. Para esta tarea, usaremos un script escrito en Python que nos permitirá almacenar estos datos en un fichero de texto.
- 7.- Vistazo de las herramientas que nos ofrece Hadoop en el clúster. Cuando el escenario completo esté finalizado, podremos ejecutar Hadoop en nuestro clúster y comprobar su funcionamiento. Hadoop nos ofrece algunas herramientas muy útiles, y por ello, veremos algunas de las disponibles en nuestro clúster.
- 8.- Ejemplo de cada una de las operaciones más importantes de Pig Latin: Para demostrar el funcionamiento del clúster, realizaremos un ejemplo de cada una de las operaciones más importantes de Pig Latin usando los datos obtenidos por el sensor MPU6050. De esta forma, quedará demostrado que es posible realizar un análisis Big Data en un hardware tan básico como puede ser el de las Raspberry Pi.

Para que tengamos una idea clara de cómo va a ser el escenario que vamos a realizar en este trabajo, tenemos la Ilustración 1. En esta Ilustración, podemos ver un esquema muy básico de cómo será el escenario. Principalmente, tendremos las cuatro Raspberry Pi interconectadas entre sí gracias al uso de un conmutador. La conexión del sensor, solo será necesaria realizarla en una de las Raspberry Pi, por tanto, a través de una "protoboard", podremos conectar el sensor. Hadoop estára instalado en las cuatro Raspberry Pi, mientras que Pig solo será necesario instalarlo en una de ellas.

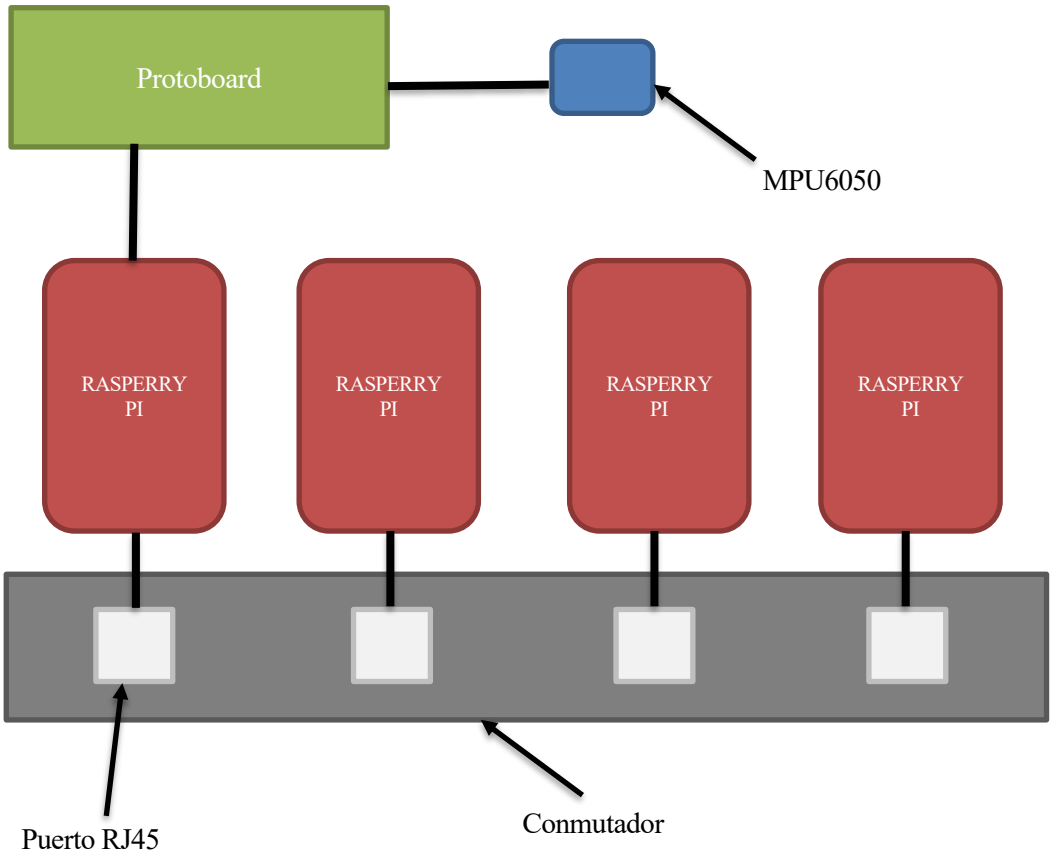


Ilustración 1: Esquema básico del escenario planteado

2 INTRODUCCIÓN

En este capítulo nos adentraremos en el mundo de Big Data. Estudiaremos sus numerosas posibilidades y aplicaciones, destacando sus características y funciones principales. Una vez que hayamos comprendido mejor el conecto de Big Data, será mucho más sencillo entender Hadoop y su ecosistema en los capítulos posteriores.

2.1 Introducción a Big Data

En el Capítulo 1 se hizo una breve definición de Big Data, pero una definición más acertada sería que Big Data es la capacidad de procesas grandes volúmenes de datos, de varios tipos, a muy bajo coste y alta velocidad. Esta tecnología nos permite poder analizar datos que antes nos resultaban imposibles de procesar, con el objetivo de hacer nuevos descubrimientos.

Actualmente, nos hemos convertido en seres digitales. Millones de personas se conectan a internet día a día, realizan búsquedas, tienen redes sociales, se hacen fotos y las publican, escuchan música online o cualquiera de las infinitas posibilidades que nos ofrece. Esto crea una cantidad de datos que nos resulta difícil de imaginar. Para hacernos una idea, veamos algunos datos de todo lo que sucedía en cada minuto del año 2019[1]:

- Los usuarios de Netflix veían 694.444 horas de video.
- Se realizaban 277.777 publicaciones en Instagram.
- Los usuarios de Youtube veían 4.500.000 horas de video.
- En Twitter se publicaban 511.200 tweets.
- Se relizaban 140.000 de “swipes” en Tinder.
- Se relizaban 4.497.420 búsquedas en Google.

Esto es sólo una pequeña parte de los datos de algunas de las plataformas más conocidas y en sólo un minuto. En un año la cantidad de datos generada es inmensa. A priori, toda esta información puede parecer que no tiene ningún valor, son datos desordenados que parecen no tener relación ni que puedan aportar nada relevante. Pero con las técnicas de Big Data se puede extraer muchísima información valiosa de todos estos datos, descubriendo patrones o comportamientos que desconocíamos.

Cabe destacar que no sólo las personas generan datos, también lo hacen las máquinas, los sensores de las industrias o nuestros hogares, entre otros cientos de dispositivos. Esto es debido a que en los últimos años también se ha popularizado el Internet de las Cosas o IoT (Internet of Things). IoT consiste en la conexión de múltiples dispositivos a una red, en la que todos ellos pueden interaccionar y generan información. Con IoT prácticamente cualquier dispositivo puede conectarse a internet. Se pronostica que para el año 2025 habrá unos 75 mil millones de dispositivos conectados [2]. Esto genera una cantidad de datos enorme, que gracias al Big Data podemos procesar y obtener información muy valiosa.

Tenemos que atender a varios factores para poder entender qué ha permitido que Big Data esté causando una revolución. El primero de ellos está relacionado con el coste del almacenamiento de la información. Hemos hablado de la gran cantidad de datos que se generan día a día en el mundo. Esto no es gratis, si se genera información hay que almacenarla, y este almacenamiento tiene un coste. Año tras año el precio de los

dispositivos de almacenamiento ha caído significativamente según ha ido evolucionando la tecnología. Si nos fijamos en datos, podemos ver que el precio por GB ha caído desde el millón de dólares por GB en 1980, a unos 2 centavos en el año 2015 [3]. En la Ilustración 2 podemos ver la gráfica de la evolución en los precios.

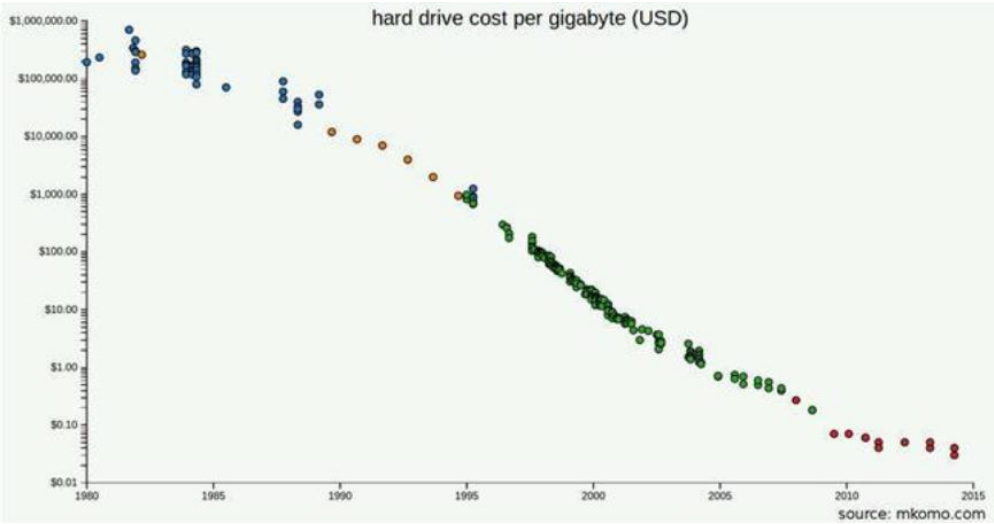


Ilustración 2: Evolución coste por almacenaje de un GB de información desde 1980 hasta 2015[1]

Además, Big Data no sólo almacena información, también la procesa. Ese procesamiento, al igual que el almacenamiento, no es gratis, también tiene un coste. Pero no sólo se ha reducido el coste del almacenamiento de información. La tecnología ha evolucionado y se ha producido una importante reducción en el coste por millón de transistores. El tamaño de estos transistores ha ido disminuyendo, por lo que en el mismo espacio se ha podido ir introduciendo mayor número de transistores y a un menor precio. Esto permite aumentar la capacidad de procesamiento y la eficiencia. En la Ilustración 3, podemos ver como el tamaño de los transistores se ha ido reduciendo a lo largo de la historia.

Ilustración 3: Evolución del tamaño de los transistores desde 1971 hasta la actualidad [3]

Por último, hay que destacar que gracias a las nuevas técnicas que se han ido desarrollando, como MapReduce, que veremos más adelante, se ha permitido poder procesar los datos no estructurados. Vamos a ver en qué

consisten los datos estructurados y no estructurados en más detalle para poder comprenderlo mejor.

2.1.1 Datos estructurados y no estructurados

La verdadera revolución de Big Data viene de la capacidad de procesamiento de datos no estructurados. Tradicionalmente, el tipo de datos que hemos podido procesar son los datos estructurados, que son los que encontramos en las bases de datos relacionales. Estos datos son archivos de texto que se almacenan en formato tabla con título para cada categoría y que así pueda ser identificada [4]. Se usan lenguajes tipo SQL para poder gestionar esta información. La ventaja de este tipo de datos es que, al estar organizados en tablas con filas, columnas y títulos, está todo perfectamente organizado e identificado, de forma que se puede acceder fácilmente y analizar estos datos. Su procedimiento de análisis es sencillo y se podía realizar con la capacidad de computación existente. Un ejemplo de este tipo de datos sería la Tabla 1.

Nombre	Edad	Sexo	Peso	Color de ojos
María	23	Femenino	54	Azul
Juan	54	Masculino	68	Marrón
Antonio	31	Masculino	80	Azul
Luisa	18	Femenino	62	Marrón

Tabla 1: Ejemplo de datos estructurados

Como podemos ver, en la tabla tenemos datos diferentes personas. Cada fila se corresponde con una persona en concreto, mientras que cada columna se corresponde con algún tipo de información sobre ella, que en este caso son datos como su nombre, edad, sexo, peso y color de ojos. Estos datos se encuentran perfectamente etiquetados y organizados. Si necesitamos por ejemplo ver el peso promedio, el color de ojos más común u otro análisis similar resultaría muy sencillo consultarlo y obtener los resultados.

Esto no ocurre con los datos no estructurados. Al contrario que los datos estructurados, los datos no estructurados no tienen una estructura interna identificable. Esa básicamente un conjunto masivo de objetos que no tienen valor hasta que son identificados y se almacenan de manera organizada en bases de datos no relacionales (NoSQL). Varios ejemplos de datos no estructurados serían los documentos PDF, los correos electrónicos, las publicaciones en redes sociales, videos o música, entre otros. [4]

Lo interesante es que este tipo de datos suponen la mayoría de información. Por ejemplo, el 80% de la información relevante para un negocio se origina de forma no estructurada [4]. Además, ya pudimos comprobar en el apartado anterior como la mayoría de datos generados en 2019 cada minuto eran datos no estructurados. De ahí viene la importancia de Big Data en este aspecto. En los datos no estructurados hay mucha información, patrones o estructuras que pueden resultarnos muy útiles.

2.1.2 Las V's de Big Data

Las características principales de Big Data se definen en las llamadas V's de Big Data. Generalmente, se habla de 3 V's principales que son las siguientes [5]:

- Volumen: Esta característica se refiere a la gran cantidad de datos que se generan y recopilan continuamente. Big Data tiene que tener la capacidad de poder procesar toda esa cantidad de información.
- Velocidad: Big Data no solo tiene que ser capaz de procesar un gran volumen de datos, también debe

ser rápido en esta tarea. Con Big Data somos capaces de poder procesar datos en tiempo real y poder obtener información prácticamente al instante.

- Variedad: Como ya hemos comentado anteriormente, todos estos datos no son los mismos. Todos los datos que se obtienen pueden ser muy distintos (distintos formatos), y tener orígenes diferentes, y esto no puede ser un problema. Big Data debe ser capaz de procesar todos estos tipos diferentes de datos que existen.

Además de estas 3 V's, últimamente se suelen incluir 2 V's más, que son las siguientes [5]:

- Veracidad: Big Data también debe ser capaz de garantizar que un dato es de calidad, y no está tomando datos que pueden ser inciertos y provocar inexactudes en los resultados.
- Valor: Big Data solo tendrá sentido si el resultado de procesar esa gran cantidad de datos resulta en el descubrimiento de patrones e información que nos permita hacer descubrimientos y tomar decisiones.

2.1.3 Algunos casos de uso

En este apartado vamos a ver algunos ejemplos de cómo Big Data ha ayudado a empresas o investigaciones a mejorar o hacer nuevos descubrimientos. Vamos a comenzar por las empresas, pero para ello primero tenemos que conocer el Business Intelligence. El Business Intelligence era la manera que anteriormente usaban las empresas para analizar los datos. Consiste en un análisis reactivo, es decir, la toma de decisiones se realizaba en función de los datos obtenidos por hechos pasados. En base a lo que ocurrió, se pueden tomar algunas decisiones. Por ejemplo, se puede saber cuantos productos se vendieron el mes pasado, o cuantos clientes se fueron, pero esto no ayuda en qué se puede hacer para que el mes siguiente esto no ocurra [6].

Con Big Data es diferente, podemos hacer un análisis predictivo. Podemos ver cuántos clientes se perdieron, que razones hubo para que eso ocurriera, y así tomar decisiones que impidan que el mes siguiente no se vayan los clientes. Esto es debido a que la cantidad de datos que disponemos con Big Data es muchísimo mayor al poder analizar la información no estructurada, y podemos establecer cuales son los motivos y así poder tomar decisiones. Incluso nos ayuda en prevenir averías, caídas de sistemas, ciberataques o similares, mientras que con el Business Intelligence no podíamos hacer nada de esto.

Por tanto, ahora que tenemos el contexto, veamos algunos casos de uso reales:

- En este ejemplo vamos a ver el caso real de una aseguradora que tenía numerosos problemas con fraudes. Es decir, esta empresa tenía problemas con las personas que eran familiares, amigos o conocidos, que se ponían de acuerdo y conseguían estafar a la aseguradora. Esto era un grave problema porque a la aseguradora le era imposible saber si esas personas eran familiares o amigos al no ser clientes. En cambio, cuando decidieron usar técnicas de Big Data, obtuvieron datos de las redes sociales de sus clientes, y con ello pudieron establecer un mapa de relaciones de las personas. El resultado fue que se detectó una cantidad de relaciones de las que se desconocía su existencia, reduciendo significativamente los casos de fraude [7].
- Otro caso en una empresa es el caso de Netflix [8]. Netflix analiza todos los datos de sus clientes de forma que no existen dos cuentas iguales. Por ejemplo crea varias portadas diferentes para una misma serie. En función de tus gustos mostrará una portada u otra de manera que aumente la probabilidad de que acabes viendo esa serie. Incluso cambia el orden de capítulos de series en función de la probabilidad de que te acabe gustando más. Esto también ocurre en las redes sociales, como Instagram, cuya pestaña "Explora" te muestra las publicaciones según tu historial anterior y así mantenerte conectado durante más tiempo.
- A la hora de la realización de este proyecto estamos bajo la pandemia del COVID-19, y Big Data también nos puede ayudar en este caso. Hay numerosos equipos científicos alrededor de todo el mundo trabajando en diferentes soluciones para poder combatir este virus. Esto genera una gran cantidad de información útil que puede servir para otros investigadores. Para ello la Comisión Europea ha creado una plataforma para que los científicos puedan compartir y obtener al instante todo tipo de información relacionada con este virus, lo que permite que las investigaciones y los descubrimientos vayan mucho más rápido [9]. También nos sirve para poder analizar si las medidas que toman los diferentes países están funcionando o no en la movilidad de las personas. Analizando los datos que se obtienen de los

teléfonos móviles y otros dispositivos se pueden analizar los patrones de movimiento de las personas, como se ha hecho en Madrid [10]. Esto permite saber si las medidas funcionan o no.

Hay miles de casos como estos, incluso podemos ayudar a proteger a nuestro planeta como lo que hace la compañía Fregata Space, que analizando imágenes satelitales evitan que más de 8 millones de toneladas de plásticos acaben en el océano cada año [11]. Como podemos ver a las empresas Big Data les está ayudando a captar más clientes, mantenerlos y así poder obtener mayores beneficios. Pero como comentamos no solo ayuda a las empresas, también a la ciencia, por lo que las posibilidades son infinitas.

2.1.4 Problemas sociales

Por ultimo, decidí incluir este apartado ya que las aplicaciones de Big Data no tienen por qué ser todas buenas. Hay usos que se están convirtiendo en un problema que cada día podemos ver que va a peor, y nos afecta tanto individualmente como sociedad.

El caso más destacable es el que habláblamos en el apartado anterior de la personalización. Nuestras redes sociales están tan personalizadas para nosotros mismos que son capaces de mantenernos atrapados durante horas. Esto está causando un problema cada vez mayor de adicción a las redes sociales. Y no sólo esto. En estas redes sociales se encuentran las "fake news". Los algoritmos creados con ayuda de Big Data y Machine Learning nos muestran estas noticias falsas que se adaptan a nuestros gustos. Esto tiene consecuencias graves, como que la sociedad cada vez se encuentre más dividida por culpa de las "fake news" relacionadas con política [12]. Incluso hay empresas como "Cambridge Analytica", que usa los datos para cambiar el comportamiento de audiencias, que han influido en el resultado de elecciones en numerosos países [13].

Además, con las técnicas de Big Data, los gobiernos de los diferentes países pueden tomar todo tipo de información de sus ciudadanos, además de control en tiempo real con cámaras y reconocimiento facial, así como otras aplicaciones. Esto es algo que causa mucha controversia y puede ser algo bueno o no según con el fin con el que estos datos sean usados.

Por último, también mencionar cómo Big Data ha hecho que nuestros datos sean tan valiosos, que las empresas los vendan por grandes sumas de dinero. Esto ocurrió con Facebook, cuando en el año 2018 se publicó que esta empresa había vendido datos sensibles de los usuarios a otras empresas [14].

Por tanto, podemos ver que Big Data es una tecnología que nos ayuda muchísimo a mejorar la rentabilidad de las empresas, realizar nuevos descubrimientos y, en definitiva, ayudar al progreso de la humanidad. Pero también tiene su parte mala, ya que es tan efectiva que también se puede usar para perjudicar a los seres huamanos y a este mismo progreso.

3 HADOOP

En este capítulo vamos a centrarnos en uno de los componentes más populares del mundo de Big Data, el cual usaremos en este trabajo, hablamos de Hadoop. Hadoop posee una serie de componentes y un ecosistema muy amplio, por lo que explicaremos el funcionamiento de todos estos componentes, así como su ecosistema, destacando a Pig, ya que éste también será utilizado en el trabajo.

3.1 ¿Qué es Hadoop?

Hadoop es una implementación "open source" de Big Data, ampliamente usada en la industria y con características que a veces son incomparables con otras implementaciones. La definición oficial de Hadoop [15] dice que es un software "open source" para computación fiable, escalable y distribuida. Básicamente es un framework para el procesamiento distribuido de grandes cantidades de datos, a través de clústers de ordenadores usando modelos simples de programación.

Hadoop surgió de un proyecto llamado "Nutch" en el año 2002, el cual iba a ser un motor de búsqueda que presentó numerosos problemas de escalabilidad [16]. En aquella época internet estaba aún extendiéndose, la cantidad de información que se iba generando era cada vez mayor y esto hacía difícil procesar, almacenar y tener disponible esa información. En el año 2004, Google presentó el modelo MapReduce y un sistema de ficheros llamado GFS, que podía solucionar estos problemas. Google publicó toda esta documentación con todos los detalles y Nutch lo implementó. A continuación, en el año 2006, Doug Cutting, que trabajaba en el proyecto Nutch, se unió a Yahoo!, y allí surgió como nuevo proyecto Hadoop. A partir de ahí surgieron numerosas versiones, cada una con diferentes características y Nutch quedó como proyecto aparte [17]. Actualmente Hadoop es mantenido por la Apache Software Fundation y es de código abierto. Como curiosidad, su logo es un elefante com el que vemos en la Ilustración 4, porque el hijo de Doug tenía un peluche de un elefante. Esto ha provocado que la mayoría de componentes de su ecosistema tengan nombres o logos de animales, como veremos más adelante.

Ilustración 4: Logo de Hadoop

Una de sus mayores ventajas es que se ejecuta en hardware básico, es decir, en hardware de ordenadores personales como los que tenemos en casa. No necesitamos un superordenador o enormes capacidades de procesamiento para poder ejecutar Hadoop. Incluso como veremos más adelante, en este trabajo implementaremos Hadoop en un clúster de Raspberry Pi, las cuales tienen un hardware básico y muy barato.

Además, tiene otra serie de ventajas, como su almacenamiento distribuido, que le permite ser muy flexible, escalable y tener tolerancia completa ante fallos. También tiene como ventaja que tiene una gran comunidad detrás al ser de código abierto, y se pueden desarrollar muchísimas aplicaciones que permitan el análisis de datos

muy complejos [17].

Los principales objetivos de Hadoop son los siguientes [18]:

- Escalabilidad: Tiene capacidad para almacenar y procesar grandes cantidades de datos, y si esta cantidad aumenta, debe ser capaz de seguir procesándola. Por tanto, puede soportar gran cantidad de equipos.
- Tolerancia a fallos: Tiene capacidad para recuperarse de errores de hardware. Hadoop asume que los ordenadores fallan queramos o no. Esto se debe a que usa hardware básico como se explicó anteriormente. Por tanto, este software es capaz de que, aunque un nodo de una implementación Hadoop falle, el sistema siga funcionando y no se produzcan pérdidas de datos.
- Tipos de datos: Tiene capacidad para gestionar datos de diferentes tipos.
- Entorno compartido: Debe ser capaz de gestionar varias tareas al mismo tiempo.
- Aportar valor: El objetivo final debe ser poder extraer valor de los datos.

Hadoop posee una arquitectura que resuelve dos de los problemas más grandes de Big Data, que son el almacenamiento distribuido y el procesamiento en paralelo. Para el almacenamiento tiene un sistema distribuido llamado HDFS, obteniendo los llamados "Hadoop clusters", para el procesamiento tiene un algoritmo que permite procesamiento paralelo y distribuido, llamado MapReduce.

Además, la arquitectura de Hadoop es de tipo maestro-esclavo. Tiene un nodo maestro y el resto serán nodos esclavos. Esta arquitectura y cómo gestiona las tareas le da una serie de ventajas que estudiaremos a continuación, pero para ello debemos explicar sus componentes, que son HDFS, Yarn y MapReduce.

Destacar que un clúster de Hadoop va a consistir en un conjunto de nodos interconectados entre sí por uno o varios conmutadores. En cada clúster va a ver el nodo máster y los esclavos. Incluso se podrían interconectar varios pequeños clústers de Hadoop para formar uno grande. Existen diferentes modos de clústers [16] [17]:

- Clúster multi-nodo (distributed): Es el tipo que hemos explicado, en el que cada instancia de Hadoop se encuentra en un nodo diferente que es un equipo diferente, todos ellos interconectados entre sí. Pueden llegar a tener cientos o miles de nodos.
- Pseudo distribuido (pseudo distributed): Existen diferentes instancias de Hadoop pero residen en una misma máquina. Esto sería el primer modo pero en un solo equipo en el que cada nodo es una máquina virtual.
- Independiente (standalone): Solo hay una instancia de Hadoop y se encuentra en la misma máquina virtual. Por lo que no hay almacenamiento distribuido (HDFS). Realmente sólo serviría para ver como funciona Hadoop y poder realizar algunas pruebas.

3.1.1 HDFS

Como se comentó anteriormente, HDFS (Hadoop Distributed File System) es una de los componentes principales que forman Hadoop. Es un sistema distribuido de almacenamiento de ficheros que, a su vez, tiene varios componentes (nodos), de los cuales dos son principales, estos son el NameNode y el DataNode. Estos componentes siguen una estructura maestro-esclavo, en la que el NameNode va a ser el maestro y los DataNodes van a ser los esclavos. HDFS se basa en lenguaje Java y se suele ejecutar sobre nodos con GNU/Linux. Los nodos se interconectan usando el protocolo TCP/IP.

HDFS tiene muchas similitudes como otros sistemas distribuidos, pero tiene unas diferencias significantes ya que es altamente tolerante a fallos y está diseñado para hardware básico. Sus principales principios son [19]:

- Fallos de hardware: como HDFS va a estar compuesto de cientos o miles de nodos con hardware básico, hay una probabilidad muy alta de que cualquiera de uno de ellos falle. Por tanto la detección rápida de estos fallos y que el sistema se recupere automáticamente de estos es uno de los objetivos principales.
- Acceso a la transmisión de datos: en HDFS no se van a ejecutar aplicaciones de propósito general, sino aplicaciones que van a procesar grandes cantidades de datos por lotes. Por ello se centra en tener un mayor rendimiento en el acceso a estos datos, en vez de tener una baja latencia para acceder a ellos. Al basarse en POSIX, se ha tenido que intercambiar su semántica en algunas zonas clave, ya que POSIX

impone numerosas restricciones que no son necesarias o pueden llegar a ser perjudiciales para las aplicaciones en HDFS.

- Grandes conjuntos de datos: HDFS está preparado para soportar cientos de miles de ficheros, los cuales pueden llegar a tener tamaños desde GB hasta TB. Por este motivo proporciona un gran ancho de banda y es capaz de escalar con cientos de nodos.
- Modelo de coherencia simple: El modelo de acceso de HDFS es un acceso único de escritura y lectura multiple. Es decir, una vez un archivo ha sido creado, escrito y cerrado no se puede volver a cambiar. Sólo se puede añadir contenido al final de los ficheros, pero no en un punto arbitrario. De esta forma el acceso a los datos tiene un gran rendimiento.
- Computación cercana a los datos: Cuando tenemos que ejecutar un cálculo que opera sobre cantidades de datos muy grandes, es mejor migrar el cálculo más cerca de donde se encuentran los datos, en vez de mover los datos al lugar donde se ejecuta la aplicación. Esto se debe a que si acercamos la aplicación al conjunto de datos, minimizamos la congestión del sistema y mejoramos el rendimiento general.
- Portabilidad a través de hardware y software: HDFS puede ser transportado fácilmente de una plataforma a otra. Esto hace que HDFS sea muy versátil y muchas empresas o proyectos acaben adoptando HDFS como opción.

Ahora vamos a centrarnos en su arquitectura. Una representación gráfica de la estructura de HDFS podría ser la Ilustración 5. Como se puede ver, normalmente hay un NameNode y un número grande que puede llegar a ser cientos o miles de DataNodes. El cliente se comunica tanto con el NameNode como con los DataNodes. También tenemos otros dos nodos que son el Secondary NameNode y el Backup Node. Ambos no pueden estar a la vez, si está el Secondary NameNode no puede haber un Backup Node o viceversa. Más adelante veremos a qué se debe esto.

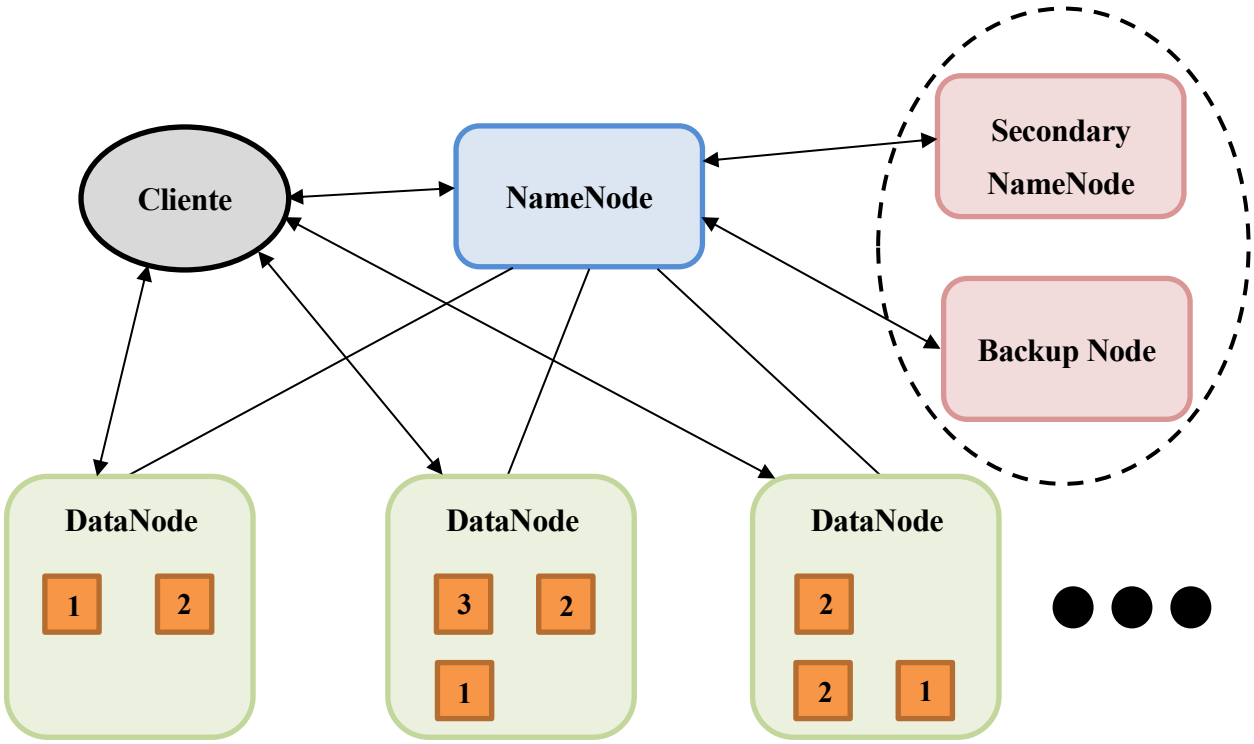


Ilustración 5: Representación gráfica de la estructura de HDFS [20]

Las principales funciones de los nodos más importantes son [17] [20]:

- NameNode: Es el servidor maestro, que administra el espacio de nombres del sistema de archivos y coordina todas las operaciones, incluyendo lectura y escritura de los archivos así como permisos,

propietarios y otras acciones. Contiene los metadatos sobre todos los bloques de archivos y en qué nodos se encuentran. Los NameNode almacenan los datos y metadatos en la memoria RAM para tener un acceso más rápido. Es un punto único de fallo, ya que solo hay uno en el clúster. Existe una configuración en la que pueden existir dos NameNode a la vez. En este caso lo que ocurre es que sólo uno de ellos estará activo, mientras que el otro se encontrará inactivo, actuando como un esclavo. Si el NameNode principal falla, entonces es cuando se activa el otro NameNode. De esta forma se mantiene una alta disponibilidad de HDFS.

- DataNodes: Almacenan los datos y son responsables de crear, eliminar y replicar bloques de datos. Estos envían constantemente los llamados "heartbeats" al NameNode para informar al NameNode de que están activos y funcionan correctamente.

Como hemos podido ver, el NameNode almacena información (metadatos) valiosa para el clúster. Esta información se almacena en dos ficheros específicos dentro del propio NameNode, son el fichero fsImage y el fichero editLog [20]:

- fsImage: este fichero contiene la estructura completa (el espacio de nombres) de HDFS, es decir, contiene la información sobre dónde están los datos y qué bloques están en cada nodo desde que el NameNode se inició. También contiene la información de todas las acciones realizadas en el clúster hasta un cierto momento concreto. Como va a ser un fichero en el que se va a guardar mucha información, éste se guarda en el disco duro del NameNode.
- editLog: contiene un registro de cualquier acción realizada en el clúster HDFS, como la creación de un bloque, eliminaxción del bloque, o similares. Va almacenando todas las acciones que ocurren en tiempo real. Como solo contiene información de los cambios más recientes, es un fichero más pequeño y por tanto se almacena en la memoria RAM.

Para entender el por qué de la existencia de los ficheros fsImage y editLog tenemos que tener en cuenta que el NameNode es un punto único de fallo. Contiene todos los metadatos de las operaciones que se han hecho y se están haciendo. Si le ocurre algo y falla se perdería toda esa información y podría causar problemas graves en el clúster, ya que todo el clúster depende del NameNode.

En el fichero fsImage se van almacenando todas las operaciones y toda la información necesaria. Al iniciarse el NameNode este fichero se carga en memoria. Para no tener que estar actualizando siempre este mismo fichero que se va a volver enorme, se tiene el fichero editLog, que también se carga en el arranque junto al fsImage. El fichero fsImage por tanto solo se usa una vez, en el arranque, el que se usa durante la ejecución del nodo es el editLog. Al guardarse en RAM se puede acceder rápidamente a él y se van guardando todos aquellas operaciones que se realizan a lo largo del tiempo.

El fichero editLog llegará un punto en el que también se hará demasiado grande y ocupe la memoria RAM, es por eso que existe el proceso llamado "checkpointing". Este proceso copia todos los últimos cambios guardados en el fichero editLog y los guarda en el último fichero fsImage que se tenía. Esto produce un nuevo fichero fsImage que es una fusión del fichero fsImage anterior y del actual editLog. El proceso de "checkpointing" solo ocurre una vez en el arranque del nodo. Si el NameNode falla, al iniciarse sólo tiene que cargar el último fichero fsImage guardado y luego cargar el editLog, de esta forma vuelve al estado en el que se encontraba antes del fallo, después fusiona ambos ficheros, crea el nuevo editLog y sigue su ejecución normal.

El proceso de "checkpointing" [21] puede llegar a consumir muchos recursos, además que el editLog va a seguir creciendo, haciendo que el NameNode vaya más lento de lo que debería. Por ello es que existe el Secondary NameNode. El Secondary NameNode copia el último fichero fsImage y editLog, y los fusiona en un nuevo fsImage, que es enviado de nuevo al NameNode. De esta forma el Secondary NameNode sirve como nodo de recuperación del NameNode. Este proceso de copia en el Secondary NameNode tiene por defecto que ocurra en un periodo de 1 hora periodicamente. También hay otro límite que es 1 millón de acciones almacenadas. Si el millón se alcanza antes de la hora, se realizaría el proceso de copia. Estos parámetros son configurables, por lo que se puede cambiar ese tiempo o el máximo de acciones.

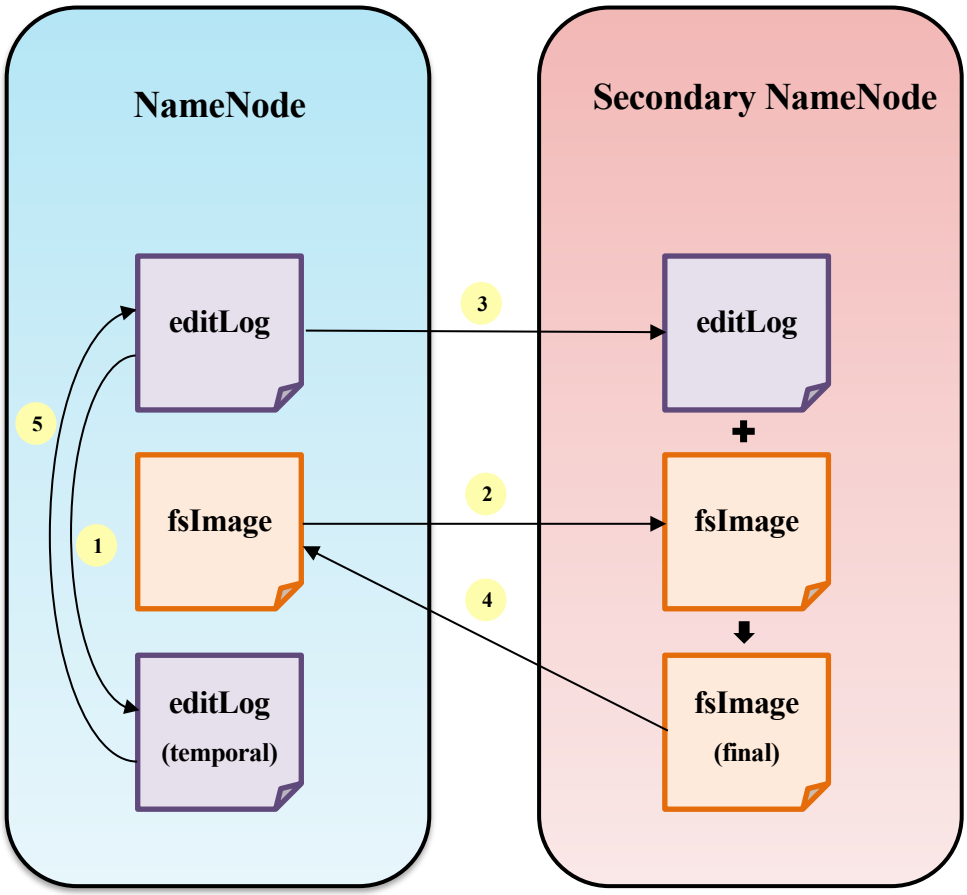


Ilustración 6: Proceso de "checkpointing" en HDFS [20]

Todo este proceso podemos verlo representado en la Ilustración 6. En ella podemos ver que lo que ocurre es que primero (paso 1) se crea un nuevo fichero editLog. Cuando se está realizando un "chekpoint", el NameNode sigue trabajando y aplicando cambios. Por eso se crea ese nuevo fichero editLog en el que se irán guardando todas las nuevas modificaciones. Hasta que no ocurra el siguiente "checkpoint" se mantendrá ese fichero, y cuando se comience uno nuevo, se creará otro editLog nuevo. Después (paso 2), se copia el fichero fsImage en el Secondary NameNode. Esto se hace sólo una vez, ya que el resto de veces se copiará el fichero editLog y se irá añadiendo al fsImage. A continuación (paso 3), se copia el editLog que se tenía en el Secondary NameNode. Ambos fichero se fusionan en el Secondary NameNode y se crea el nuevo fsImage que se sustituirá al fsImage que había en el NameNode (paso 4). Por último (paso 5) el fichero editLog nuevo que se había creado sustituirá al anterior y en él se escribirán los cambios.

Este proceso tiene un problema, y es que si el NameNode falla antes de que se pueda hacer un "checkpoint", las operaciones que se hicieran hasta ese momento se perderían. Aquí es donde entra el Backup Node que mencionamos antes. Este nodo realiza una copia en tiempo real del NameNode y también el proceso de "checkpoint". Es por eso que si usamos un Backup Node no tiene sentido de poner un Secondary NameNode, cuando su función ya la está realizando el Backup Node.

Ahora vamos a profundizar un poco en cómo funciona su sistema de almacenamiento [20]. El espacio de nombres del sistema de archivos de HDFS sigue una estructura jerárquica tradicional como la de otros sistemas. Se pueden crear directorios y almacenar ficheros en esos directorios. También se pueden eliminar esos ficheros, moverlos entre directorios y renombrar.

En este sistema de archivos el almacenamiento se realiza con un sistema en bloque. Este sistema consiste en dividir un fichero en bloques de un determinado tamaño. En el caso de HDFS, el tamaño por defecto es de 128 MB. Por ejemplo, si tenemos un fichero de 500 MB, HDFS lo divide en 3 bloques de 128 MB y 1 de 116 MB.

Estos bloques se distribuyen por los diferentes DataNodes, y cada bloque acaba en un DataNode diferente. Es importante que si el fichero no llega a 128, solo ocupa el tamaño que tenga, no más. Con dos o tres archivos parece que se ahorra poco (en el caso de ejemplo solo 12 MB los que se ahorran), pero cuando son miles de archivos el ahorro de memoria es considerable.

Este método de almacenamiento distribuido tiene muchas ventajas. Supongamos el fichero anterior que se dividió en 4 bloques. Cada bloque va a ser procesado por un nodo diferente, esto permite que la velocidad sea 4 veces mayor a si se realizase con un solo nodo. Por otra parte, si tenemos que guardar un fichero cuyo tamaño supera al del clúster, como el sistema es escalable, solo tenemos que añadir nuevos nodos hasta que la demanda de memoria quede satisfecha.

Para que no se pierda información cuando un nodo falla, HDFS lo que hace es replicar cada bloque de 128mb en 3 nodos distintos [19]. Si un nodo falla sigo teniendo 2 nodos con copias de ese bloque. Esto es debido a que Hadoop usa hardware normal y tiende a fallar. El valor de replicar 3 veces es por defecto, también se puede modificar. Esta forma de almacenamiento da lugar a una tolerancia completa a fallos.

Una representación sería la mostrada en la Ilustración 5. Como se puede observar, cada DataNode contiene unos cuadrados naranjas. Esos cuadrados vienen a representar un bloque de datos. El bloque número 1 está repartido en los 3 DataNodes mostrados, el bloque número 3 en los dos de la derecha y en otro distinto. Y así se irían distribuyendo todos los bloques por los DataNodes del clúster.

El mecanismo de réplica de un fichero también tiene un procedimiento [17]. Supongamos que vamos a almacenar un fichero en HDFS que se ha dividido en dos bloques, A y B. El cliente lo primero que hace es mandar una solicitud al NameNode para informar que quiere almacenar un bloque. El NameNode, si está configurado por defecto, le mandará 3 direcciones IP distintas, que serán las direcciones de los tres DataNodes donde debe realizar la copia. Le manda las direcciones de los DataNodes 1, 2 y 3. El cliente primero pregunta al nodo 1 si está preparado para copiar un fichero en él, y le manda las direcciones de 2 y 3 para que él le pregunte lo mismo. A esto 1 responde y 1 pregunta a 2, y 2 pregunta a 3. Cuando se tiene confirmación, primeramente se copia el bloque en el nodo 1, después 1 lo copia en 2 y 2 lo copia en 3. Si el cliente no recibiese ninguna confirmación, se volvería a preguntar al NameNode para volver a obtener otras 3 direcciones IP distintas. La copia del bloque B se hace al mismo tiempo que la copia del A, y el mecanismo es exactamente el mismo, pero obviamente se usan 3 direcciones IP distintas.

Cuando se quiere leer un fichero es mucho más sencillo que copiarlo. El cliente primero le dice al NameNode que quiere leer un fichero en particular. El NameNode le devuelve las direcciones IP de los nodos donde ese fichero está copiado. De esta forma el cliente acude a esas IP y recibe todos los bloques al mismo tiempo.

Por último, mencionar la existencia de las "HDFS Federation" [21]. Viene a resolver un problema de escalabilidad. Básicamente permite dividir el sistema en particiones y establece un NameNode para cada una de esas particiones. Cada NameNode va a actuar de forma totalmente indepediente sin requerir coordinación entre ellos. Esto es muy efectivo para cuando tenemos un clúster de grandes dimensiones.

3.1.2 MapReduce

Ahora vamos a ver qué es MapReduce. MapReduce es un framework que nos permite procesar grandes cantidades de datos. Se caracteriza porque nos permite hacerlo de forma paralela en clústers muy grandes, en hardware básico y de una forma totalmente tolerante a fallos [17]. Nos encontramos de nuevo ante una estrutura maestro-esclavo.

La arquitectura maestro-esclavo se debe a que la arquitectura de MapReduce tiene dos procesos principales:

- JobTracker: Es el proceso maestro. Es el responsable de coordinar y completar un trabajo MapReduce en Hadoop. Sus principales funciones son la gestión de recursos, seguimiento de la disponibilidad de los recursos y controlar el ciclo del proceso de las tareas. Es un punto único de fallo en MapReduce.
- TaskTracker: Es el proceso esclavo. Realiza las tareas que les asigna el JobTracker. Los TaskTracker envian los "heartbeats" periodicamente al JobTracker y le notifican si hay espacios libres y le dice el estado de la tarea o verifica si debe realizar alguna.

MapReduce realmente tiene numerosos procesos y tareas pero en la mayoría de casos solo nos tenemos que

preocupar de tres procesos: el Map, Suffle and Sorting y Reduce [20].

- Map: Es el proceso que consigue el paralelismo. Lee y procesa los datos obteniendo como salida un conjunto de pares clave-valor <Key, Value>. Esta salida no se guarda en HDFS, se crea un archivo intermedio que es procesado por la siguiente tarea.
- Shuffle and Sorting: Este proceso es manejado por Hadoop y puede ser suprimido si fuese necesario. Procesa la salida del Map agrupando los valores clave y agregando los valores a una lista de valores. La salida que se produce es <Key, List<Lista de Values>>. Esta salida es enviada al Reduce.
- Reduce: Es el proceso de agregación que obtiene la salida del Suflle and Sorting. Su salida final se almacena en HDFS.

El objetivo final es pasar de tener un conjunto de datos muy grande a uno de menor tamaño en un formato de pares clave-valor. Por tanto, para una explicación mejor tenemos la Ilustración 7 en la que podemos ver un ejemplo de un proceso completo de MapReduce. Vamos a explicar que ocurre en cada paso representado:

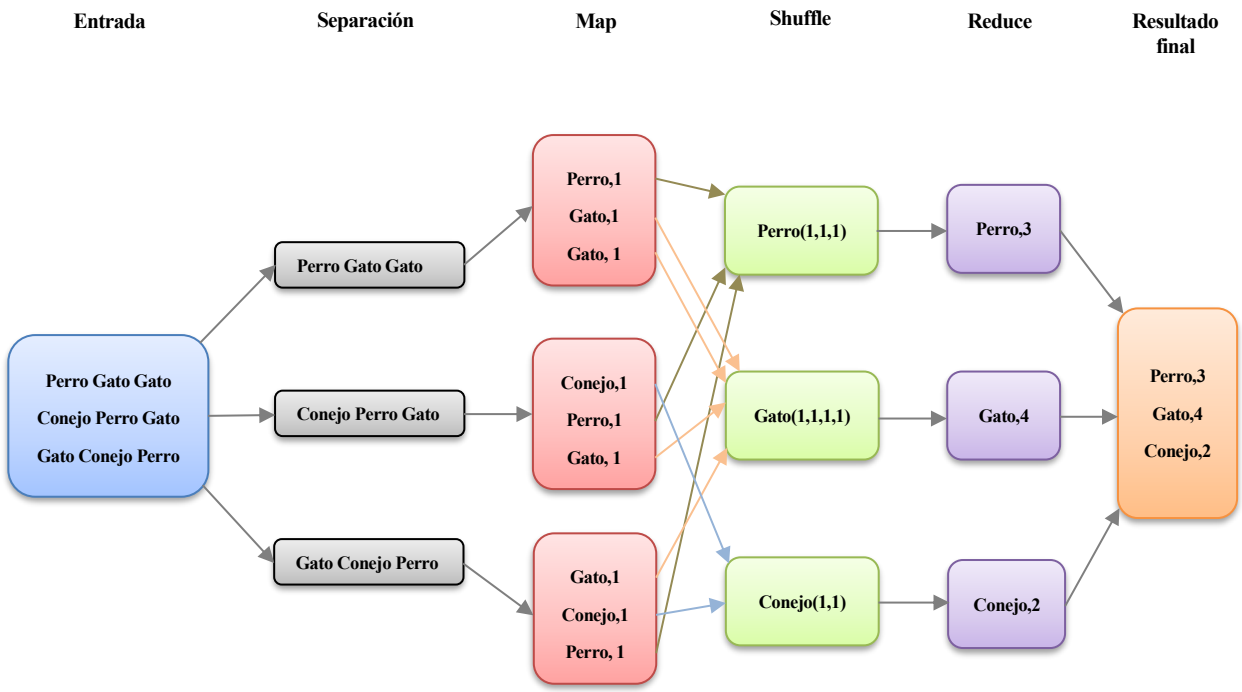


Ilustración 7: Ejemplo de MapReduce

- Entrada: Al inicio tenemos un fichero de texto que estará almacenado en HDFS. En el ejemplo que muestra la Ilustración 7 tenemos un fichero de texto plano que contiene 9 palabras. Estas palabras son repeticiones de 3 animales, que son perro, gato y conejo.
- Separación: Lo primero que se hace es dividir el fichero en bloques del mismo tamaño. En este caso se dividen en 3 bloques y cada uno tiene 3 palabras.
- Map: Esta tarea forma las pareja clave valor. Por ejemplo del primer bloque separa las palabras y le pone de valor un 1, es decir, la palabra aparece 1 vez.
- Shuffle: Se juntan todas las palabras que son iguales y sus valores. Por ejemplo perro aparece 1 vez en el primer bloque, 1 vez en el segundo y 1 vez en el tercero, por ello podemos ver que aparece (1,1,1).
- Reduce: Se suman los valores y de esta forma tenemos que la palabra “perro” aparece 3 veces, “gato” aparece 4 y “conejo” aparece 2.
- Resultado final: Finalmente, obtenemos un fichero nuevo que se almacenará en HDFS. Este fichero tendrá todas las palabras que aparecían en el fichero de entrada, y cuántas veces aparece cada palabra.

MapReduce es un programa Java [17]. Las claves y los valores deben ser serializables, pero no usan el paquete

serializable de Java, sino que usan otra interfaz. Se usan dos interfaces, Key tiene que implementar la interfaz WritableComparable, y Value la interfaz Writable. Generalmente se pueden crear los programas Java para realizar las tareas MapReduce, pero como ya veremos existen herramientas como Pig que nos pueden facilitar mucho este proceso.

3.1.3 Yarn

Yarn (Yet Another Reource Negotiator) nace con la idea de separar las funcionalidades de administrar recursos y las de monitorizar y programar tareas [22]. Permite ejecutar múltiples aplicaciones distribuidas en Hadoop. Por tanto Yarn es un negociador de recursos. Vamos a tener otra estructura maestro-esclavo. En este caso tenemos como master al ResourceManager y como esclavos a los NodeManager. Fue lanzado en la version 2.0 de Hadoop en el año 2012, suponiendo un gran cambio.

Cuando hablamos de Big Data, vimos que no es sólo importante ser capaces de procesar grandes cantidades de datos, sino de ser capaz de hacerlo en tiempo real. MapReduce está diseñado para hacer un procesamiento por lotes, por lo que no puede procesar en tiempo real. Yarn sí nos permite procesar aplicaciones interactivas y en tiempo real, de ahí su importancia [23].

Para entender mejor Yarn, tenemos que comparar las versiones de Hadoop. En la version Hadoop 1.0, sólo se encontraba MapReduce 1.0 y HDFS, en la version Hadoop 2.0 ya se añadió Yarn con MapReduce 2.0. En MapReduce 1.0 teníamos dos demonios diferentes, el JobTracker y los TaskTrackers. El JobTracker se encargaba tanto de la planificación como de la monitorización. Los TaskTrackers ejecutaban tareas e iban enviado informes de estado a los JobTrackers. En cambio, en Yarn esas dos funciones del JobTracker las realizan dos entidades diferentes, que son el Resourcer Manager y el Application Master. Se va a tener uno por cada tarea MapReduce.

Yarn posee varios componentes y subcomponentes. Como tiene una estructura maestro-esclavo, el maestro va a ser el llamado Resource Manager, que a su vez tiene otros componentes, que son el Scheduler, Application Manager y Resource Tracker. Los esclavos serían los Node Manager y también tenemos un Application Master por aplicación y un Container por tarea MapReduce. Veamos qué función cumple cada uno [17] [22] [23]:

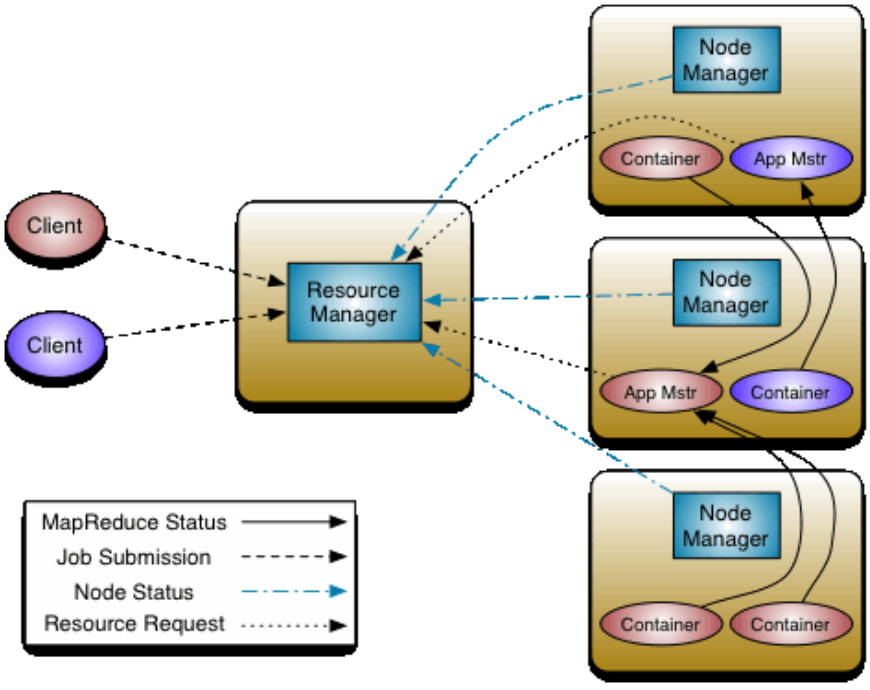


Ilustración 8: Representación gráfica de la estructura de YARN [22]

- Resource Manager: se encarga de organizar todos los recursos (memoria, CPU o RAM, entre otros)

para las aplicaciones del sistema. Va a recibir peticiones de procesado, como por ejemplo tareas MapReduce, que las va a ir asignando a los NodeManager. Además, va monitorizando si una tarea MapReduce se está llevando a cabo correctamente o no. Sus subcomponentes son:

  - Scheduler: Se encarga de asignar recursos a las aplicaciones en ejecución según ciertas limitaciones pero sin monitorizar el estado de esa aplicación. Solo se basa en los requisitos de cada aplicación y según los recursos disponibles los va asignando.
  - Application Manager: Tiene una colección de todas las aplicaciones ejecutadas. Mantiene en caché cuales han sido completadas para poder atender solicitudes de los usuarios aunque haya pasado mucho tiempo desde que terminaron.
  - Resource Tracker: Contiene la configuración ensencial como cada cuanto tiempo hay que comprobar que los Container siguen disponibles, o cuanto tiempo debe pasar para considerar que un NodeManager no está funcionando.

- Node Manager: se encuentra en el resto de nodos y se encarga de monitorizar el consume de recursos, por ejemplo el uso de memoria, CPU y RAM en un solo nodo y reporta estos estados al ResourceManager. También envían los "heartbeats" al ResourceManager con la información de los recursos.
- Application Master: Negocia los recursos del ResourceManager y ejecuta y monitoriza los Container de los NodeManager. La instancia de un Aplication Master dura el mismo tiempo que la Aplicación para el que fue creado.
- Container: Representa una colección de recursos físicos como CPU, memoria o RAM. Cuando una aplicación va a ser ejecutada con Yarn, el cliente asocia un Container desde el Resource Manager, que es donde se correrá el Application Master. Una vez iniciado, pouede solicitar más contenedores al Resource Manager. Para las tareas MapReduce se tiene un container específico lamado MRAppMaster, en el que cada tarea Map y Reduce tienen su propio Container.

En la Ilustración 8 podemos ver una representación de los componentes de Yarn. El cliente solo se comunica con el Resource Manager. El Resource Manager se comunica con todos los componentes excepto los Container, que son manejados por los Application Master. Para entender mejor cómo es el proceso de ejecución de una aplicación en Yarn tenemos la Ilustración 9.

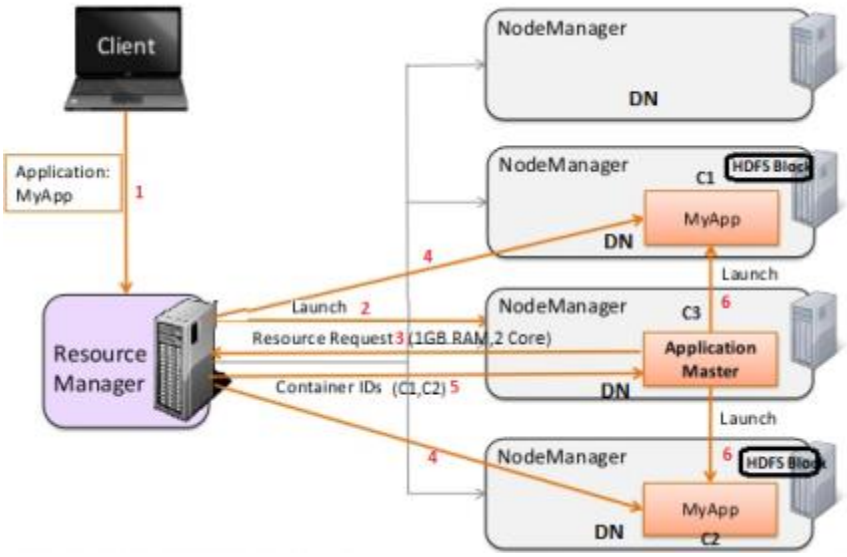


Ilustración 9: Flujo de ejecución de una tarea en YARN [23]

Veamos los pasos detalladamente:

- 1.- El cliente envía la aplicación a ejecutar al Resource Manager.

- 2.- El Resource Manager manda a un Node Manager que ejecute esa aplicación.
- 3.- Se crea un Application Master en ese Node Manager y le manda una solicitud de recursos al Resource Manager.
- 4.- El Resource Manager comprueba la disponibilidad de recursos y observa que C1 y C2, que son dos container en dos nodos distintos pueden proporcionar lo necesario. Por tanto, le envía esta información a esos Node Manager.
- 5.- Le envía la información de los recursos y containers a C3.
- 6.- El Application Master ejecuta la aplicación usando los conainer en ambos Node Manager.

Como podemos ver, con esta estructura Hadoop pasó a ser mucho más versátil, incluso se compara con pasar de un ordenador con un sistema operativo en el que sólo tenemos un bloc de notas, a un sistema operativo con múltiples y diferentes aplicaciones como son los actuales [23].

3.1.4 Versiones de Hadoop

En el apartado de YARN anterior mencionamos que existen varias versiones de Hadoop, en concreto mencionamos la versión de Hadoop 1.x y la 2.x. Existe otra más que es la versión 3.x, cuya última version es la 3.3.x [24]. En este apartado vamos a mencionar las principales diferencias entre las versiones de Hadoop, para así entender mejor su desarrollo, y las diferentes soluciones que se han propuesto a los problemas que se han ido presentando.

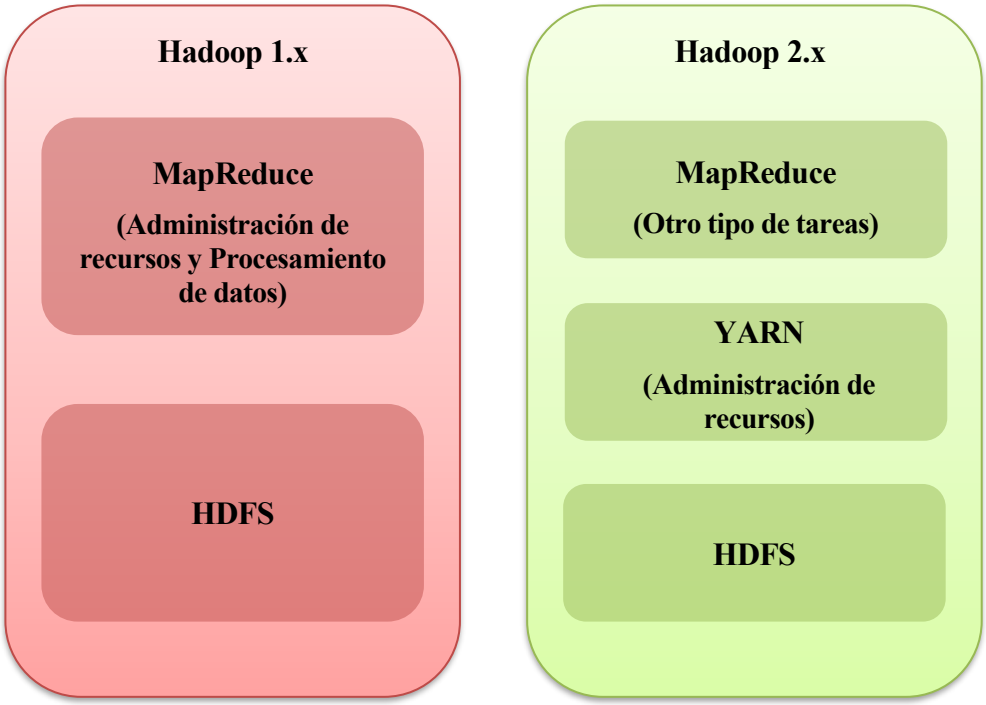


Ilustración 10: Diferencias entre la arquitectura de Hadoop 1.x y Hadoop 2.x

Como ya sabemos, la principal diferencia entre Hadoop 1.x y 2.x es la introducción de YARN. Pasamos de sólo poder procesar aplicaciones por lotes a aplicaciones interactivas y en tiempo real. Pero no sólo se introdujeron estas mejoras, hay muchas más. Las principales a destacar son [25]:

- Con Hadoop 1.x sólo podíamos tener un NameNode para poner administrar el espacio de nombres completo, pero con Hadoop 2.x se introdujeron las federaciones que comentamos en el apartado de HDFS, permitiendo que existieran varios NameNode por clúster.

- Con Hadoop 2.x se introdujo el soporte para Microsoft Windows, ya que Hadoop 1.x no tenía ese soporte.
- Hadoop 1.x soportaba 4000 nodos por clúster, pero Hadoop 2.x soporta hasta 10.000 nodos por clúster.
- Hadoop 1.x no soportaba la escalabilidad horizontal, pero Hadoop 2.x sí.

Estas son las principales diferencias que podemos destacar entre estas dos versiones de Hadoop. Entre Hadoop 2.x y Hadoop 3.x también tenemos numerosas diferencias, pero vamos a destacar una de ellas. Esta es que en Hadoop 2.x la tolerancia a fallos se debe gracias a la replicación como se comentó anteriormente, pero en Hadoop 3.x se puede realizar por un nuevo método llamado "HDFS erasure coding".

El principal problema del método de réplica es que como el factor predeterminado de replicación es 3, se tiene un 200% de sobrecarga en el almacenamiento. Es decir, por cada fichero se está ocupando 3 veces su tamaño en memoria. En Hadoop 3.x, esta sobrecarga no llega al 50% y es gracias a este nuevo método de "erasure coding".

No vamos a entrar en detalle acerca de cómo funciona este método, pero a grandes rasgos, lo que se hace es dividir los datos en unidades más pequeñas como bit, bytes o bloques, y se almacenan las unidades consecutivas en diferentes discos. Para cada unidad de datos se carcula un cierto número de celdas de paridad en función de un algoritmo elegido (algoritmo XOR o el algoritmo de Reed-Salomon) y se almacena, este sería el proceso de "encoding". Después, cualquier error puede ser recuperado del cálculo basado en los datos restantes y las celdas de paridad, este sería el proceso de "decoding". Así, de un archivo que se divide en 6 bloques, pasamos a ocupar 18 bloques (6x3) en el almacenamiento con el método de réplica, a sólo 9 bloques (6 de datos y 3 de paridad) con el método de "erasure coding" [26].

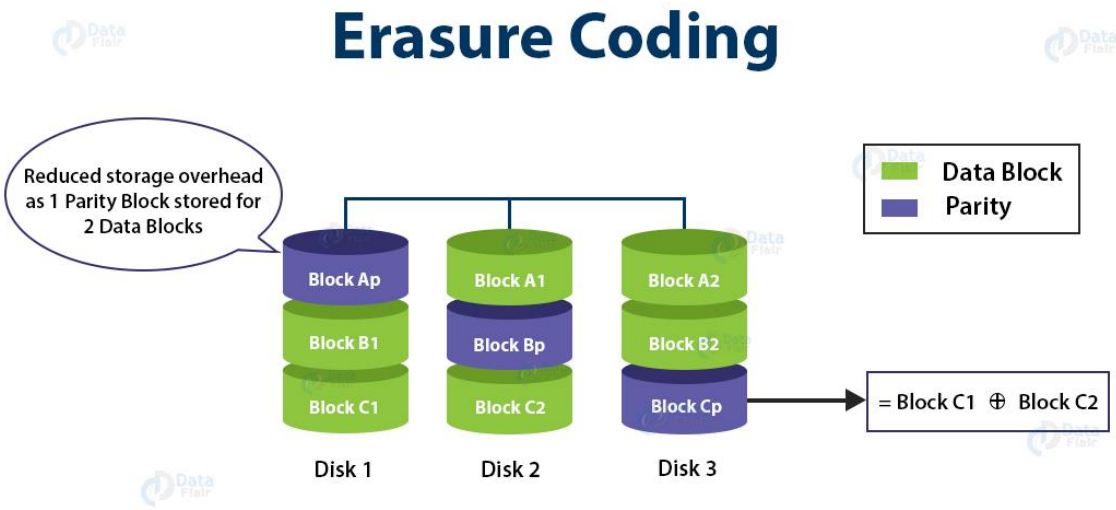


Ilustración 11: Representación gráfica del método "Erasure coding" [26]

Otras diferencias entre Hadoop 2.x y 3.x es que se cambia el rango de puertos por defecto, se soportan más de 10.000 nodos por clúster, se soporta la versión 11 de Java, entre otros cambios[27].

3.1.5 Ecosistema Hadoop

Un clúster de Hadoop con miles de nodos es fácil de imaginar que no es tarea sencilla configurar y administrar todo el clúster. Para ayudar a esto, tenemos el ecosistema Hadoop, que es un conjunto de herramientas muy extenso para poder administrar los datos de manera fácil y efectiva. [17]. El ecosistema Hadoop consta de un conjunto muy amplio de subproyectos que, aunque en su mayoría están administrados por Apache, también hay algunos administrados por otras empresas u organizaciones. Un primer vistazo al ecosistema Hadoop lo tenemos en la Ilustración 12.

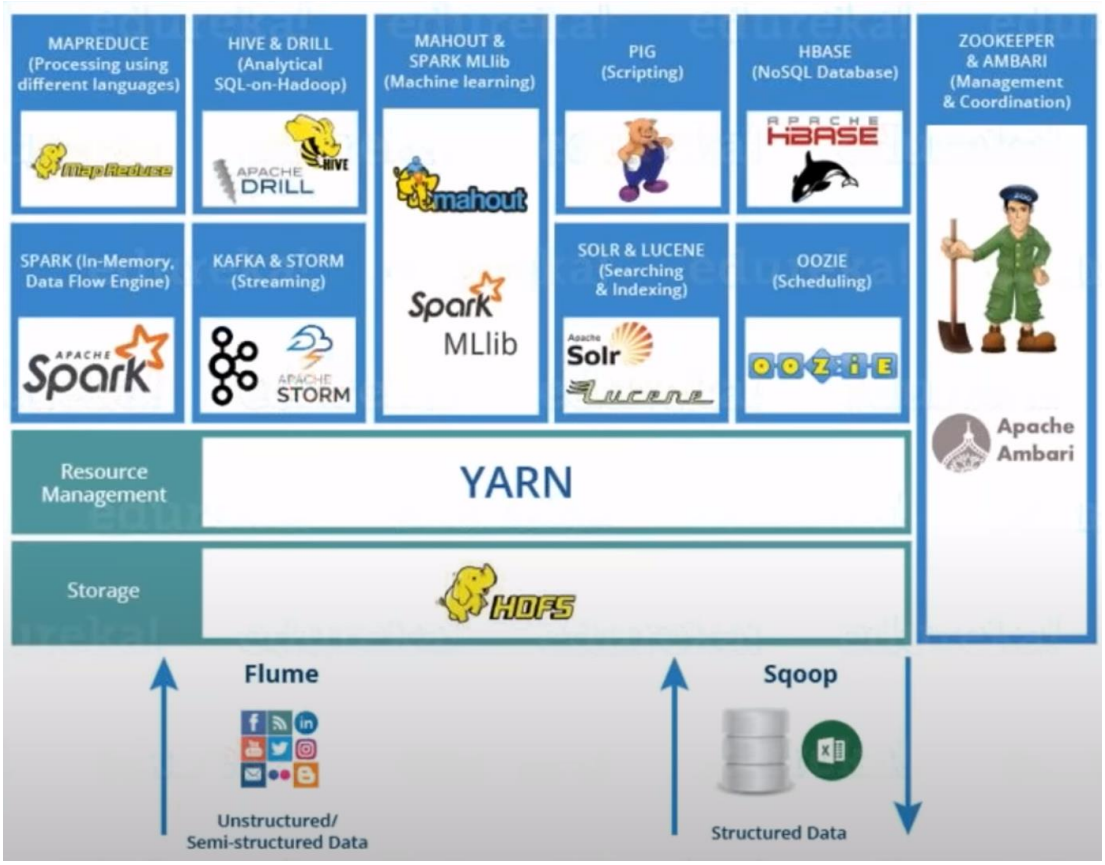


Ilustración 12: Ecosistema Hadoop [20]

Como podemos ver, los componentes HDFS, Yarn y MapReduce que ya hemos analizado a fondo también forman parte de este ecosistema. Pero el resto de herramientas también forman parte de Hadoop y es por ello que vamos a explicar brevemente en qué consisten algunas de ellas.

Vamos a comenzar por herramientas que ayudan a la ingesta de datos en nuestro clúster Hadoop. Estas son Flume y Sqoop. Estas herramientas nos ayudan a la ingesta de datos a muy alta velocidad. Cada una de ellas está pensada para un tipo de datos diferente. Por tanto tenemos que Flume es para la ingesta de datos no estructurados o semiestructurados, mientras que Sqoop es para la ingesta de datos estructurados [20].

Para el análisis de datos tenemos herramientas como Hive y Apache Drill. Hive fue desarrollado por Facebook y nos permite realizar el análisis de los datos en un lenguaje similar a SQL. Hive funciona sobre MapReduce por lo que Hive ejecutará las tareas MapReduce. Apache Drill nos permite realizar consultas en multitud de bases de datos noSQL y sistemas de archivos. De esta forma, con una simple consulta, podemos unir datos de diferentes sistemas de almacenamiento [28].

Otras herramientas nos ayudan con el scripting como Apache Pig. Apache Pig nos permite realizar tareas MapReduce con una sola línea, de manera que una línea en Pig son 20 líneas de código Java MapReduce. Pig lo que hace es traducir ese comando a lenguaje MapReduce y por eso es tan sencillo. Estudiaremos su funcionamiento a fondo más adelante, ya que es un componente importante en este trabajo [20].

Tenemos herramientas para el streaming y análisis de datos en tiempo real como Apache Spark, Apache Storm o Apache Kafka. Apache Spark es una alternativa muy potente a MapReduce, ya que es un framework de de procesamiento de datos en paralelo que puede llegar a ejecutar programas 100 veces más rápido que Hadoop en memoria, o 10 veces más rápido en disco [17]. Apache Storm es un sistema de computación en tiempo real que hace un procesamiento fiable de los datos en tiempo real, permitiendo aplicaciones como analíticas en tiempo real, machine learning online y otras posibilidades. [29]. Y por último, Apache Kafka es una plataforma de "event streaming". Estos significa que usa el patrón publicador-suscriptor. De esta forma nos permite publicar y suscribirnos a flujos de eventos, almacenarlos y procesarlos [30].

Existen otras herramientas para la administración y coordinación entre los diferentes componentes como Apache

Zookeper y Apache Ambari. Apache Zookeeper es un servicio de coordinación distrubuido. Tiene una serie de primitivas que las aplicaciones pueden usar para sicronización o mantenimiento. Apache Ambari nos permite administrar las actividades en un clúster Hadoop. Con él podemos instalar, desarrollar, administrar o monitorizar un clúster Hadoop. Además, nos proporciona una interfaz gráfica web muy intuitiva para todas estas tareas [17].

Otros también nos permiten planificar el flujo de trabajo como Apache Oozie. Apache Oozie es un sistema de procesamiento de servicios de coordinación y flujos de trabajo. Permite administrar múltiples trabajos o cadenas de trabajos escritos en MapReduce, Pig, Hive, Java u otros lenguajes. Nos permite establecer reglas tanto para el inicio como para el final de un flujo de trabajo, y también puede detectar la finalización de tareas [17].

Por último, también tenemos herramientas para la búsqueda e indexación de información como Apache Lucene y Apache Solr. Apache Lucene es una librería de Java que proporciona funciones de búsqueda e indexación, corrección ortográfica, acentos y similares. Y Apache Solr se basa en Lucene para ofrecer un servidor de búsqueda con las características de Lucene a través de interfaces JSON, HTTP o Java. [31]

Incluso existen herramientas para Machine Learning como Apache Mahout que es una API escalable de machine learning que puede ser usada de forma independiente o integrada con Hadoop [17].

Aparte de todos los componentes que hemos mencionado, existen otros muchos más, ya que el ecosistema de Hadoop es muy grande. El uso y la implementación de los diferentes componentes vendrá determinado por la aplicación que queramos desarrollar. Según necesitemos una función u otra, será necesario analizar las diferentes opciones y, finalmente, decantarse por una de ellas.

3.1.5.1 Pig

Como se ha explicado anteriormente, en este trabajo usaremos Apache Pig en nuestra aplicación práctica. Por eso se ha includo este subapartado, en el que explicaremos Apache Pig en más detalle.

Pig nació originalmente en Yahoo! en el año 2006, a partir del año 2007 fue adoptado por la Apache Software Foundation, hasta que finalmente surgió la versión inicial como parte de un subproyecto de Hadoop. Pig es una plataforma que nos permite analizar grandes cantidades de datos usando un lenguaje de alto nivel. Básicamente es una plataforma de scripting, que permite realizar programas MapReduce con muy pocas líneas de código. El lenguaje que utiliza se denomina Pig Latin, el cual tiene unas características que lo hacen muy interesante que son la facilidad de programación, las oportunidades de optimización y la extensibilidad. Estas características nos permite podernos enfocar más en el análisis de los datos, y no tener que pasar horas en el desarrollo del código, lo cual es una gran ventaja. Además, nos permite crear nuestras propias funciones gracias a las funciones definidas por el usuario o UDF (User Defined Functions), que usando lenguajes como Java, Python o Ruby podemos escribir funciones concretas en el caso de que lo necesitemos para nuestro análisis [32].

El nombre de esta herramienta hace referencia a todas sus funciones. En español, pig es el animal cerdo, y de este animal se definió la “Pig philosophy”, que concentra todas las características [33]:

- “Pigs eat anything”, es decir, los cerdos comen cualquier cosa. Esto hace referencia a que Pig soporta cualquier tipo de dato, ya sea estructurado, semiestructurado o no estructurado.
- “Pigs live anywhere”, es decir, los cerdos viven en cualquier sitio. Aunque en un principio Pig estaba pensado para funcionar sobre Hadoop, ya no está orientado solo a él.
- “Pigs are domestic animals”, es decir, los cerdos son animales domésticos. Esto se refiere a las UDF que hemos descrito, el usuario puede crear sus propias funciones y de esta forma puedan personalizar como quiera el análisis de los datos.
- “Pigs fly”, es decir, los cerdos vuelan. Aunque los cerdos en la realidad no vuelan, con esto quieren explicar que el principal objetivo es mejorar el rendimiento en vez de las funcionalidades. De esta forma Pig puede procesar los datos rápidamente y que una gran cantidad de funciones no sean un lastre.

Vamos a explicar un poco qué es y cómo funciona Pig Latin. Pig Latin es un lenguaje de alto nivel, que sirve para flujo de datos y trabaja en paralelo. De esta forma, permite a los programadores decidir cómo los datos entrantes deben ser leídos, procesados y almacenados. Su sintaxis es muy similar a la de SQL, con operadores tradicionales como son JOIN, SORT o FILTER, y otro tipo de operadores. En la Tabla 2 [34] tenemos un conjunto de operadores y un breve resumen de su función, aunque los veremos de forma detallada en el Capítulo

5.

Carga y almacenamiento	LOAD	Carga datos desde un sistema de archivos en una relación
	STORE	Guarda la relación en un sistema de archivos dentro de un directorio
	DUMP	Imprime en consola la relación.
Filtrado	FILTER	Remueve filas no deseadas de una relación según un filtro
	DISTINCT	Remueve filas duplicadas de una relación
	FOREACH	Agrega, mueve o modifica campos de una relación
	SAMPLE	Selecciona una muestra aleatoria de una relación
	ASSERT	Verifica que una condición sea verdadera para todas las filas de una relación
Agrupamiento	JOIN	Une dos o más relaciones en base a uno o más campos
	COGROUP	Agrupa datos de dos o más relaciones en base a uno o más campos
	GROUP	Agrupa datos de una relación
	CROSS	Realiza el producto cartesiano de dos o más relaciones
	CUBE	Realiza una agregación para todas las combinaciones de los campos especificados en una relación
Ordenamiento	ORDER	Ordena una relación por uno o más campos
	RANK	Asigna un rango para cada tupla en una relación, opcionalmente ordenando por un campo
	LIMIT	Limita el tamaño de una relación a un número máximo de tuplas
Combinación y separación	UNION	Combina dos o más relaciones en una
	SPLIT	Divide una relación en una o más relaciones

Tabla 2: Operadores Pig Latin [34]

Pig Latin tiene numerosos tipos de datos como son int, long, float, double u otros. Después tenemos otros tipos de datos más complejos que son los datos que las sentencias en Pig toman como entrada. Estos datos de entrada son los "bag", y un "bag" es un conjunto de tuplas. Una relación es una "bag". Una tupla es un conjunto ordenado de campos y cada campo es cualquier tipo de dato. También tenemos los "maps", que es un conjunto de pares clave/valor. También tiene operadores aritméticos como "addition" (+), "subtraction" (-), entre muchas otras opciones [32].

El proceso principal consiste en cargar los datos desde HDFS, realizar las transformaciones (que en el fondo es MapReduce), y finalmente se hace DUMP para mostrar el resultado por pantalla o STORE para almacenarlos.

La importancia de Pig se puede demostrar, ya que no sólo empresas como Yahoo! han usado Pig, otras como LinkedIn y su función de "Gente que podrías conocer", Twitter para analizar los tweets u otros, AOL o

WhitePages también han usado Pig y Hadoop, por lo que tiene una gran importancia.

Destacar que Pig tiene seis modos de ejecución [32]:

- Modo local: Todos los archivos serán instalados y ejecutados usando el host y el sistema de archivos local.
- Modo local Tez: Es similar al anterior, pero se invocará el motor en tiempo de ejecución de Tez (Tez es otro componente del ecosistema Hadoop).
- Modo local Spark: Es similar al modo local, pero se invocará al motor en tiempo de ejecución de Spark.
- Modo MapReduce: Es el modo por defecto, se ejecutará sobre un clúster Hadoop y una instalación HDFS.
- Modo Tez: Se ejecutará sobre Tez.
- Modo Spark: Se ejecutará sobre Spark.

Como podemos ver, Pig a una herramienta muy potente que puede facilitar mucho las cosas a la hora de tener que relizar un análisis de datos en Hadoop. Por tanto, en este trabajo podremos usar Pig para realizar las tareas MapReduce de manera sencilla y efectiva, aunque más adelante se explicará cómo se hará detalladamente.

3.1.6 Distribuciones Hadoop

Como ya sabemos, el ecosistema Hadoop es muy extenso y ofrece multitud de herramientas para diferentes soluciones. Pero existen otros ecosistemas que se denominan distribuciones Hadoop con otra gran variedad de opciones y herramientas. Las distribuciones Hadoop son ofrecidas por diferentes empresas, algunas son gratis y otras tienen un costo para poderlas usar.

Sabemos que el ecosistema Hadoop ofrece sus herramientas de forma gratuita, ya que son “open source”. Pero las distribuciones tienen unas ventajas extra. Además de ofrecer diferentes herramientas y soluciones, también ofrece soporte comercial, asistencia técnica, funciones empresariales, entre otros, lo que permite reducir los efuerzos para poder desarrollar y desplegar un nuevo producto o proyecto. Además, en su mayoría también ofrecen interfaces gráficas que facilitan aún más todas estas tareas.

En el mercado existen numerosas distribuciones, pero vamos a destacar las siguientes [17] [35] [36]:

- Cloudera: Fue la primera en lanzar una distribución comercial de Hadoop, por lo que actualmente es la más usada a nivel global. Doug Cutting, el creador de Hadoop, se unió a esta compañía en 2009, y desde entonces ha trabajado en ella. Proporciona herramientas como la consola Cloudera Manager, que muestra información de forma sencilla y la Cloudera Manager Suite, que tiene un asistente para gestionar recursos. También ofrece servicios de consultoría, formación y certificación. Gran parte de su ecosistema es “open source”, lo que reduce en gran medida el riesgo a depender de esta distribución. Algunos de sus clientes son marcas como Cisco, Siemens o Samsung
- Hortonworks: Esta distribución es completamente libre y se centra en mejorar la usabilidad del ecosistema Hadoop. Su uso es gratuito, pero proporciona un servicio técnico de pago. Es la única distribución que soporta Windows y es el principal contribuyente del proyecto Ambari del que hablamos anteriormente. Algunos de sus clientes son Ebay, Bloomberg o Spotify.
- MapR Technologies: Se centra en ofrecer el máximo rendimiento y tolerancia a fallos. Ha centrado todos sus esfuerzos en hacer fiables y eficientes todas las implementaciones de Hadoop. MapR usa su propio sistema de archivos nativo en Unix, que es NFS. En NFS se pueden usar los comandos nativos de Unix en vez de los comandos de Hadoop. Algunos de sus clientes son HP o Audi.

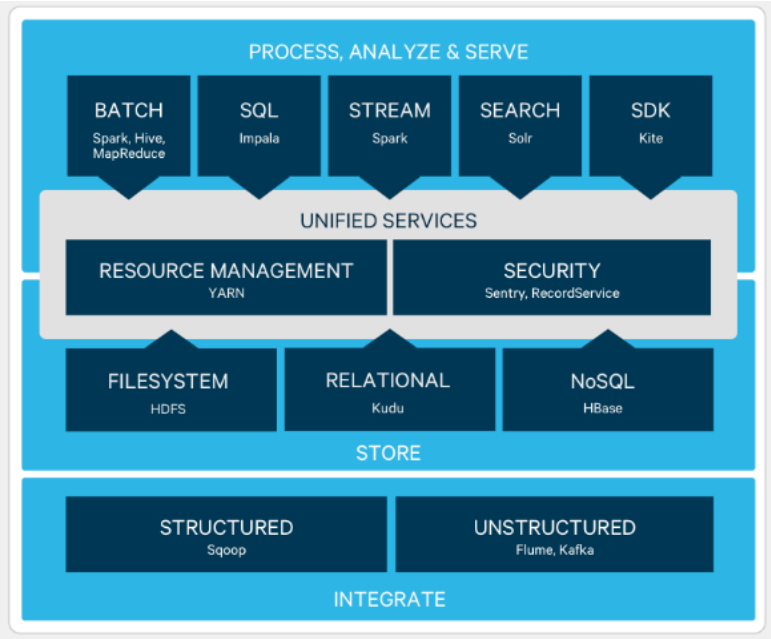


Ilustración 13: Ecosistema Hadoop Cloudera [37]

- Amazon Elastic MapReduce (EMR): Ofrece soluciones para los clientes que desean implementar los servicios de Hadoop aprovechando sus servicios en la nube como Amazon EC2 o Amazon S3.

Al igual que con el ecosistema Hadoop, la decisión de utilizar una de estas distribuciones reside en muchos factores. Dependiendo de los objetivos, el tipo de datos, las herramientas necesarias, si se dispone de presupuesto o no, si se necesita servicio técnico o no, u otros factores, será mas recomendable elegir una opción u otra.

4 DESCRIPCIÓN DE LA SOLUCIÓN

En este capítulo vamos a ver tanto los recursos hardware como los software que utilizaremos en este proyecto. Además, explicaremos como se conectarán todos los componentes de forma que el funcionamiento sea correcto, y todos los componentes puedan ser usados de forma cómoda.

4.1 Equipo necesario

Empezaremos con los componentes hardware más importantes que vamos a utilizar en este trabajo.

4.1.1 Ordenador portátil MSI GE60 2PC Apache

El ordenador portátil se usará para conectarse usando SSH a cada una de las Raspberry Pi. De esta forma la instalación y configuración de los diferentes componentes será mucho más sencilla. El ordenador es de uso personal y cuenta con las siguientes características:

- Procesador Intel® Core™ i5-4200H CPU @ 2.80GHz, 2801 MHz, 2 procesadores principales, 4 procesadores lógicos.
- 16 GB de memoria RAM
- Sistema Operativo Windows 10 64 bits
- 256 GB disco duro principal SSD, 500 GB disco duro secundario HDD
- Tarjeta gráfica integrada Intel® HD Graphics 4600 y NVIDIA GeForce GTX 850M de 2 GB dedicada

4.1.2 Raspberry Pi 3 Model B+

Las Raspberry Pi serán los nodos del clúster Hadoop. En este caso se usarán cuatro Raspberry Pi 3 model B+, por lo que tendremos un NameNode y tres DataNodes. Las carácterísticas de cada Raspberry Pi 3 model B+ son las siguientes [38]:

- Procesador Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC a 1.4GHz.
- 1 GB LPDDR2 de memoria RAM
- 40 pines GPIO
- Puerto HDMI
- Puerto CSI para cámara
- Puerto DSI para pantalla táctil
- Toma de auriculares
- Micro usb para alimentación

- Puerto Ethernet con capacidad Power-over-Ethernet (PoE)
- 2 puertos USB 2.0

4.1.3 Switch TP-Link TL-SG108 V3.0

Se usará un switch o conmutador para interconectar las Raspberry Pi y el ordenador portátil. En este caso se ha elegido un switch de la marca TP-Link que es "Plug and Play", es decir, no necesita ningún tipo de configuración para funcionar. Cuenta con 8 puertos ethernet RJ45 gigabit 10/100/1000 Mbps.

Ilustración 14: Switch TP-Link TL-SG108 V3.0 [39]

4.1.4 Tarjetas microSDHC SanDisk Ultra 32GB

Se necesitarán cuatro tarjetas microSDHC para cada una de las Raspberry Pi. En este caso tenemos dos modelos distintos, tres tarjetas microSDHC Sandisk Ultra de 32 GB, clase 10 con velocidades de transferencia de hasta 98 MB/s, y una tarjeta microSDHC Kingston de 32 GB, clase 10 con velocidades de transferencia de hasta 90 MB/s. Se dispone de dos modelos distintos porque son las tarjetas que teníamos disponibles para el proyecto. No influye en nada en concreto, sólo que la tarjeta de la marca Kingston es un poco más lenta y se nota a la hora de ejecutar alguna aplicación, descomprimir un archivo o tareas similares.

4.1.5 Sensor MPU6050

El MPU6050 es un IMU, es decir, una unidad de medición inercial (Inertial Measurement Units) de 6 grados de libertad o DoF (Degrees of Freedom). Un IMU es aquel dispositivo que permite medir e informar de velocidad, orientación y fuerzas gravitacionales usando combinaciones de acelerómetros y giroscopios. Por eso, el sensor MPU6050 posee un acelerómetro de 3 ejes y un giroscopio de 3 ejes. También incluye un sensor de temperatura.

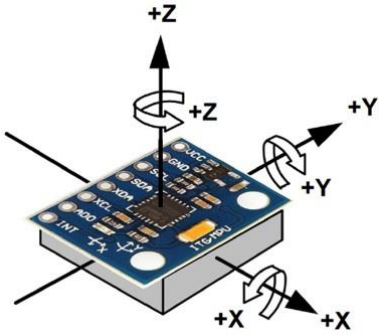


Ilustración 15: Ejes en el MPU6050 [40]

Los ejes en los que podemos medir estos datos son los ejes X, Y y Z. En el propio sensor viene impresa la dirección de estos ejes para que podamos saber los signos correctos. Lo podemos ver en la Ilustración 15 [40].

El giroscopio y acelerómetro que integra este sensor son sistemas microelectromecánicos o MEMS (MicroElectroMechanical Systems). Estos nos permiten medir la velocidad angular en grados por segundo (º/s) o revoluciones por segundo (RPS), y también la aceleración. El funcionamiento básico y muy resumido de este tipo de sistema consiste en un sensor muy pequeño, con el tamaño de un cabello humano (de 1 a 100 micrómetros), que, al girarse, una pequeña masa de resonancia se va desplazando y genera señales eléctricas muy débiles que acaban siendo amplificadas para poder ser leídas, esto es lo que se representa en la Ilustración 16 [41].

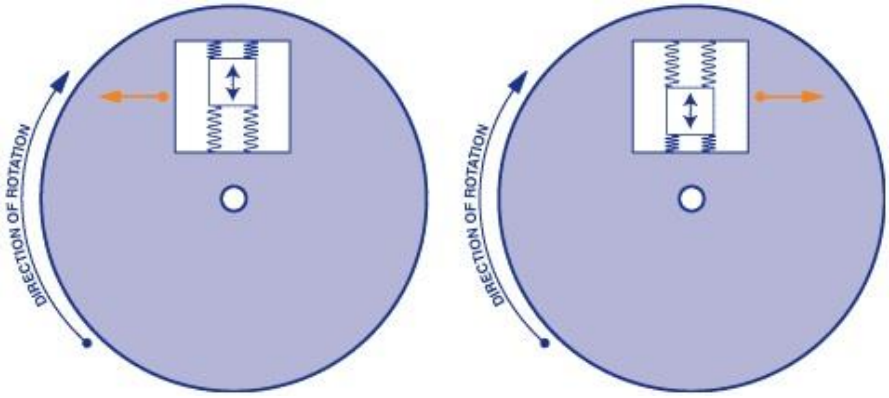


Ilustración 16: Representación gráfica de un sistema MEMS [41]

Este sensor también posee el llamado Procesador Digital de Movimiento o DMP, que permite realizar cálculos complejos en tiempo real directamente en el chip [42].

La comunicación se realiza a través de I2C, lo que le permite trabajar con una amplia variedad de dispositivos. I2C es un protocolo de comunicaciones en un bus en serie. Lo principal que tenemos que destacar es que utiliza dos líneas para transmitir los datos (SDA) y otra para la señal del reloj (SCL) [43]. Por tanto, los pines que nos interesan para conectar con los pines I2C de las Raspberry son:

- El pin VCC, que es el pin de alimentación a 5V.
- El pin GND, es el pin de tierra.
- El pin SCL, el pin de la señal de reloj I2C.
- El pin SDA, el pin de datos I2C.

Para mejorar la compresión del funcionamiento de este sensor, vamos a profundizar un poco en los términos de aceleración y velocidad angular.

4.1.5.1 Aceleración

La aceleración es básicamente la variación de la velocidad con respecto al tiempo, que matemáticamente se puede definir como la derivada de la velocidad con respecto al tiempo:

$$a=dV/dt$$

La aceleración también se define con la segunda ley de Newton. Esta indica que, en un cuerpo con masa

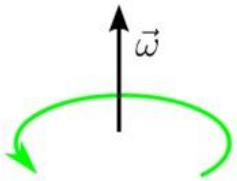
constante, la aceleración del cuerpo es proporcional a la fuerza que actúa sobre él mismo, de forma que:

$$\mathbf{a=F/m}$$

Esta segunda ley de Newton es la que usa el acelerómetro para medir la aceleración, usando los sistemas microelectromecánicos o MEMS que mencionamos anteriormente. A pesar de que no exista movimiento, el acelerómetro siempre estará sintiendo la aceleración de la gravedad. [44]

4.1.5.2 Velocidad Angular

La velocidad angular es la tasa de cambio del desplazamiento angular por unidad de tiempo, es decir, cómo de rápido gira un cuerpo alrededor de su eje:



$$\omega = \frac{d\theta}{dt}$$

Ilustración 17: Movimiento angular

Los giroscopios, al igual que los acelerómetros, también utilizan MEMS, aunque en este caso para medir la velocidad angular usando el efecto Coriolis. Este efecto hace que un objeto que se mueve sobre el radio de un disco en rotación tienda a acelerarse con respecto a ese disco según si el movimiento es hacia el eje de giro o alejándose de éste.

Las salidas del giroscopio están en grados por segundo, por lo tanto, para poder obtener la posición angular, es necesario integrar la velocidad angular. [44]

4.2 Recursos software

En este apartado vamos a explicar brevemente los recursos software que se van a usar en el proyecto.

4.2.1 NOOBS

NOOBS es un instalador de sistemas operativos para la Raspberry Pi. Nos permite instalar Raspbian OS y otros sistemas operativos de forma muy sencilla. Solo tenemos que descargar el archivo y copiarlo en nuestra tarjeta microSD de la Raspberry Pi. Los pasos y una explicación más detallada podemos verlo en el anexo 1 “Configuración inicial de cada Raspberry Pi”.

4.2.2 SD Card Formatter

Esta aplicación nos permite formatear tarjetas de memoria de forma muy sencilla. Al igual que el apartado anterior, su uso podemos verlo en el anexo 1.

4.2.3 Hadoop

Como ya sabemos, Hadoop será usado en nuestro proyecto como framework Big Data. La versión que usaremos será la 2.7.7, ya que es la última versión lanzada con compatibilidad garantizada con Pig. Tendrá que ser instalado en cada una de las Raspberry Pi y los pasos y todo lo necesario podemos verlo en el Anexo 2 "Instalación y configuración del clúster Hadoop".

4.2.4 Pig

También sabemos que Pig será utilizado para realizar las tareas MapReduce en nuestro clúster Hadoop. Se usará la versión 0.17. Su instalación y configuración podemos encontrarla en el anexo 3 "Instalación de Pig".

4.3 Diseño de la solución

En este subapartado vamos a ver cómo todos los componentes que se han mencionado han sido conectados para su correcto funcionamiento, y que resulte cómodo de usar.

Como ya hemos mencionado, se usarán cuatro Raspberry Pi. Cada Raspberry Pi necesita mímino dos elementos para funcionar, un adaptador de corriente y una tarjeta microSD. Por tanto, en cada Raspberry Pi hemos insertado una tarjeta microSD y hemos adquirido un adaptador de corriente de 5V y 3A oficial. En el apartado de "Recursos hardware" vimos que teníamos tres tarjetas microSD Samsung y una Kingson. Esto no influye en nada, sólo que la Kingston es un poco más lenta, por lo que la hemos colocado en la Raspberry Pi 2, que será un DataNode y más adelante explicaremos cuál es. Para poder enchufar todos los adaptadores de corriente también se ha adquirido una regleta de 6 tomas, las cuales serán suficientes para este proyecto.

Ya que tener cuatro Raspberry Pi sueltas sería demasiado molesto para poderlas utilizar, se ha utilizado un soporte de cuatro Raspberry Pi como el de la Ilustración 18. De esta forma las Raspberry Pi quedan ordenadas y al estar atornilladas resulta muy cómo de conectar y desconectar elementos en ellas. La Raspberry Pi de arriba del todo será la Raspberry Pi 1, que será el NameNode o "master", el resto serán la 2, 3 y 4, que serán los DataNodes o "workers" como podemos ver también en la Ilustración 18.

Raspberry Pi 1

Raspberry Pi 2

Raspberry Pi 3

Raspberry Pi 4

Ilustración 18: Raspberry Pi en soporte y numeración dada [45]

Este soporte también incluye unos extras que son cuatro ventiladores para ayudar a la refrigeración de las Raspberry Pi. Estos ventiladores se conectan de una forma muy sencilla a ellas. Cada uno posee dos cables, uno rojo y uno negro que deben ser conectados al conjunto de pines GPIO que poseen las Raspberry Pi, como podemos ver en la Ilustración 19. Por tanto, sólo hay que conectar el cable rojo al pin 4, que da 5V de corriente continua, y el cable negro al pin 6, que es el pin de tierra.

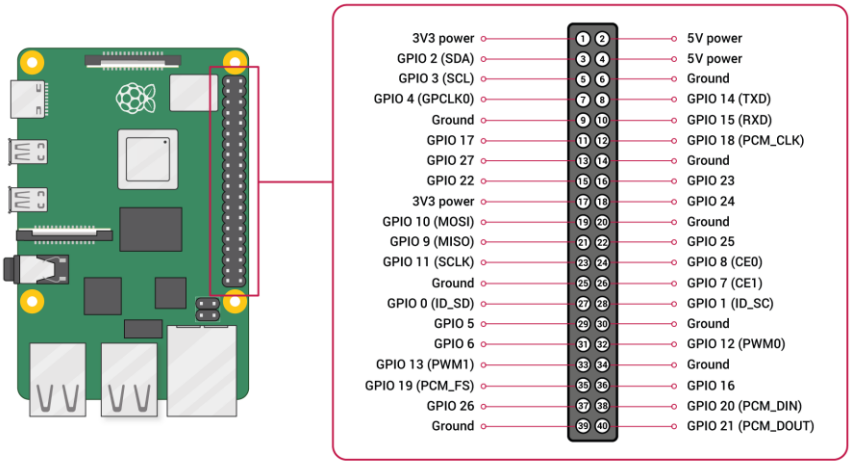


Ilustración 19: Pines GPIO en la Raspberry Pi 3 model B+ [47]

Usando la regleta anteriormente mencionada, también podemos conectar el adaptador de corriente del conmutador. El conjunto de Raspberry Pi con el soporte se ha pegado con cinta de doble cara encima del conmutador para que todos los elementos estén unidos. Para la conexión de las Raspberry entre sí, se han utilizado cuatro cables ethernet RJ45 de categoría 5 de medio metro (50 centímetros). Aparte, se ha usado otro cable ethernet RJ45 de categoría 5 de 1 metro para la conexión del ordenador portátil con el conmutador. Los cables se han enrrollado y atado para que queden de la forma más cómoda y recogida posible. El resultado final lo podemos ver en la Ilustración 20.

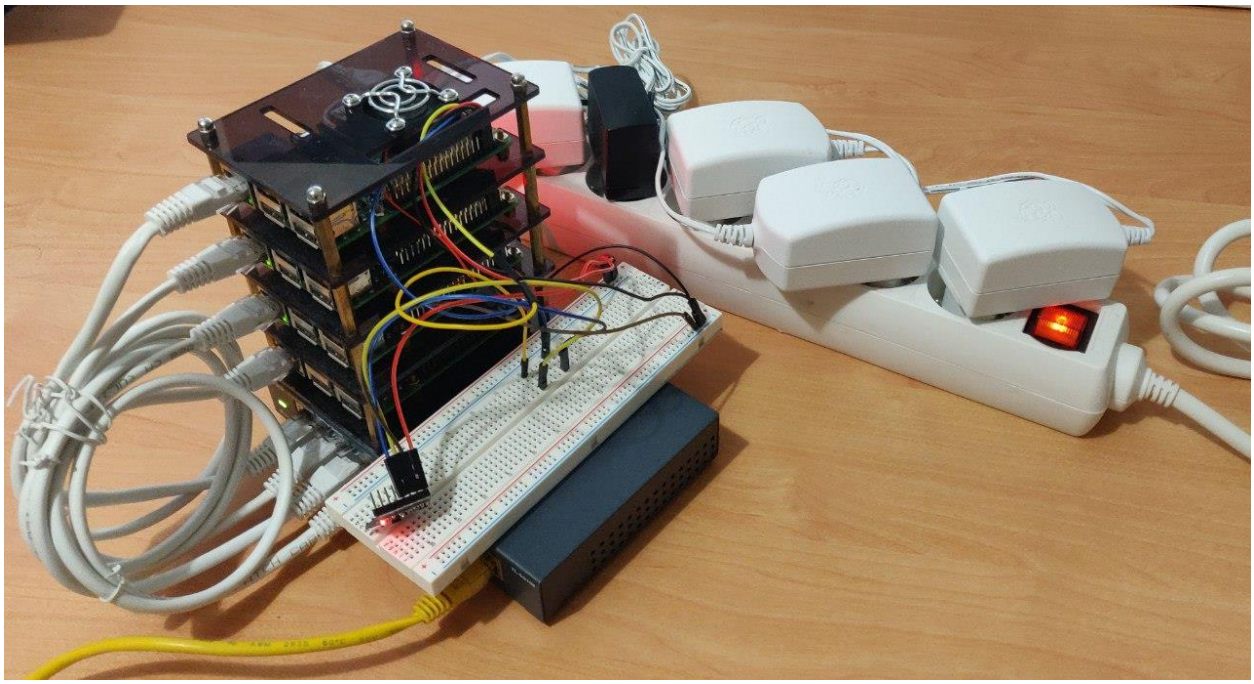


Ilustración 20: Escenario completo

Ahora sólo falta conectar el sensor MPU6050. Este sensor se conectará a "master" y dado que usará los mismos pines de alimentación y tierra que el ventilador, usaremos una "protoboard" o placa prototipo para poder conectar todo. Además, también nos permitirá mover más cómodamente el sensor para poder obtener las medidas deseadas.

La conexión es muy sencilla. Se pondrá un cable desde el pin 4 a la fila de pines positivos de la placa, y un cable desde el pin 6 a la fila de pines negativos de la placa. El pin 3 y el pin 5 son los pines SDA y SCL, respectivamente. Por lo que también se ha conectado un cable desde el pin 3 a la fila 25 de la placa, y otro cable desde el pin 5 a la fila 30 de la placa. El cable rojo del ventilador se ha conectado a la fila positiva de la placa y el negro a la fila negativa. Para el pin SDA se ha usado un color azul, y para el SCL se ha usado el amarillo. Por tanto, se ha conectado al sensor un cable rojo a su pin VCC, uno marrón (los negros se agotaron)

a su pin GND, uno amarillo al pin SCL y uno azul al SDA. Estos se han conectado a las correspondientes filas de la placa prototipo. El esquema de conexiones final lo podemos ver en la Ilustración 21.

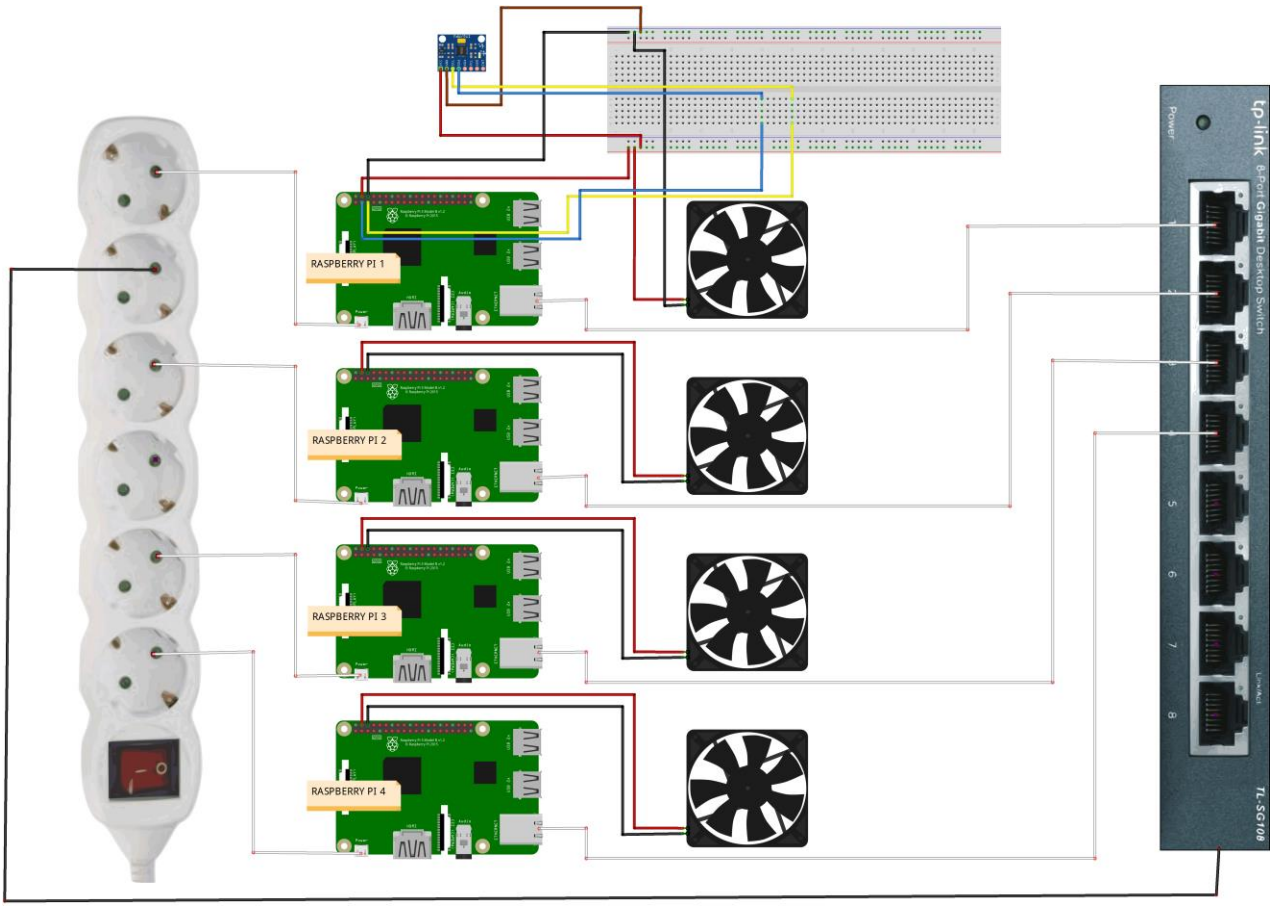


Ilustración 21: Esquema de conexiones completo del escenario planteado

5 DESARROLLO, IMPLANTACIÓN Y PRUEBA DE LA SOLUCIÓN

En este capítulo veremos la aplicación práctica realizada con el clúster de Hadoop y Apache Pig. Primero veremos cómo son los datos que se van a usar y cómo los hemos tomado y almacenado. A continuación, nos centraremos en Hadoop, cómo funciona en este clúster y algunas herramientas que nos ofrece, así como el almacenamiento de datos en él. Y, por último, veremos los operadores más importantes que podemos encontrar en Pig, utilizando un ejemplo con los datos del MPU6050 para todos ellos.

5.1 Toma y almacenamiento de datos

Como ya sabemos, los datos que usaremos en este trabajo serán los obtenidos a través del sensor MPU6050. Generaremos un fichero que almacenará todos los datos obtenidos por este sensor durante un tiempo determinado. En este caso el tiempo total en el que hemos tomado datos ha sido durante 1 hora y 15 minutos.

Estos datos serán obtenidos gracias a un script escrito en Python, que es el que se detalla en el Anexo 4 "Configuración MPU6050". Con este script capturaremos cada 0.1 segundos los datos obtenidos por el sensor, de forma que tendremos gran cantidad de datos que podrán ser analizados. En total tendremos un fichero con 40699 líneas de información, con un tamaño de 1.6 MB. Está muy lejos de los terabytes de datos que se suelen usar en Hadoop, pero nos servirá para poder realizar el trabajo. Los datos que se capturarán en este fichero de nombre "datosMPU.txt" serán los siguientes:

- El valor de los 3 ejes (X, Y y Z) capturado por el giroscopio en grados por segundo.
- El valor de los 3 ejes (X, Y y Z) capturado por el acelerómetro en metros por segundo al cuadrado (m/s^2) o fuerzas G (g).
- El valor de la temperatura en grados centígrados.

```
-0.27   -2.34   -0.22   0.01   0.97   -0.28   33.80
```

Ilustración 22: Ejemplo de datos del MPU6050

Como ejemplo, en la Ilustración 22 tenemos una línea de los datos obtenidos por el sensor. Los tres primeros valores se corresponden a los datos del giroscopio en grados por segundo. A estos valores le llamaremos "Gx", "Gy" y "Gz" para el eje x, el eje y y el eje z, respectivamente. En el momento de la medida estaba intentando mantener el sensor fijo, aunque con un poco de movimiento, por eso vemos valores cercanos al 0. Debido al ruido en las medidas, generalmente, aunque el sensor esté completamente quieto, las medidas nunca se mantendrán en 0 exacto.

Los tres valores que van a continuación del giroscopio son los valores de los tres ejes del acelerómetro que si los multipicamos por 10, obtenemos la medida en metros por segundo al cuadrado. A estos valores le llamaremos "Ax", "Ay" y "Az" para el eje x, el eje y y el eje z, respectivamente. En el momento de la medida el sensor se encontraba levantado hacia el eje y, por eso la medida en el eje y es de 9.7 m/s^2, al estar el sensor sin apenas movimiento, esta medida debe corresponderse más o menos con la de la gravedad. Sabemos que la gravedad de la tierra es 9.8 m/s^2, por tanto, podemos ver la que medida obtenida se acerca mucho a este valor.

Por último, tenemos el último valor de la fila, que se corresponde con la temperatura. Este valor está en grados centígrados, por tanto, la temperatura que registró el sensor durante ese momento fue de 33.80ºC.

Para que los datos obtenidos durante la toma de datos para el fichero "datosMPU.txt" no fuese siempre la misma, se movió el sensor aleatoriamente usando la mano para que los valores registrados fueran variando. Para la temperatura, usamos el dedo para transmitir el calor corporal al sensor, hasta que este llegaba a una temperatura de 37ºC, que es la temperatura corporal media. Una vez que se retiraba el dedo, la temperatura iba descendiendo progresivamente hasta la temperatura del ambiente donde se encontraba el sensor.

5.2 Hadoop e interfaz gráfica

Hadoop nos ofrece diversas herramientas para poder gestionar nuestra información en el clúster, poder ver el estado de los nodos, las instrucciones que se ejecutan y más información. Para gestionar los datos almacenados en HDFS tenemos el comando "hdfs fs", y para comprobar datos sobre el clúster o logs tenemos una interfaz web donde poder ver toda la información.

Una vez que tenemos iniciado nuestro clúster Hadoop como se detalla en el Anexo 2 "Instalación y configuración del clúster Hadoop", podemos comprobar los datos accediendo a la url "http://192.168.1.10/50070". Como se explicaba en el Anexo 1 "Configuración inicial de cada Raspberry Pi", un ordenador portátil estará conectado al clúster, de forma que podremos acceder a la interfaz gráfica escribiendo esa url en el navegador web. La dirección IP será la de la Raspberry Pi "master" y el puerto es el especificado en los ficheros de configuración.

Cuando accedemos a la url nos encontraremos con la página principal, que es la que podemos ver en la Ilustración 23. En ella podremos ver bastante información acerca del clúster con un solo vistazo. En la primera tabla de "Overview", podemos ver que la Raspberry Pi máster está disponible en el puerto 9000, así como la fecha, la versión (2.7.7 como se indicó anteriormente), el id del clúster y otros datos.

A continuación, tenemos la tabla "Summary", donde se resume el estado de HDFS. Encima de la tabla encontramos información como que tenemos 705 archivos y directorios creados en Hadoop y 325 bloques. Podemos ver que la capacidad total disponible entre las tres Raspberry Pi que se ejecutan como DataNodes es de 78.8 GB. Este dato viene de que cada una de ellas tiene una tarjeta microSD de 32 gigas, pero con la instalación del sistema y otros datos, entre las tres nos queda disponible esa cantidad de memoria. De esa cantidad de memoria, HDFS no usará 20.94 GB, mientras que el resto sí lo podrá usar.

En el momento de la captura tenemos que se está usando 2.46 GB de datos en HDFS y de esos 2.46GB de datos la misma cantidad se está usando en "Black Pool", es decir, los metadatos de donde se encuentran cada bloque o fichero de datos en el clúster. Esto se debe a que los datos que hay almacenados en el fichero ocuparán escasos MB, por lo que, redondeando, Hadoop muestra el mismo valor. Además, tenemos el porcentaje de memoria ocupado en cada Datanode.

También podemos comprobar que en la parte de arriba tenemos un menú que nos llevará a distintas secciones de la interfaz web. En ellas podremos ver información acerca de los DataNodes, los fallos producidos en el clúster, el estado de arranque, entre otros. Al final del menú tenemos un menú desplegable llamado "Utilities" donde podremos acceder a un buscador en el sistema de ficheros, y a los logs producidos por el clúster.

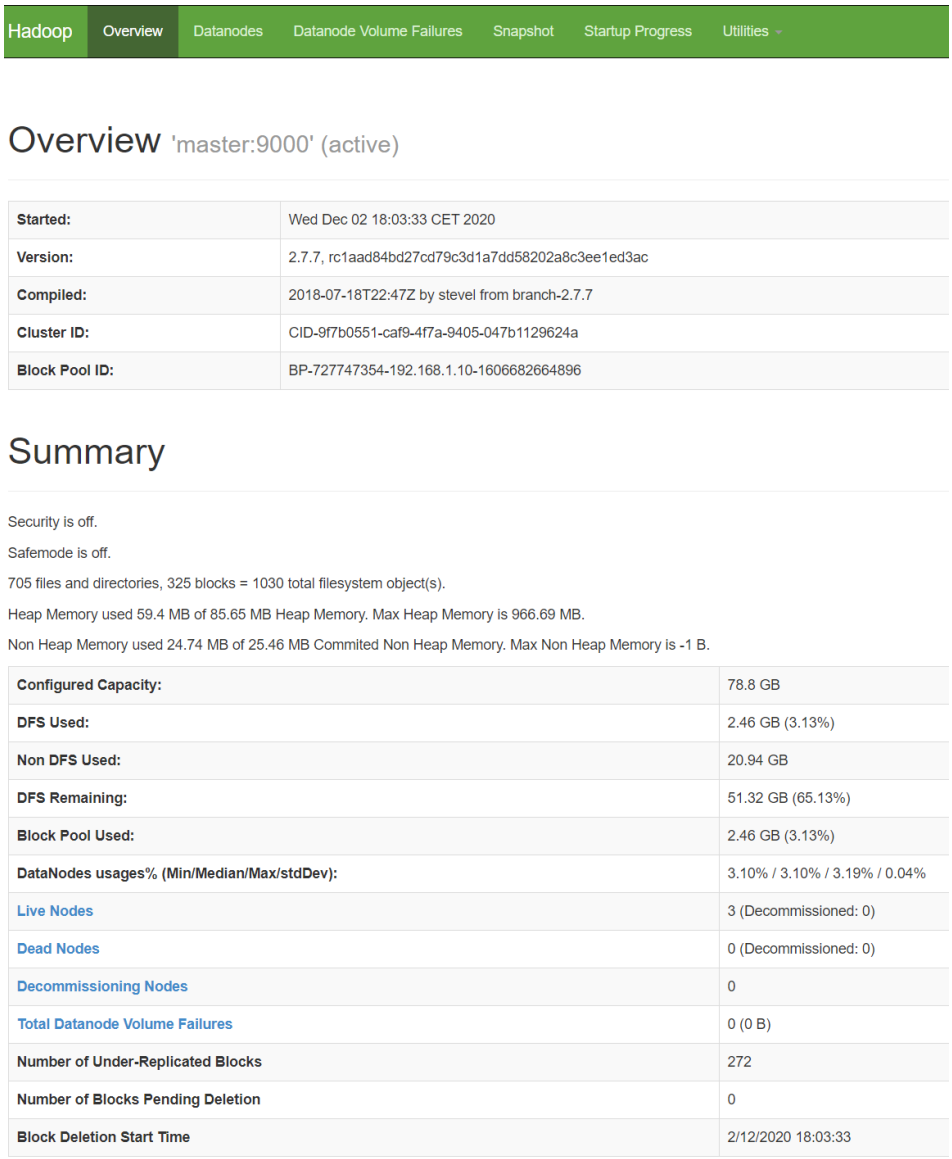


Ilustración 23: Página principal de la interfaz web de Hadoop

Si accedemos a la pestaña “Datanodes” podremos obtener información de los Datanodes del clúster como podemos ver en la Ilustración 24. En ella vemos que tenemos 3 DataNodes, llamados “worker1”, “worker2” y “worker3” como se especifica en el Anexo 3, en los puertos 50010. En la tabla tenemos la capacidad de cada uno de ellos, el espacio que HDFS está usando o no, la memoria disponible o bloques.

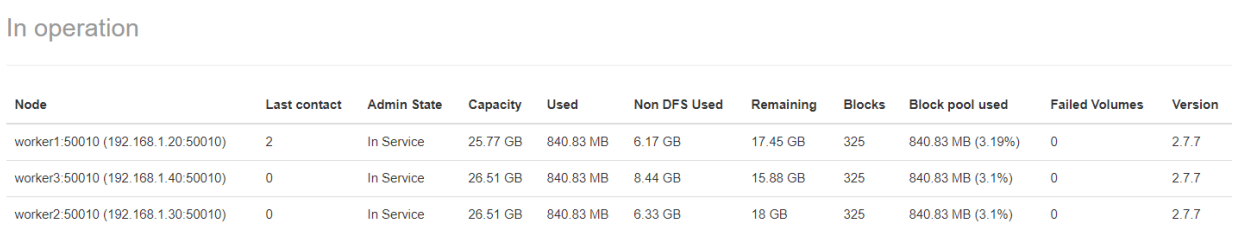


Ilustración 24: Pestaña "Datanodes" de la interfaz gráfica de Hadoop

Ahora que ya conocemos un poco mejor la interfaz web de Hadoop y los datos que podemos consultar, vamos a proceder a almacenar el fichero "datosMPU.txt", creado en el apartado anterior, en HDFS. Esto lo haremos con ayuda del comando "hadoop dfs" que mencionamos anteriormente. Este comando nos permite realizar diversas operaciones sobre HDFS. Sus comandos son muy similares a los que podemos encontrar en Linux, ya que solo tenemos que escribir "hdfs dfs -comando" y en "comando" escribir el que queramos. Por ejemplo, tenemos "ls" para ver lo que contiene un directorio, "mkdir" para crear un directorio, "cat" para imprimir un fichero, "cp" para copiar, entre otros muchos más. Esto hace que nos resulte muy familiar trabajar con HDFS.

Lo primero que haremos será crear el directorio "/datos" en HDFS, que es donde almacenaremos el fichero. Para ello solo tenemos que escribir "hdfs dfs -mkdir /datos". Con esto ya tendremos el directorio creado en HDFS. Para almacenar los ficheros resultantes de las ejecuciones de Pig como veremos más adelante, crearemos el directorio "/resultadosPig". Para ello usamos el comando "hdfs dfs -mkdir /resultadosPig".

El fichero "datosMPU.txt" se encuentra almacenado en la Raspberry Pi "master" en el directorio "/home/hduser/datosMPU.txt". Por tanto, para poder almacenar ese fichero en el directorio, tendremos que usar el comando "put" de la siguiente forma: "hdfs dfs -put /home/hduser/datosMPU.txt /datos/". Una vez lo ejecutemos, el NameNode lo replicará en los 3 DataNodes que tenemos, ya que el factor de replicación configurado es 3. Lo podemos comprobar en la Ilustración 25, donde tenemos un log del NameNode donde vemos que se va a realizar la copia del fichero en los tres DataNodes (las direcciones IP señaladas en amarillo).

```
2020-12-02 20:15:19,652 INFO org.apache.hadoop.hdfs.StateChange: BLOCK* allocate blk_1073742173_1350{UCState=UNDER_CONSTRUCTION,
truncateBlock=null, primaryNodeIndex=-1, replicas=[ReplicaUC[[DISK]DS-0b07605d-e71a-436c-a952-
16239f8e6373:NORMAL:192.168.1.30:50010|RBW], ReplicaUC[[DISK]DS-0af71a4b-1868-47e0-984a-19d5fa225046:NORMAL:192.168.1.20:50010|RBW],
ReplicaUC[[DISK]DS-adf24a61-2af6-4f07-b726-fa2dd9e11894:NORMAL:192.168.1.40:50010|RBW]]} for /datos/datosMPU.txt._COPYING_
```

Ilustración 25: Réplica de "datosMPU.txt"

5.3 Pig y procesamiento de datos

Como comentamos anteriormente, Apache Pig nos permite realizar las tareas MapReduce con unos comandos de una forma mucho más sencilla. En este apartado vamos a poner a prueba las principales operaciones que posee Pig Latin. Para ello usaremos los datos obtenidos del fichero "datosMPU.txt". Pig Latin posee numerosas operaciones, cláusulas, tipos de datos, entre otras muchas opciones. En este trabajo nos centraremos en las principales, para mostrar el funcionamiento y ver las posibilidades que nos aporta.

5.2.1 Funcionamiento de los operadores de Pig Latin

Veamos qué función tiene cada operador [47] y un ejemplo con cada uno de ellos.

5.2.1.1 LOAD

LOAD nos permite cargar datos a Pig desde el sistema de almacenamiento, que en este caso es HDFS. Su sintaxis es la siguiente:

LOAD 'datos' [USING función] [AS esquema]

En "datos" puedes escribir tanto el nombre de un fichero como el de un directorio, en el caso de introducir un directorio se almacenarán todos los datos que se encuentren en él. Opcionalmente, se pueden usar las cláusulas USING y AS:

- Con USING podemos definir una palabra clave que será la que se use como separador. Por ejemplo, podemos separar los datos por una "," o por el tabulador.
- Con AS formamos el esquema de la relación, es decir, le damos nombre a cada columna de datos y definimos el tipo de datos de esa columna.

Este comando es esencial ya que es el que nos permite cargar datos en Pig para poderlos procesar. Vamos a ver como usaríamos este comando para cargar el fichero "datosMPU.txt". Como vimos anteriormente, este fichero ya fue cargado en HDFS en el directorio "/datos/datosMPU.txt". A cada columna le asignaremmos el nombre correspondiente y especificaremos el tipo de datos, que todos serán de tipo "double". Como los datos estaban separados por el tabulador, indicaremos que se encuentran separados por "\t". La sentencia sería el siguiente:

```
datosMPU = LOAD '/datos/datosMPU.txt' USING PigStorage('\t') AS
(Gx:double,Gy:double,Gz:double,Ax:double,Ay:double,Az:double,T:double);
```

Una vez ejecutado, tendremos la relación datosMPU con todos los datos que se encontraban en el fichero "datosMPU.txt".

5.2.1.2 DUMP

DUMP es un operador de diagnóstico que nos permite mostrar una relación por pantalla. Su sintaxis es muy sencilla:

```
DUMP alias;
```

El alias será el que hemos asociado a una relación en un comando anterior. No sólo nos muestra el resultado por pantalla, también ejecuta los comandos de Pig Latin que hemos escrito. Por ejemplo, tras ejecutar el comando LOAD debemos ejecutar DUMP. De esta forma se ejecutará el comando LOAD y podremos ver el resultado por pantalla. Este comando nos permite ir comprobando la ejecución de los comandos uno a uno y ver si el resultado va siendo el esperado. Si ejecutamos "DUMP datosMPU;" podemos ver la salida como en la Ilustración 26, en la que tenemos una muestra de las tuplas que se han mostrado por pantalla.

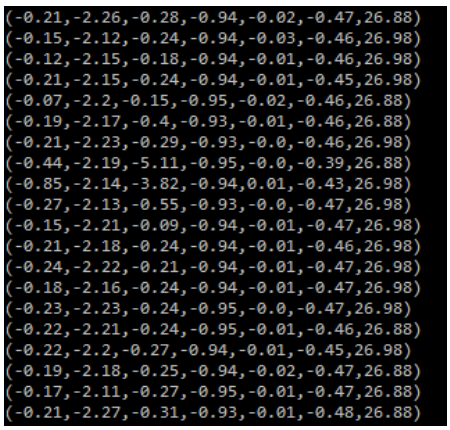


Ilustración 26: Resultado de la operación DUMP

5.2.1.3 FILTER

FILTER selecciona tuplas de una relación según una condición determinada. Su sintaxis es la siguiente:

FILTER alias BY expresión;

Se selecciona el alias de la relación que se vaya a filtrar y con la clausula BY se indica la expresión que queramos. Esta expresión será una expresión booleana. Por ejemplo, vamos a seleccionar las tuplas que tengan una temperatura de 26,88ºC. Para ello sólo tenemos escribir la sentencia:

temperaturaFiltrada = FILTER datosMPU BY T==26.88;

Tras ejecutar el correspondiente "DUMP temperaturaFiltrada" podemos ver una muestra del resultado en la Ilustración 27, ahora sólo vemos las tuplas con ese valor de temperatura (última columna).

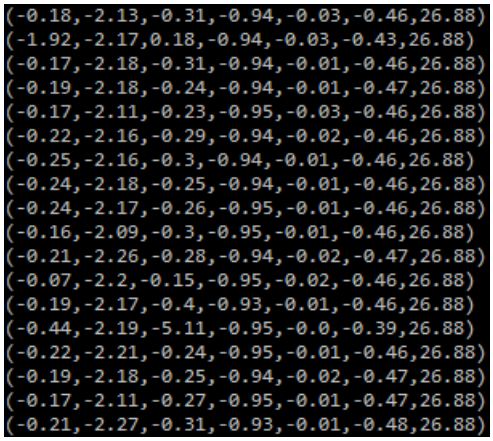


Ilustración 27: Resultado de la operación FILTER

5.2.1.4 STORE

STORE almacena los resultados en el sistema de ficheros, en este caso HDFS, de manera que no se pierdan los resultados que consideremos importantes. Su sintaxis es la siguiente:

STORE alias INTO 'directorio' [USING función];

Se selecciona el alias de la relación que queramos almacenar, y se escribe el nombre del directorio. Si ese directorio ya existe la operación fallará. Opcionalmente se puede usar USING para especificar el separador con el que se desean guardar los datos. Por ejemplo, si deseamos guardar la relación anterior con alias "temperaturaFiltrada", podemos escribir la siguiente sentencia:

STORE temperaturaFiltrada INTO '/resultadosPig/filter/' USING PigStorage(',');

Con esta sentencia almacenaremos el resultado de temperaturaFiltrada en el directorio "/resultadosPig/filter" de HDFS y separando los valores con una ",".

5.2.1.5 DISTINCT

DISTINCT elimina las tuplas duplicadas en una relación, pero no conserva el orden original de los datos, ya que para realizar esta operación primero tiene que ordenar los datos. Su sintaxis es la siguiente:

DISTINCT alias [PARTITION BY particionador] [PARALLEL n];

Primero se indica cual es el alias de la relación que se va a usar y después podemos usar PARTITION BY y PARALLEL:

- PARTITION BY especifica el "Hadoop Partitioner". Esto permite controlar la partición de las claves de las salidas de las salidas intermedias de la fase map. Estas claves se usan para derivar la partición, normalmente mediante una función hash. Cada salida se va a particionar y los pares clave-valor con la misma clave van a la misma partición, y cada partición se envía al reduce. El número de particiones va a ser el mismo que el número de tareas reduce, por lo que el particionador va a controlar cuál de las m tareas reduce se va a enviar la clave intermedia para su reducción.
- PARALLEL: Nos permite determinar el número de tareas reduce de MapReduce.

Por ejemplo para eliminar las tuplas repetidas de la relación "datosMPU" sólo tenemos que escribir la sentencia:

eliminaRepetidas = DISTINCT datosMPU;

El resultado lo podemos ver en la Ilustración 28, en la que vemos una muestra del resultado tras ejecutar "DUMP eliminaRepetidas". En ella vemos que no hay tuplas repetidas y que están ordenadas según la primera columna.

```
(-39.08,-3.66,19.2,-0.9,0.15,0.43,26.98)
(-39.16,-4.11,-4.43,0.06,-0.67,-0.85,32.53)
(-39.86,-2.64,26.55,0.14,-0.47,0.56,31.68)
(-40.82,-2.15,1.1,-0.88,-0.48,0.26,32.34)
(-42.47,-3.11,28.56,0.25,-0.27,0.6,31.87)
(-42.61,-2.95,14.95,-0.44,0.28,0.63,32.25)
(-42.71,-3.35,8.73,-1.74,1.83,0.12,32.62)
(-42.82,-2.64,-20.95,-0.51,2.0,-0.37,32.62)
(-46.78,-1.9,0.89,-0.66,0.61,0.52,30.55)
(-46.81,-2.05,21.21,-0.04,-0.39,0.51,32.15)
(-46.93,-4.11,12.07,-1.92,1.29,0.05,32.53)
(-48.61,-0.24,29.47,-0.1,-0.43,-0.39,32.44)
(-49.85,-4.79,24.83,-0.94,-0.66,-0.16,27.07)
(-59.4,-4.44,-5.4,-1.17,0.93,0.24,32.53)
(-61.44,-3.98,-2.36,-1.18,0.53,-0.03,32.53)
(-64.43,-4.45,-15.95,-0.57,-0.22,-0.3,32.62)
(-64.54,-4.63,-14.57,-1.65,0.68,0.49,32.62)
(-69.49,-5.13,-20.34,-1.69,-0.01,-0.08,32.53)
```

Ilustración 28: Resultado de la operación DISTINCT

5.2.1.6 SAMPLE

SAMPLE toma una muestra aleatoria de datos según un tamaño especificado. La sintaxis es la siguiente:

SAMPLE alias tamaño;

Solo hay que indicar el alias de la relación de la que se quiere tomar la muestra y el tamaño de la muestra. El tamaño puede ser una constante de 0 a 1, la cual indicará un porcentaje (1 es el 100%). Por ejemplo, para obtener una muestra que sea el 0.01% de todos los datos en "datosMPU" tenemos que escribir la sentencia:

muestra = SAMPLE datosMPU 0.0001;

Y una vez ejecutemos la sentencia con "DUMP muestra" podemos ver el resultado en la Ilustración 29, en la que sólo tenemos 5 tuplas.

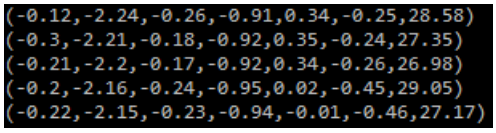


Ilustración 29: Resultado de la operación DUMP

5.2.1.7 ASSERT

ASSERT comprueba si una condición se cumple o no en una relación. La sintaxis es la siguiente:

ASSERT alias BY expresión [, mensaje];

Se indica el alias de la relación que se va a usar, la expresión que se va a comprobar y opcionalmente se puede indicar el mensaje que se mostrará en el caso de que la condición no se cumpla. Este operador falla si se incumple la condición. Por ejemplo vamos a comprobar en nuestra relación "datosMPU" que todas las tuplas tengan una temperatura inferior a 38ºC. Para ellos usamos el comando:

ASSERT datosMPU BY T<38, 'Hay tuplas con una temperatura mayor de 38';

Después de introducir el comando, lo ejecutamos con "DUMP datosMPU;" y veremos que la ejecución no da ningún error. En cambio, si queremos comprobar que todas las tuplas tengan una temperatura inferior a 37ºC, lo cual sabemos que no es así ya que hay tuplas que superan esa temperatura, podemos introducir la sentencia:

ASSERT datosMPU BY T<37, 'Hay tuplas con una temperatura mayor de 37';

Al ejecutarla con "DUMP datosMPU" veremos que la ejecución falla y nos muestra el mensaje que hemos escrito, ya que sí hay tuplas con una temperatura mayor de 37ºC.

```
.backend.executionengine.ExecException: ERROR 2078: Caught error from UDF: org.apache.pig.builtin.Assert
[Assertion violated: Hay tuplas con una temperatura mayor de 37]
```

Ilustración 30: Fallo de la operación ASSERT

5.2.1.8 LIMIT

LIMIT limita el número de tuplas a la salida. Su sintaxis es la siguiente:

LIMIT alias n;

Con "n" especificamos el número de tuplas que se mostrarán a la salida. Si introducimos un número mayor que el de la cantidad de tuplas de nuestra relación, todas las tuplas serán mostradas. No se garantiza cuáles serán las tuplas mostradas, en cada ejecución pueden ser diferentes. Por ejemplo, para mostrar seis tuplas de datosMPU, solo tenemos que introducir la siguiente sentencia:

limitado = LIMIT datosMPU 6;

Tras ejecutarla con "DUMP limitado", podemos ver la salida en la Ilustración 31, que como era de esperar, son 6 tuplas.

```
(-0.21,-2.19,-0.7,-0.97,0.04,-0.4,31.12)
(-0.29,-2.15,-0.45,-0.96,0.05,-0.39,31.12)
(-0.3,-2.27,-0.53,-0.96,0.03,-0.4,31.12)
(0.22,-2.33,0.11,-0.95,0.04,-0.39,31.02)
(-0.34,-2.22,-0.18,-0.96,0.05,-0.39,31.02)
(-0.49,-2.2,-0.41,-0.93,0.08,-0.41,31.12)
```

Ilustración 31: Resultado de la operación LIMIT

5.2.1.9 CROSS

Realiza el producto cartesiano de dos o más relaciones. Su sintaxis es la siguiente:

CROSS alias, alias [alias …] [PARTITION BY particionador] [PARALLEL n];

Como se puede ver, solo hay que indicar los alias de las relaciones con las que queremos realizar el producto cartesiano. También se pueden usar las cláusulas PARTITION BY y PARALLEL que ya indicamos para qué servían. Por tanto, vamos a realizar el producto cartesiano entre la relación "limitado" que creamos anteriormente y una nueva relación que vamos a obtener de un nuevo fichero que vamos a subir a Hadoop. Ese fichero va a contener únicamente dos tuplas que serán "A" y "B". Una vez almacenado ese fichero en Hadoop, lo usaremos en Hadoop en una relación que llamaremos "cruzado". Para realizar el producto cartesiano entre las dos relaciones tendremos que insertar la siguiente sentencia:

productoCruzado = CROSS limitado, cruzado;

Y como podemos ver en la Ilustración 32, el resultado de “¨DUMP productoCruzado”; es el producto cartesiano de las dos relaciones.

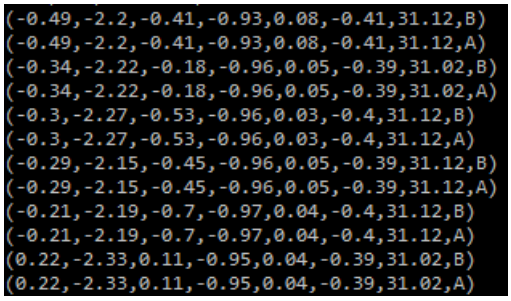


Ilustración 32: Resultado de la operación CROSS

5.2.1.10 SPLIT

Divide una relación en dos o más relaciones según las condiciones que queramos. Su sintaxis es la siguiente:

SPLIT alias1 INTO alias2 IF expresión [, alias3 IF expresión …];

Primero especificamos el alias de la relación que vamos a definir. Después, con la expresión INTO indicamos los nuevos alias donde queremos guardar las nuevas relaciones, y con IF indicamos la condición para dividir la relación. Por ejemplo, vamos a dividir nuestra relación datosMPU en dos relaciones, una en la que todas las tuplas tengan una temperatura menor de 27 grados y otra en la que todas las tuplas tengan una temperatura mayor de de 37 grados. Para ello usaremos la sentencia:

SPLIT datosMPU INTO temperaturaMenor IF T<27, temperaturaMayor IF T>37;

Para ver ejecutar el comando y ver los resultados usamos el comando DUMP con los nuevos alias creados. Así, si ejecutamos “DUMP temperaturaMenor;” tenemos una muestra del resultado en la Ilustración 33, donde podemos ver que todas las tuplas tienen una temperatura menor de 27 grados.

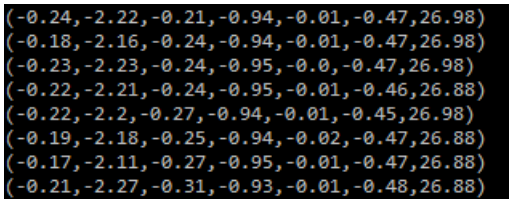


Ilustración 33: Primer resultado de la operación DUMP

En cambio, si ejecutamos “DUMP temperaturaMayor;”, podemos ver en la Ilustración 34 que todas las tuplas tienen una temperatura mayor de 37 grados.

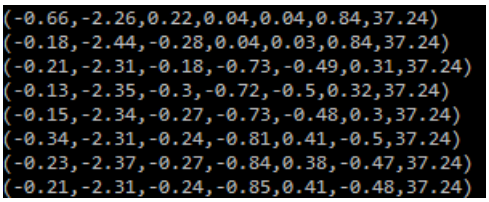


Ilustración 34: Segundo resultado de la operación DUMP

5.2.1.11 GROUP

GROUP agrupa los datos en una o más relaciones. Existe COGROUP que es idéntico a GROUP, ambos soportan una o más relaciones, solo que por legibilidad GROUP se usa para una relación y COGROUP para dos o más relaciones. COGROUP permite hasta 127 relaciones a la vez. La sintaxis, tanto para GROUP como COGROUP es la siguiente:

GROUP alias ALL | BY expresión [, alias ALL | BY expresión] [USING 'collected' | 'merge'] [PARTITION BY particionador] [PARALLEL n];

Primero se indica el alias de la relación que queramos. A continuación, podemos usar ALL si queremos que todas las tuplas se agrupen en un único grupo, o BY para agrupar la relación por campos, tuplas o expresiones. Si hemos usado BY tenemos que indicar la expresión a continuación. Después, podemos indicar nuevos alias con sus respectivos ALL o BY. Además, podemos usar las cláusulas USING, PARTITION BY y PARALLEL. Con USING podemos indicar las cláusulas collected y merge:

- Collected: Se usa con la operación GROUP y para una sola relación. Debe cumplir una serie de condiciones y si se cumplen la operación se ejecutará en la fase map, evitando la fase reduce.
- Merge: Se usa con la operación COGROUP y para dos o más relaciones. También se deben cumplir una serie de condiciones, y al igual que collected, si se cumplen, la operación se ejecutará en la fase map, evitando la fase reduce.

Por ejemplo, para agrupar los datos de la relación "datosMPU" según la temperatura, tendríamos que introducir la siguiente sentencia:

agrupado = GROUP datosMPU BY T;

Como podemos ver, tras ejecutar "DUMP agrupado" en la muestra de la Ilustración 35, tenemos en una tupla, la temperatura, y todas las tuplas de la relación que tenían esa temperatura.

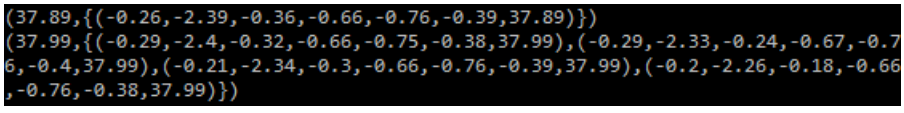


Ilustración 35: Resultado de la operación GROUP

5.2.1.12 FOREACH

FOREACH genera transformaciones de datos basadas en columnas de datos. Su sintaxis es la siguiente:

FOREACH {bloque | bloque_anidado}

Para usar el operador FOREACH necesitamos un bloque o un bloque anidado. Vamos a ver qué son.

- Bloque: Se usa una relación, por lo que se tiene la siguiente sintaxis:

FOREACH alias GENERATE expresión [AS esquema] [expresión [AS esquema] …];

Primeramente se indica el alias de la relación sobre la que vamos a realizar la operación, y con GENERATE podemos indicar los nombres de las columnas de la relación sobre las que queremos realizar la operación o una expresión, y con AS podemos indicar un nuevo nombre para esas columnas (esquema).

Un ejemplo muy sencillo sería formar una nueva relación con sólo las columnas Gx, Ax y T. Para ello, habría que introducir la siguiente sentencia:

foreachBloque = FOREACH datosMPU GENERATE Gx, Ax, T;

Como podemos ver en la muestra de la Ilustración 36, tras ejecutar "DUMP foreachBloque", esto nos devuelve la relación con estas tres columnas como queríamos.

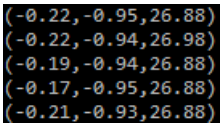


Ilustración 36: Resultado de la operación FOREACH con bloque

- Bloque anidado: se anida una relación en otra, esto lo podemos ver en su sintaxis:

FOREACH alias_anidado {
alias = {operación_anidada | expresión _anidada}; [alias = {operación_anidada | expresión _anidada}; …]
GENERATE expresión [AS esquema] [expresión [AS esquema] ….]
};

Se indica el alias de la relación sobre la que vamos a realizar la operación, y después, entre llaves, se indican nuevos alias que pueden tener las operaciones o expresiones que se deseen, y por último se inserta el GENERATE siguiendo las mismas pautas como vimos en el bloque. Podemos usar la relación "agrupado" que creamos en el apartado anterior y usar el FOREACH anidado para generar una relación, cuyas tuplas sean la temperatura, y la otra columna sólo los ejes Gz y Gy del giroscopio, y Az y Ay del acelerómetro, de las tuplas que tengan ese valor de temperatura. Para ello usaremos la sentencia:

foreachAnidado = FOREACH argupado{
anidado = FOREACH datosMPU GENERATE Gy, Gz, Ay, Az;
GENERATE group, anidado;}

El nombre group es el nombre que le da el comando GROUP a la columna que se ha usado para agrupar al resto de tuplas. Como podemos ver en la Ilustración 37, tras ejecutar "DUMP foreachAnidad", ahora cada tupla tiene la temperatura, y todas las tuplas que tenían esa temperatura, pero sólo almacenando Gy, Gz, Ay y Az.

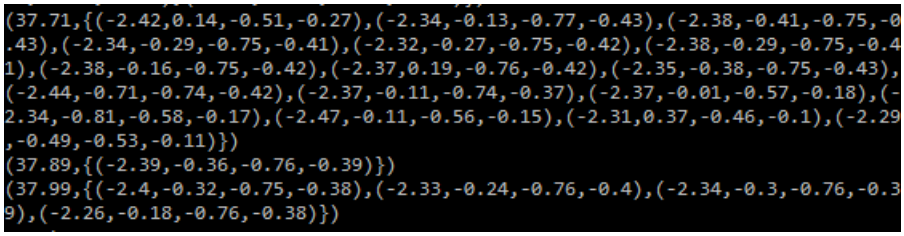


Ilustración 37: Resultado de la operación FOREACH anidado

5.2.1.13 JOIN

En Pig Latin disponemos de dos tipos de JOIN, el JOIN (inner) y el JOIN (outer). Vamos a verlos de forma separada.

- JOIN (inner) realiza un inner join entre dos o más relaciones basándose en unos valores de campo comunes. Consiste en combinar cada tupla de una relación con cada tupla de la otra relación, seleccionando las que cumplan una determinada condición y eliminando el resto. Su sintaxis es la siguiente:

JOIN alias BY expresión [, expresión…] [, alias BY expresión] [USING 'replicated' | 'bloom' | 'skewed' | 'merge' | 'merge-sparse'] [PARTITION BY particionador] [PARALLEL n];

Primeramente se escribe el alias de la relación que queramos y con BY indicamos la expresión o expresiones, separando con "," podemos añadir más alias y más expresiones. Adicionalmente, podemos usar USING para indicar unos tipos especiales de JOIN, con los que no entraremos en detalle, llamados replicated, bloom, skewed, merge y merge-space. También podemos seleccionar usar las cláusulas PARTITION BY y PARALLEL. Por ejemplo vamos a utilizar la relación "foreachBloque" que ya teníamos creada con Gx, Ax y T, y vamos a limitarla a 100 tuplas con LIMIT, almacenando el resultado en la relación "limitado1". Por otra parte, vamos a crear otra relación similar a "foreachBloque" pero con Gy, Ay y T, la cual limitaremos también a 100 tuplas con LIMIT en la relación "limitado 2". Con estas dos nuevas relaciones, vamos a hacer el inner JOIN, para ello usamos la sentencia:

joinInner = JOIN limitado1 BY T, limitado2 BY T;

Como podemos ver en la Ilustración 38, tras ejecutar "joinInner", como resultado tenemos que la operación de JOIN se ha realizado en las dos tuplas según la temperatura.

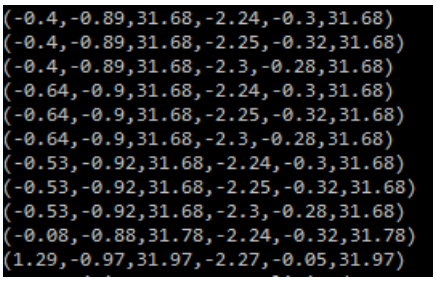


Ilustración 38: Resultado de la operación (inner)

- JOIN (outer) realiza un outer join entre dos relaciones basándose en unos valores de campo comunes. Es como el inner join pero también muestra los datos de la primera relación (izquierda), de la segunda (derecha) o de las dos (full). Su sintaxis es la siguiente:

JOIN alias-izquierdo BY columna-alias-izquierdo [LEFT|RIGHT|FULL] [OUTER], alias-derecho BY columna-alias-derecho [USING 'replicated' | 'bloom' | 'skewed' | 'merge'] [PARTITION BY particionador] [PARALLEL n];

En este caso tenemos que especificar primero el alias de la primera relación, que será la de la izquierda, y con BY indicamos el nombre de la columna. Después indicamos el alias de la segunda relación, que será la de la derecha, y con BY seleccionamos la columna también. Seguidamente indicamos qué relación va a mantener sus tuplas, si la izquierda (left), la derecha (right) o las dos (full). También podemos usar PARTITION BY y PARALLEL. Para mostrar un ejemplo, vamos a usar las mismas relaciones de antes, pero la relación "limitado2" en vez de estar limitada a 100 tuplas, vamos a limitarla a 150, y la llamaremos "limitado3". Usaremos un right outer JOIN con la siguiente sentencia:

joinOuter = JOIN limitado1 BY T RIGHT, limitado3 BY T;

En la Ilustración 39 podemos ver el resultado tras ejecutar "DUMP joinOuter;", en el que podemos ver que ahora hay datos vacíos que se corresponden a la tupla de la izquierda (limitado1). "limitado3" tiene tuplas con información que no se encuentra en "limitado1", no sólo puede hacer el JOIN con datos vacíos.

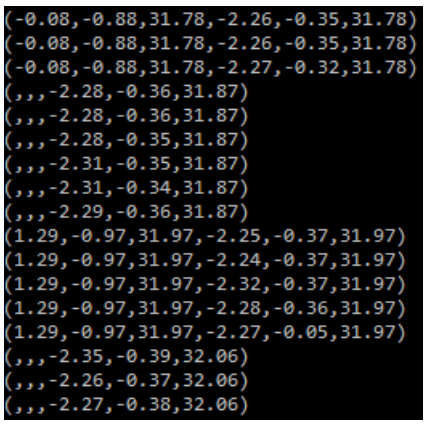


Ilustración 39: Resultado de la operación JOIN (outer)

5.2.1.14 CUBE

CUBE permite realizar dos tipos de operaciones, CUBE y ROLLUP. Ahora veremos a detalle que hace cada operación, pero primero veamos su sintaxis, que es la siguiente:

CUBE alias BY {CUBE expresión | ROLLUP expresión}, [CUBE expresión | ROLLUP expresión] [PARALLEL n];

Primero escribimos el alias sobre el que queremos realizar la operación, y seguidamente con BY, elegimos la operación CUBE o ROLLUP y la expresión que deseemos.

- CUBE: Consiste en calcular agregados para todas las combinaciones posibles de grupos específicos por dimensiones, de manera que el número de combinaciones de grupos generados por CUBE para n dimensiones será 2^n. Esto quiere decir que va a agrupar las tuplas según las columnas que le digamos. Por ejemplo si le indicamos dos columnas, mostrará en una tupla las tuplas que coincidan con dos valores específicos, en otra tupla con el primero específico y el segundo cualquiera, en otra el primero cualquiera y el segundo específico, y otra en la que los dos sean cualquier valor. Por eso, de dos valores que se indicaban (dimesión 2) surgen 2^2=4 tuplas. De nuestra relación "datosMPU" vamos a agrupar las tuplas que tengan el mismo eje x de giroscopio (Gx), eje x de acelerómetro (Ax) y temperatura. Para ello podemos usar la sentencia:

  cube = CUBE datosMPU BY CUBE(Gx, Ax, T);

  En la Ilustración 40, tras ejecutar "DUMP cube;" podemos ver una pequeña muestra del resultado, donde podemos ver que se han agrupado las tuplas que tienen unos valores específicos de los ejes Gx y Ax y de temperatura, ya que las tuplas que contienen cualquier valor de uno o más de esos parámetros, son tuplas demasiado largas que no se podrían insertar en la imagen.

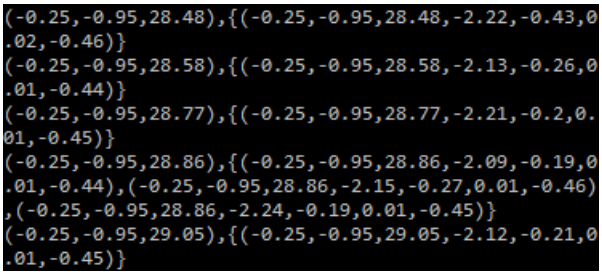


Ilustración 40: Resultado de la operación CUBE

- ROLLUP: Calcula varios niveles de agregados según el orden jerárquico del grupo especificado por dimensiones. El resumen es útil cuando hay un orden jerárquico en las dimensiones. El número de agrupaciones por combinaciones generadas por el resumen para n dimensiones será n+1. Es decir, es similar a CUBE, pero es de forma jerárquica. Por ejemplo si indicamos que agrupe por tres parámetros, lo hará primero con 3 específicos, otra tupla con los dos primeros específicos y el último cualquiera, otra con el primero específico y los dos últimos cualquiera, y otra con los 3 cualquiera. Por eso, de 3 parámetros (dimension 3) obtenemos 3+1=4 tuplas. Al igual que antes, vamos a usar esta operación con los valores de los ejes x y la temperatura. Para ello usamos la siguiente sentencia:

```
cube = CUBE datosMPU BY ROLLUP(Gx, Ax, T);
```

- Como podemos ver en la muestra de la Ilustración 41, primeramente tenemos tuplas donde los 3 valores son específicos, y después tenemos una en la que los dos primeros son específicos, y el tercero uno cualquiera.

```
(-0.22,-0.94,29.61),{(-0.22,-0.94,29.61,-2.25,-0.29,0
.03,-0.45)}
(-0.22,-0.94,29.99),{(-0.22,-0.94,29.99,-2.23,-0.24,0
.03,-0.44)}
(-0.22,-0.94,30.08),{(-0.22,-0.94,30.08,-2.23,-0.29,0
.04,-0.45)}
(-0.22,-0.94,31.68),{(-0.22,-0.94,31.68,-2.22,-0.58,0
.18,0.03)}
(-0.22,-0.94,),{(-0.22,-0.94,,-2.14,-0.24,-0.01,-0.44
),(-0.22,-0.94,,-2.18,-0.29,-0.01,-0.46),(-0.22,-0.94
,,-2.15,-0.2,-0.01,-0.46),(-0.22,-0.94,,-2.15,-0.23,-
0.01,-0.47),(-0.22,-0.94,,-2.2,-0.22,0.0,-0.45),(-0.2
```

Ilustración 41: Resultado de la operación CUBE con ROLLUP

5.2.1.15 ORDER BY

Ordena una relación basándose en uno o más campos. Su sintaxis es la siguiente:

```
ORDER alias BY nombre_columna [ASC|DESC] [, nombre_columna [ASC|DESC] …] [PARALLEL n];
```

Sólo hay que indicar el alias de la relación sobre la que queremos realizar la operación y con BY indicamos el nombre de la columna y si queremos orden ascendente (ASC) o descendente (DESC). También podemos usar la clausula PARALLEL. Por ejemplo vamos a ordenar por el eje x del girospio en orden descendente, para ello sólo tenemos que usar la sentencia:

```
ordenado = ORDER datosMPU BY Gx DESC;
```

Como podemos ver en la Ilustración 42, tras ejecutar “DUMP ordenado;” las tuplas están ordenadas según la primera columna (Gx) y en orden descendente.

```
(-46.93,-4.11,12.07,-1.92,1.29,0.05,32.53)
(-48.61,-0.24,29.47,-0.1,-0.43,-0.39,32.44)
(-49.85,-4.79,24.83,-0.94,-0.66,-0.16,27.07)
(-59.4,-4.44,-5.4,-1.17,0.93,0.24,32.53)
(-61.44,-3.98,-2.36,-1.18,0.53,-0.03,32.53)
(-64.43,-4.45,-15.95,-0.57,-0.22,-0.3,32.62)
(-64.54,-4.63,-14.57,-1.65,0.68,0.49,32.62)
(-69.49,-5.13,-20.34,-1.69,-0.01,-0.08,32.53)
```

Ilustración 42: Resultado de la operación ORDER BY

5.2.1.16 RANK

RANK asocia a cada tupla un rango dentro de una relación, es decir, usa cada campo o campos para ordenar la relación. El rango de una tupla es uno más el número de diferentes valores de rango que lo preceden. Si dos o más tuplas empatan reciben el mismo rango. Si no se especifica ningún campo para ordenar, se añade un valor secuencial a cada tupla. Su sintaxis es la siguiente:

RANK alias [BY { nombre_columna [ASC|DESC] [, nombre_columna [ASC|DESC] …] } [DENSE]];

Primeramente se indica el alias de la relación sobre la que se quiere realizar la operación. Con BY indicamos el nombre de la columna de la relación, y podemos añadir ASC o DESC para ordenar de forma ascendente o descendente, respectivamente. DENSE se puede usar para que los empates no causen espacios en los valores de clasificación, es decir, sin DENSE cada tupla cuenta un número, si añadimos DENSE, cada tupla repetida no se cuenta. Por ejemplo, vamos a usar RANK con la relación "limitado" que teníamos creada anteriormente, y vamos a generar el rango según el eje y (Ay) del acelerómetro. Para ello podemos usar la siguiente sentencia:

rango = RANK limitado BY Ay DESC;

El resultado lo podemos ver en la Ilustración 43 tras ejecutar "DUMP rango;". En ella podemos ver como las tuplas que empatan reciben el mismo rango, y que las tuplas repetidas también cuentan, por ello podemos ver que la sexta tupla tiene asignado como rango el número 6.

```
(1,-0.49,-2.2,-0.41,-0.93,0.08,-0.41,31.12)
(2,-0.34,-2.22,-0.18,-0.96,0.05,-0.39,31.02)
(2,-0.29,-2.15,-0.45,-0.96,0.05,-0.39,31.12)
(4,-0.21,-2.19,-0.7,-0.97,0.04,-0.4,31.12)
(4,0.22,-2.33,0.11,-0.95,0.04,-0.39,31.02)
(6,-0.3,-2.27,-0.53,-0.96,0.03,-0.4,31.12)
```

Ilustración 43: Resultado de la operación RANK sin DENSE

En cambio, si usamos DENSE, podremos comprobar como las tuplas repetidas no cuentan, esto lo podemos ver en la Ilustración 44, donde la sexta tupla tiene como rango el número 4. Esto es el resultado de la sentencia:

rangoDense = RANK limitado BY Ay DESC DENSE;

```
(1,-0.49,-2.2,-0.41,-0.93,0.08,-0.41,31.12)
(2,-0.34,-2.22,-0.18,-0.96,0.05,-0.39,31.02)
(2,-0.29,-2.15,-0.45,-0.96,0.05,-0.39,31.12)
(3,-0.21,-2.19,-0.7,-0.97,0.04,-0.4,31.12)
(3,0.22,-2.33,0.11,-0.95,0.04,-0.39,31.02)
(4,-0.3,-2.27,-0.53,-0.96,0.03,-0.4,31.12)
```

Ilustración 44: Resultado de la operación RANK con DENSE

5.2.1.17 UNION

UNION une dos o más relaciones. Destacar que no conserva el orden de las tuplas, no se encarga de asegurar que las tuplas de las relaciones tengan el mismo esquema o número de campos y que no elimina las tuplas duplicadas. Su sintaxis es la siguiente:

UNION [ONSCHEMA] alias, alias [, alias …] [PARALLEL n];

Sólo hay que indicar los alias de las tuplas con las que queremos realizar la operación. Podemos usar las cláusulas ONSCHEMA y PARALLEL. ONSCHEMA nos permite basar la unión por el nombre de las columnas, y no por la posición, es decir, si tenemos dos tablas no va a asumir que la primera columna de la primera tabla se corresponde con la primera conlumna de la segunda tabla, sino que primero va a comprobar el nombre, y unirá las que tengan el mismo. Para probar esta operación vamos a unir las relaciones "limitado" y "muestra" que teníamos creadas anteriormente, por tanto, usaremos la siguiente sentencia:

unido = UNION limitado, muestra;

Como podemos ver en la Ilustración 45, el resultado es la unión de las dos relaciones que habíamos mencionado anteriormente.

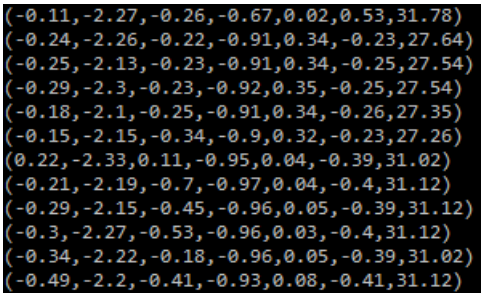


Ilustración 45: Resultado de la operación UNION

5.2.1.18 DEFINE

DEFINE nos permite asignar alias a UDFs, comandos o macros. Recordemos que las UDF eran las funciones definidas por el usuario como ya pudimos ver en el Capítulo 3. Como veremos en el operador STREAM, podemos usar comandos de linux en nuestras sentencias de Pig Latin, por tanto con DEFINE podemos asignarle un alias a estos comandos para no tener que escribir el comando completo cada vez que lo vayamos a usar. La sintaxis de DEFINE en estos dos casos es la siguiente:

DEFINE alias {función | [`comando` [input] [output] [ship] [cache] [stderr]] };

Primero indicamos el alias que le vamos a dar a la UDF o al comando. A continuación, podemos escribir el nombre de la UDF o el nombre del comando que queramos y con las cláusulas input, output, cache y stderr podemos especificar la salida del comando. Por ejemplo vamos a crear un alias para el comando "cat -n". Este comando imprime un fichero escribiendo en cada línea el número de línea correspondiente. Por tanto, se ha usado la siguiente sentencia:

DEFINE cat `cat -n`;

Ahora cada vez que queramos hacer uso del comando “cat -n” solo tendremos que escribir “cat”

También tenemos las macros. Las macros son muy parecidas a las funciones que conocemos de otros lenguajes de programación. Nos permiten definer unos parámetros de entrada, y luego usar esos parámetros en sentencias en el interior de la macro. La sintaxis en este caso es la siguiente:

```
DEFINE nombre_macro (parámetro [, parámetro ...]) RETURNS {void | alias [, alias ...]}
{pig_latin_fragment};
```

Tras indicar el nombre que deseemos asignarle a la macro, indicamos los nombres de los parámetros vaya a tener nuestra macro. Estos alias los podremos usar dentro de nuestra macro añadiendo el símbolo $ al alias del parámetro. Con RETURN se indican los alias de las relaciones que devolverá la macro, y si no devuelve ninguna debemos escribir “void”. Por último, entre llaves, podemos escribir las sentencias Pig Latin de nuestra macro. Por ejemplo vamos a crear una macro que muestre sólo las tuplas con el eje X del acelerómetro (Ax) de la relación “datosMPU” con valor 0.00. Para ello podemos introducir la siguiente sentencia:

```
DEFINE macroFiltro(nombre_relacion, nombre_columna) RETURNS resultadoMacro{
$resultadoMacro = FILTER $nombre_relacion BY $nombre_columna==0.00;
};
```

Para poder ejecutar una prueba de la macro podemos usar la sentencia:

```
ejecucionMacro = macroFiltro (datosMPU, Ax);
```

Una vez hecho esto, podemos realizar “DUMP ejecucionMacro”, y obtendremos el resultado final, el cual podemos ver en la Ilustración 46. En ella se observa que todas las tuplas tienen en la columna Ax el valor 0.00.

```
(-1.92,-2.36,1.21,0.0,0.18,0.75,36.48)
(0.8,-1.76,14.46,0.0,-0.08,-0.36,32.06)
(23.67,-3.17,2.44,0.0,0.16,0.35,32.44)
(-0.82,-2.22,-0.23,0.0,0.05,0.86,32.53)
(-0.17,-2.35,-0.41,0.0,-0.0,0.84,32.53)
(-0.27,-2.28,-0.32,0.0,0.02,0.83,32.62)
(-0.42,-2.27,-0.16,0.0,0.02,0.84,32.62)
(-0.18,-2.24,-0.26,0.0,0.0,0.86,32.53)
(-0.05,-2.24,-0.42,0.0,-0.01,0.86,32.62)
(-0.4,-2.37,-0.12,0.0,0.03,0.86,32.53)
(-0.39,-2.3,-0.24,0.0,0.02,0.85,32.53)
(-0.08,-2.31,-0.07,0.0,0.95,0.09,32.53)
(0.18,-2.29,-0.08,0.0,0.89,0.08,32.62)
```

Ilustración 46: Resultado de la ejecución de la macro "macroFiltro"

5.2.1.19 IMPORT

IMPORT nos permite importar macros que hayamos definido previamente en un fichero con extensión “.pig”. Su sintaxis es muy sencilla:

```
IMPORT ´nombre_fichero.pig`;
```

Sólo hay que indicar el nombre o la ruta donde se encuentre nuestro fichero. La macro que creamos anteriormente era bastante sencilla, por ello, vamos a crear una nueva macro que tenga un poco más de dificultad para ver la utilidad de este operador. El objetivo será la obtención de la temperatura media cuando el eje x del giroscopio (Gx) tiene valor negativo y cuando tiene valor positivo. Sólo tendríamos que introducir como parámetro la relación "datosMPU" y obtendríamos como salida dos relaciones, una cuando Gx es negativo y otra cuando es positivo. La macro la vamos a definir en el fichero "temperaturaMedia.pig", que lo ubicaremos en el directorio "/home/hduser/temperaturaMedia.pig" del sistema de ficheros local. Este fichero tendrá una macro del mismo nombre "temperaturaMedia", cuyo contenido podemos ver en la Ilustración 47.

```
DEFINE temperaturaMedia(nombre_relacion) RETURNS tempMenor, tempMayor{
SPLIT $nombre_relacion INTO filtroMenor IF Gx < 0, filtroMayor IF Gx > 0;
agrupacionMenor = GROUP filtroMenor ALL;
agrupacionMayor = GROUP filtroMayor All;

$tempMenor = FOREACH agrupacionMenor GENERATE
        COUNT(filtroMenor) AS num_tuplas,
        AVG(filtroMenor.T) AS temp_media;

$tempMayor = FOREACH agrupacionMayor GENERATE
        COUNT(filtroMayor) AS num_tuplas,
        AVG(filtroMayor.T) AS temp_media;
};
```

Ilustración 47: Macro "temperaturaMedia"

Como podemos ver, esta macro lo primero que hace es dividir la relación que pasemos como parámetro en dos relaciones, la que tiene Gx positivo y la que tiene Gx negativo. A continuación, genera una relación en la que agrupa todas las tuplas de la relación que tiene Gx positivo usando la cláusula ALL, también hace lo mismo para la que tiene Gx negativo. Finalmente, con FOREACH, genera una tupla que va a tener el número de tuplas de la relación y la temperatura media, tanto para Gx positivo como Gx negativo. Podemos ver que hay dos cláusulas que no habíamos visto anteriormente, estas son COUNT y AVG. Pig posee muchas opciones, con cláusulas y operaciones diferentes, por lo que se alargaría demasiado la explicación de todas ellas. En este caso, la clausula COUNT cuenta las tuplas de una relación, y AVG calcula la media. Una vez creado este archivo podemos importarlo usando la siguiente sentencia:

IMPORT '/home/hduser/ temperaturaMedia.pig';

Una vez cargado, ya podemos usar la macro de la misma forma que hicimos anteriormente, y ejecutarla con:

"tempMenor, tempMayor = temperaturaMedia(datosMPU);".

El resultado lo podemos ver en la Ilustración 48, tras ejecutar "DUMP tempMenor;" y "DUMP tempMayor;", la primera línea se corresponde con "tempMenor", en la que la temperatura media es de 28.46ºC, y la segunda es "tempMayor", cuya temperatura media es de 32.67ºC.

```
(39686,28.459897192965865)

(1004,32.668426294820726)
```

Ilustración 48: Resultado de la ejecución de la macro "temperaturaMedia"

5.2.1.20 STREAM

STREAM envía datos a un script o programa externo. Su sintaxis es la siguiente:

STREAM alias [, alias …] THROUGH {‘comando’ | cmd_alias} [AS esquema];

Primero se indica el alias de la relación sobre el que queremos realizar la operación. Con THROUGH podemos indicar el comando entre comillas simples o el alias que le hayamos dado a un script o un comando. usando el operador DEFINE. Con AS podemos indicar el esquema. Por ejemplo, para mostar el funcionamiento vamos a hacer que la relación “datosMPU” se imprima por pantalla usando el comando “cat -n”, con alias “cat” que creamos anteriormente. Para ello solo tendremos que usar la siguiente sentencia:

numeroLinea = STREAM datosMPU THROUGH cat;

Una muestra del resultado la podemos ver en la Ilustración 49 tras ejecutar “DUMP numeroLinea”. En ella se puede ver que cada línea tiene incluido el número de línea que le corresponde. Las líneas que se muestran son las últimas del fichero, por ello la última tiene el número 40699, que es el número de líneas totales que había en el fichero “datosMPU.txt” original.

```
( 40694,-0.23,-2.23,-0.24,-0.95,-0.0,-0.47,26.98)
( 40695,-0.22,-2.21,-0.24,-0.95,-0.01,-0.46,26.88)
( 40696,-0.22,-2.2,-0.27,-0.94,-0.01,-0.45,26.98)
( 40697,-0.19,-2.18,-0.25,-0.94,-0.02,-0.47,26.88)
( 40698,-0.17,-2.11,-0.27,-0.95,-0.01,-0.47,26.88)
( 40699,-0.21,-2.27,-0.31,-0.93,-0.01,-0.48,26.88)
```

Ilustración 49: Resultado de la operación STREAM

6 CONCLUSIONES Y LÍNEAS FUTURAS

Para finalizar este trabajo, vamos a realizar una conclusión, comentando la realización del proyecto, así como destacando si los objetivos planteados en el Capítulo 1 han podido ser alcanzados. Además, comentaremos unas posibles líneas futuras que permitan mejorar este proyecto y que lo hagan aún más interesante.

6.1 Conclusiones

Como ya hemos podido ver, Big Data está ampliamente extendido en nuestra sociedad hoy en día. Son muchas las empresas, investigaciones, entre otros cientos de casos, que lo utilizan para conseguir unos resultados que, sin la ayuda de Big Data, serían imposibles. La elección de centrarnos en Hadoop se debe a que es uno de los frameworks más extendidos hoy en día, y como hemos podido comprobar, tiene un potencial enorme.

Nunca antes había tenido la oportunidad de entender nada de Big Data y menos aún de Hadoop. Este proyecto ha sido un reto para mí, en el que he tenido que conocer, primeramente, qué es Big Data, y después, entender Hadoop, su arquitectura y sus posibilidades. Se puede decir con total seguridad que no ha decepcionado.

Además, no sólo tenemos Hadoop, también pudimos conocer todo el amplio ecosistema que existe. Este ecosistema ofrece posibilidades prácticamente infinitas, por lo que sería imposible tratar de cubrirlas todas en este mismo proyecto. Como el objetivo final es poder analizar los datos y sacar conclusiones de este análisis, es por ello que, de todas las opciones, decidimos seleccionar Apache Pig. Pig nos permite analizar los datos de una forma mucho más sencilla a como lo haríamos usando otros lenguajes de programación como Java.

Cuando escuchamos análisis de datos usando Big Data, podemos pensar en superordenadores con una capacidad de procesamiento enorme, que ocupen una habitación, procesando estos datos. Pero ya hemos podido demostrar en este trabajo que esto no es así. Hadoop nos permite procesar los datos en un hardware básico, el mismo que tenemos en los ordenadores de nuestras casas. Uno de los principales objetivos de este trabajo era demostrar que esto era posible, y para ello decidimos usar cuatro Raspberry Pi 3 model B+. El hardware de estos ordenadores, porque, aunque a simple vista no lo parezcan, realmente son ordenadores, es muy económico. Por lo que la mejor forma de demostrar que el hardware básico sirve para Big Data, era usando estas Raspberry Pi, que es de lo más económico que podemos encontrar en el mercado. Los resultados han sido totalmente satisfactorios, ya que, a pesar de la poca capacidad de procesamiento de estas Raspberry Pi, hemos podido instalar y usar Hadoop sin ningún tipo de problema.

Vamos a hacer un repaso sobre los objetivos planteados en el Capítulo 1 para ver si hemos podido cumplirlos:

- 1.- Introducción al mundo de Big Data y explicación de sus posibilidades: En el Capítulo 2 pudimos entender el concepto de Big Data, explicar por qué tiene una gran importancia y vimos algunos casos de uso para comprobar sus aplicaciones.
- 2.- Explicación de Hadoop, sus componentes y su ecosistema: En el Capítulo 3, estudiamos Hadoop y sus 3 componentes principales. En él, profundizamos en su funcionamiento y explicamos qué estructura seguían esos componentes. Además, vimos a modo de resumen las diferentes opciones que nos podemos encontrar en su ecosistema, así como realizando un repaso de las diferentes versiones de

Hadoop.

- 3.- Montaje del clúster de Raspberry Pi 3 Model B+: En el Capítulo 4, explicamos todos los componentes usados en nuestro clúster y detallamos el proceso de montaje del escenario planteado. En el Anexo 1, también se incluye toda la configuración que fue necesaria realizar para el posterior correcto funcionamiento del clúster.
- 4.- Instalación de Hadoop en el clúster de Raspberry Pi: En el Anexo 2, se explica de forma detallada el proceso de instalación de Hadoop en todas las Raspberry Pi y las configuraciones que fueron necesarias para hacerlo funcionar de la forma deseada.
- 5.- Instalación de Pig en el clúster Hadoop: En el Anexo 3, se detalló el proceso de instalación de Pig en una Raspberry Pi. Sólo fue necesario instalarlo en una de ellas, que fue la Rasperry Pi a la que denominamos "master".
- 6.- Conexión del MPU6050 a la Raspberry Pi y toma de datos del mismo: En el Capítulo 4, también se detalló la conexión del sensor MPU6050 a la Raspberry Pi "master". Para la obtención de datos, se usó el script explicado en el Anexo 4. También se detalló cómo se realizó la toma de datos en el Capítulo 5.1.
- 7.- Vistazo de las herramientas que nos ofrece Hadoop en el clúster: En el Capítulo 5.2, se realizó una demostración de la interfaz gráfica que nos ofrece HDFS para el manejo de nuestro clúster, así como el uso de los comandos para poder gestionar el sistema de almacenamiento.
- 8.- Ejemplo de cada una de las operaciones más importantes de Pig Latin: Finalmente, en el Capítulo 5.3 estudiamos las operaciones más relevantes de Pig Latin y realizamos un ejemplo de cada una de ellas usando los datos obtenidos por el sensor MPU6050.

Por tanto, con este trabajo, hemos podido entender el Big Data y el funcionamiento de Hadoop. Ha quedado demostrado que con uno de los ordenadores más básico y barato que podemos encontrar, podemos realizar análisis de Big Data usando Hadoop. Hemos podido usar las diversas operaciones que Apache Pig nos ofrece y mostrar un ejemplo de cada una de ellas usando datos del sensor MPU6050.

6.2 Líneas futuras

Los datos que nos ofrece el sensor MPU6050 no son inútiles, podrían ser usandos en aplicaciones prácticas. Para ello, primero se podría mejorar la inserción de datos en el clúster. En el ecosistema de Hadoop, tenemos herramientas como Spark, el cual le está arrebatando protagonismo a Hadoop en la actualidad, o Apache Kafka. Estas herramientas nos permitirían insertar los datos en tiempo real en el clúster, de forma que no sería necesario almacenar los datos primeramente en un fichero de texto, y después almacenarlo en HDFS.

A esto se le podría añadir un módulo de tarjeta SIM, que nos permita transmitir los datos que captura el sensor por la red móvil, y recibirlos en el clúster a distancia. Esto abriría muchas posibilidades, ya que podemos usar el sensor en otros entornos, donde nos ofrezca información útil.

Uno de estos entornos, podría ser el de los coches. En los últimos tiempos, se escucha mucho hablar de los coches conectados. Estos coches tienen muchos sensores que trasmiten información la cual necesita ser procesada. Este sensor podría instalado en un coche, de forma que transmita los datos al clúster de Hadoop. Con estos datos, podríamos ver las aceleraciones y desaceleraciones del coche, de manera que se podría evaluar los estados de los frenos o nuestros hábitos en la conducción. También podríamos analizar los datos de la temperatura o los del giroscopio para poder obtener otras conclusiones. Además, no solo nos permitiría analizar estos datos una vez almacenados, sino también hacerlo en tiempo real gracias a las herramientas previamente mencionadas.

REFERENCIAS

[1] Data Never Sleeps 7.0, [En línea]. Disponible en: https://www.domo.com/learn/data-never-sleeps-7

[2] Internet de las Cosas: 75 mil millones de objetos estarán conectados en el mundo en 2025, según informes, [En línea]. Disponible en: https://www.galileo.edu/trends-innovation/internet-de-las-cosas-75-mil-millones-de-objetos-conectados-en-2025/

[3] Así han evolucionado los tamaños de fabricación de semiconductores, [En línea]. Disponible en: https://www.adslzone.net/2016/06/27/asi-evolucionado-los-tamanos-fabricacion-semiconductores/

[4] Diferencia entre datos estructurados y no estructurados, [En línea]. Disponible en: https://www.kyoceradocumentsolutions.es/es/smarter-workspaces/insights-hub/articles/diferencia-entre-datos-estructurados-y-no-estructurados.html

[5] Big Data: El impacto de los datos masivos en la sociedad – Democratización del Big Data, [En línea]. Disponible en: https://www.coursera.org/learn/impacto-datos-masivos/lecture/fAAN3/democratizacion-del-big-data

[6] Big Data: El impacto de los datos masivos en la sociedad – Toma de decisiones basadas en datos, [En línea]. Disponible en: https://www.coursera.org/learn/impacto-datos-masivos/lecture/nJm3O/toma-de-decisiones-basadas-en-datos

[7] Big Data: El impacto de los datos masivos en la sociedad – Caso 2: Big Data en el sector Seguros, [En línea]. Disponible en: https://www.coursera.org/learn/impacto-datos-masivos/lecture/mQN7t/caso-2-big-data-en-el-sector-seguros

[8] Netflix te crea portadas personalizadas según tu perfil de usuario, [En línea]. Disponible en: https://clipset.com/netflix-portadas-personalizadas/

[9] Los avances científicos sobre el COVID-19 compartidos gracias al Big Data, [En línea], Disponible en: https://virtualdesk.es/2020/04/30/covid19-big-data/

[10] Lo que dice el "big data" sobre este Madrid extraño, [En línea], Disponible en: https://elpais.com/espana/madrid/2020-10-24/lo-que-dice-el-big-data-sobre-este-madrid-extrano.html

[11] Fregata Space usa el Big Data para evitar que más de 8 millones de toneladas de plásticos acaben cada año en los océanos, [En línea]. Disponible en: https://www.europapress.es/epsocial/responsables/noticia-fregata-space-usa-big-data-evitar-mas-millones-toneladas-plasticos-acaben-cada-ano-oceanos-20201030193840.html

[12] Por qué las redes sociales podrían estar dañando la democracia, [En línea]. Disponible en: https://elpais.com/tecnologia/2017/11/10/actualidad/1510310772_776262.html

[13] Cambridge Analytica, el big data y su influencia en las elecciones, [En línea]. Disponible en: https://www.telesurtv.net/opinion/cambridge-analytica-elecciones-eeuu-facebook-20180402-0063.html

[14] Facebook compartió datos sensibles de sus usuarios con más de 150 grandes empresas, [En línea]. Disponible en: https://elpais.com/tecnologia/2018/12/19/actualidad/1545221673_589059.html

[15] Apache Hadoop, [En línea]. Disponible en: https://hadoop.apache.org

[16] Hadoop: Introducción, Componentes y Ecosistema, [En línea]. Disponible en: http://blog.jacagudelo.com/hadoop-introduccion-componentes-y-ecosistema/#Breve_Historia

[17] Hadoop Essentials – Shiva Achari, [En línea]. Disponible en: https://fama.us.es/permalink/34CBUA_US/1pfq23c/alma991013032240804987

[18] Big Data: adquisición y almacenamiento de datos – El ecosistema Hadoop, [En línea]. Disponible en: https://www.coursera.org/learn/adquisicion-almacenamiento-de-datos/lecture/vhjfJ/el-ecosistema-hadoop

[19] HDFS Architecture, [En línea]. Disponible en: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

[20] Apache Hadoop Tutorial | Hadoop Tutorial For Beginners | Big Data Hadoop | Hadoop Training | Edureka, [En línea]. Disponible en: https://www.youtube.com/watch?v=mafw2-CVYnA

[21] Deep-drive: Understanding NameNode, [En línea]. Disponible en: https://hadoopabcd.wordpress.com/2015/05/31/deep-drive-understanding-namenode/

[22] Apache Hadoop YARN, [En línea]. Disponible en: https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YARN.html

[23] YARN: NExtGen Hadoop Data Processing Framework, [En línea]. Disponible en: https://hadoopabcd.wordpress.com/2015/03/27/yarn-nextgen-hadoop-data-processing-framework/

[24] Download – Apache Hadoop, [En línea]. Disponible en: https://hadoop.apache.org/releases.html

[25] Differences between Hadoop 1.x, 2.x and 3.x?, [En línea]. Disponible en: https://thebigan.wordpress.com/2017/09/11/differences-between-hadoop-1-x-2-x-and-3-x/

[26] HDFS Erasure Coding in Big Data Hadoop – An Introduction, [En línea]. Disponible en: https://data-flair.training/blogs/hadoop-hdfs-erasure-coding/

[27] Comparison Between Hadoop 2.x vs Hadoop 3.x, [En línea]. Disponible en: https://data-flair.training/blogs/hadoop-2-x-vs-hadoop-3-x-comparison/

[28] Apache Drill, [En línea]. Disponible en: https://drill.apache.org

[29] Apache Storm, [En línea]. Disponible en: https://storm.apache.org

[30] Apache Kafka, [En línea]. Disponible en: https://kafka.apache.org/intro

[31] Apache Lucene, [En línea]. Disponible en: https://lucene.apache.org

[32] Pig 0.17.0 Documentation, [En línea]. Disponible en: http://pig.apache.org/docs/r0.17.0/index.html

[33] Análisis de Big Data con Apache Pig, [En línea]. Disponible en: https://developer.ibm.com/es/technologies/data-management/articles/bigdata-apachepig/

[34] Pig Hadoop: Devorando Datos, [En línea]. Disponible en: http://blog.jacagudelo.com/pig-hadoop/

[35] Distribuciones Hadoop: ¿Cómo gestionar tu clúster Hadoop?, [En línea]. Disponible en: https://www.agiliacenter.com/distribuciones-hadoop-gestionar/

[36] Cloudera, MapR, Hortonworks… ¿Qué distribución Hadoop necesitas? (IV y fin), [En línea]. Disponible en: https://empresas.blogthinkbig.com/cloudera-mapr-hortonworksque/

[37] Ecosistema de Apache Hadoop, [En línea]. Disponible en: https://es.cloudera.com/products/open-source/apache-hadoop.html

[38] Raspberry Pi 3 Model B+, [En línea]. Disponible en: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/?resellerType=home

[39] TP-Link TL-SG108 V3.0, [En línea]. Disponible en: https://www.amazon.es/TP-Link-TL-SG108-Gigabit-Escritorio-Auto-MDI/dp/B01EXDG2MO

[40] Tutorial MPU6050, Acelerómetro y Giroscopio, [En línea]. Disponible en: https://naylampmechatronics.com/blog/45_Tutorial-MPU6050-Acelerómetro-y-Giroscopio.html

[41] Introducción al giroscopio, [En línea]. Disponible en: https://www.5hertz.com/index.php?route=tutoriales/tutorial&tutorial_id=13#:~:text=MEMS%20(sistemas%20microelectromecánicos)%20giroscopios%20son,de%20la%20velocidad%20de%20rotación.

[42] Sensor MPU6050, [En línea]. Disponible en: https://www.diarioelectronicohoy.com/blog/sensor-mpu6050

[43] Aprendiendo Aduino, [En línea]. Disponible en: https://aprendiendoarduino.wordpress.com/2017/07/09/i2c/#:~:text=I2C%20es%20un%20bus%20de,velocidades%20de%203.4%20Mbit%2Fs.

[44] Recolección de datos de sensores en la IoT con Arduino y su gestión mediante un protocolo de mensajes basado en el patrón publicación y subscripción, Dolores Raigada Romero.

[45] Para Raspberry Pi 4 Model B,Raspberry Pi 3 B + Caja con Ventilador de refrigeración y disipador de Calor, [En línea]. Disponible en: https://www.amazon.es/gp/product/B07J9VMNBL/ref=ppx_yo_dt_b_search_asin_title?ie=UTF8&psc=1

[46] GPIO, [En línea]. Disponible en: https://www.raspberrypi.org/documentation/usage/gpio/

[47] Pig Latin Basics, [En línea]. Disponible en: http://pig.apache.org/docs/r0.17.0/basic.html#import

[48] Noobs, [En línea]. Disponible en: https://www.raspberrypi.org/downloads/noobs/

[49] Building a Raspberry Pi Hadoop / Spark cluster, [En línea]. Disponible en: https://dev.to/awwsmm/building-a-raspberry-pi-hadoop-spark-cluster-8b2

[50] Raspberry Pi Hadoop cluster guide, [En línea]. Disponible en: https://medium.com/data-waffles/raspberry-pi-hadoop-cluster-guide-2559437d232

[51] Building a Hadoop cluster with Raspberry Pi, [En línea]. Disponible en: https://developer.ibm.com/recipes/tutorials/building-a-hadoop-cluster-with-raspberry-pi/

[52] Release 2.7.7 available, [En línea]. Disponible en: https://hadoop.apache.org/release/2.7.7.html

[53] Parámetros de configuración core-site.xml, [En línea]. Disponible en: https://hadoop.apache.org/docs/r3.3.0/hadoop-project-dist/hadoop-common/core-default.xml

[54] Parámetros de configuración hdfs-site.xml, [En línea]. Disponible en: https://hadoop.apache.org/docs/r2.7.7/hadoop-project-dist/hadoop-hdfs/hdfs-default.xml

[55] Parámetros de configuración mapred-site.xml, [En línea]. Disponible en: https://hadoop.apache.org/docs/r2.7.7/hadoop-mapreduce-client/hadoop-mapreduce-client-core/mapred-default.xml

[56] Parámetros de configuración yarn-site.xml, [En línea]. Disponible en: https://hadoop.apache.org/docs/r2.7.7/hadoop-yarn/hadoop-yarn-common/yarn-default.xml

[57] Apache PIG: Installation and Connect with Hadoop Cluster, [En línea]. Disponible en: https://blog.knoldus.com/apache-pig-installation-and-connect-with-hadoop-cluster/

[58] MPU6050 Gyro Sensor Interfacing with Raspberry Pi, [En línea]. Disponible en: https://circuitdigest.com/microcontroller-projects/mpu6050-gyro-sensor-interfacing-with-raspberry-pi

[59] MPU6050 (Accelerometer+Gyroscope) Interfacing with Raspberry Pi, [En línea]. Disponible en: https://www.electronicwings.com/raspberry-pi/mpu6050-accelerometergyroscope-interfacing-with-raspberry-pi

[60] MPU6050 (Gyroscope + Accelerometer + Temperature) Sensor Module, [En línea]. Disponible en: https://www.electronicwings.com/sensors-modules/mpu6050-gyroscope-accelerometer-temperature-sensor-module

ANEXOS

1. Configuración inicial de cada Raspberry pi

Vamos a ver cómo hemos configurado inicialmente cada una de las cuatro Raspberry Pi. Básicamente vamos a explicar cómo hemos instalado Raspberry Pi OS usando NOOBS y qué otras configuraciones hemos realizado una vez instalado para el correcto funcionamiento del trabajo. NOOBS es un instalador de sistemas operativos que contiene los sistemas operativos Raspberry Pi OS y LibreELEC. También podemos instalar otras alternativas que este instalador descargará a través de internet.

Primeramente, tenemos que descargar NOOBS desde la página oficial de Rasperry [48]. Se descargará un archivo ".zip" que tendremos que descomprimir una vez finalice su descarga. Como sabemos del apartado 4.2, contamos con cuatro tarjetas microSD, una para cada Raspberry Pi. Usando un adaptador microSD a SD vamos a conectar una a una las tarjetas en el lector de tarjetas SD incorporado en el ordenador portátil.

Una vez conectada la primera micro SD al ordenador portátil, abrimos el programa SD Card Formatter y formateamos la tarjeta microSD. Nos dejará la tarjeta formateada con una sola partición de 32 GB en formato FAT32.

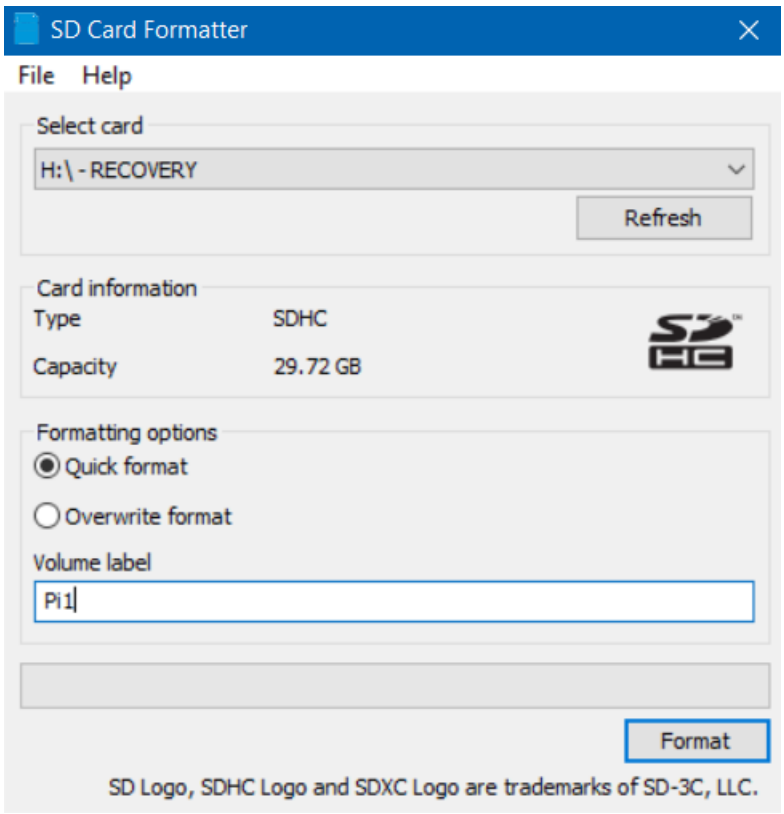


Ilustración 50: SD Card Formatter

A continuación, extraemos la tarjeta microSD del ordenador portátil y del adaptador, y la insertamos en la Raspberry Pi. Conectamos un cable HDMI a la Raspberry para ver la imagen en un monitor, y conectamos un teclado y un ratón Bluetooth usando los puertos USB. Una vez esté todo conectado, conectamos la Raspberry Pi a la corriente usando el adaptador de corriente y veremos el menú de instalación de NOOBS en pantalla.

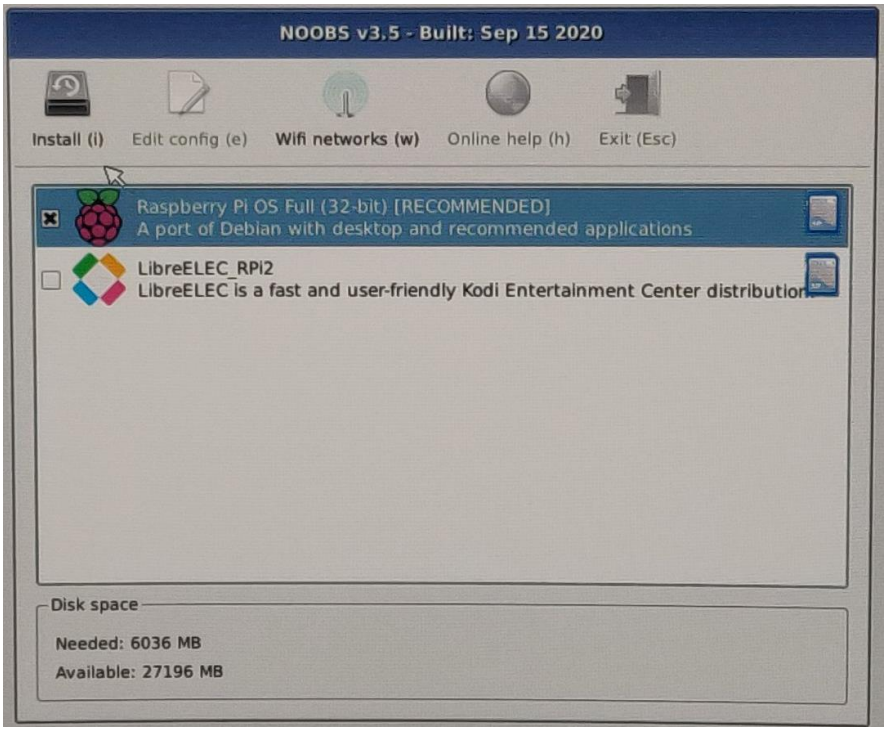


Ilustración 51: Pantalla inicial de NOOBS

Hacemos click en la opción de “Rasperry Pi OS Full (32-bit) [RECOMMENDED]” y damos click en “Install (i)”. Esto comenzará el proceso de instalación de Rasperry Pi OS en nuestra Raspberry Pi. Tras un tiempo la instalación habrá terminado. Aceptamos y la Raspberry Pi se reiniciará. Cuando vuelva a encenderse tendremos acceso a un menú de configuración de Raspberry Pi OS.

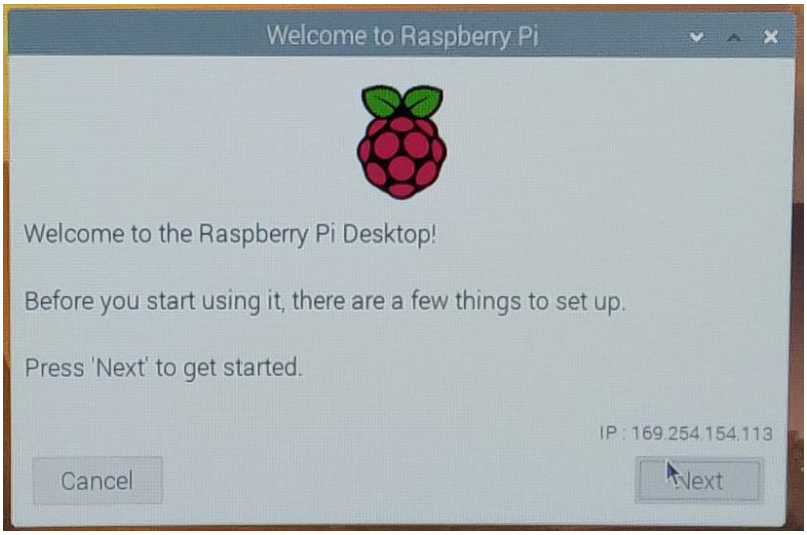


Ilustración 52: Menú de configuración de Rasperry Pi OS

Al hacer click en “Next” tendremos que configurar el país, el idioma y la zona horaria. En nuestro caso hemos seleccionado España como país, español como lenguaje, y zona horaria Madrid (GMT+1).

Ilustración 53: Selección país, idioma y zona horaria en Rasperry Pi OS

Una vez configurada, tendremos que establecer una contraseña para el usuario “pi” de Rasperry Pi OS, que en este caso hemos usado un pin de 4 dígitos.

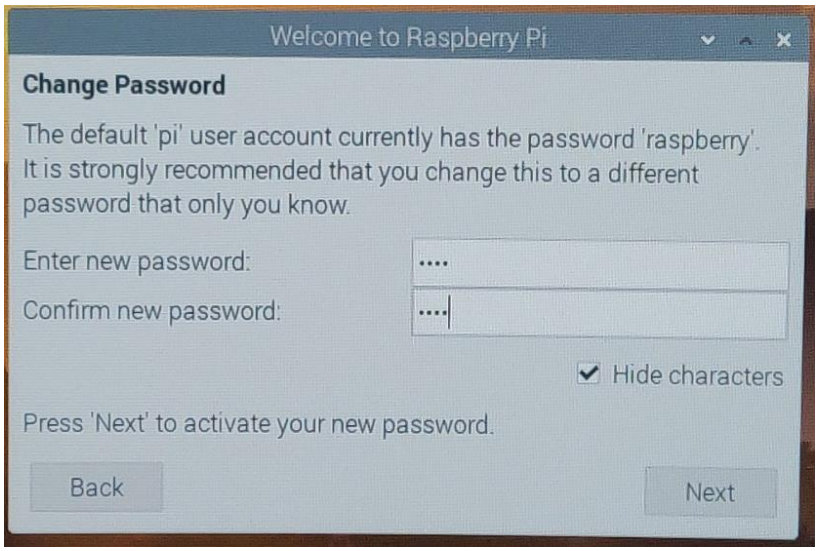


Ilustración 54: Establecimiento de contraseña para usuario “pi” en Raspbian OS

Seguidamernte, tenemos que seleccionar una red inalámbrica para conectarnos mediante wifi. Una vez la selecciones, introducimos la contraseña y nos conectamos.

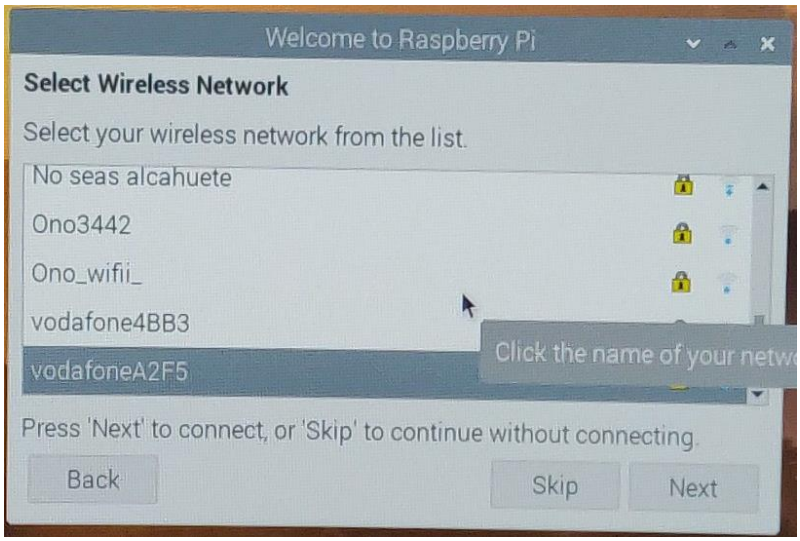


Ilustración 55: Selección red wifi en Raspbery Pi OS

Ahora nos preguntará si la pantalla tiene bordes negros para que pueda ajustarse la resolución al monitor correctamente. En nuestro caso como se veían bordes negros seleccionamos la opción y le damos a siguiente. Esto ajustará la imagen a los bordes del monitor correctamente en el próximo reinicio.

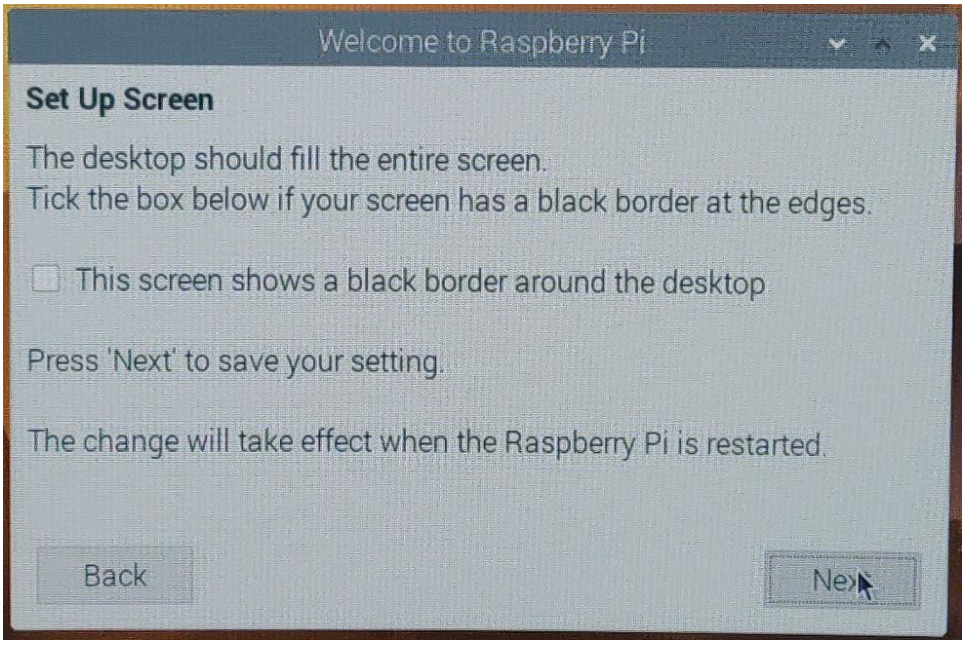


Ilustración 56: Selección bordes negros en Raspberry Pi OS

Finalmente, nos preguntará si queremos comprobar si hay actualizaciones diponibles. En este caso le hemos dado a siguiente para que todas las Raspberry Pi estén actualizadas a la útima versión de Raspberry Pi OS disponible.

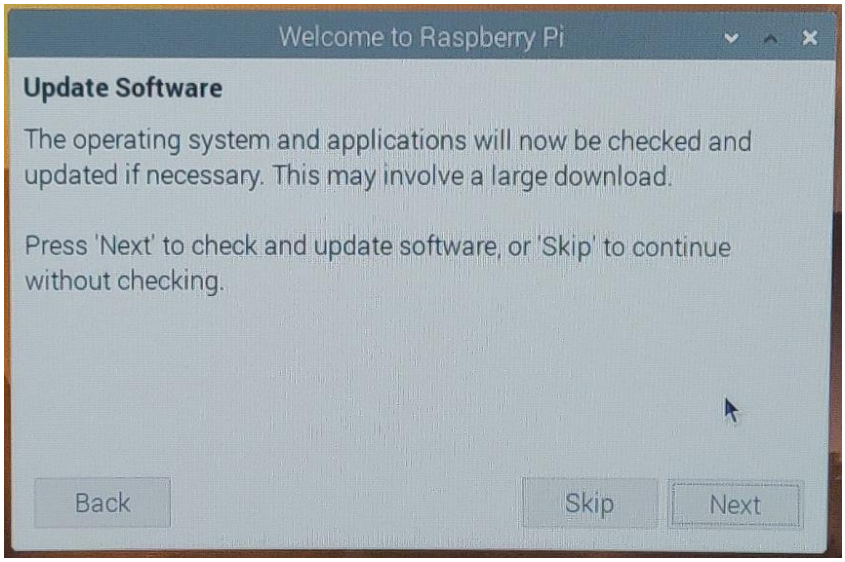


Ilustración 57: Actualización Raspberry Pi OS

Una vez se complete la actualización, tendremos que reiniciar la Raspberry Pi y ya tendremos correctamente instalado y configurado Raspberry Pi OS. Con la instalación de Raspberry Pi OS no finaliza este apartado ya que aún quedan unas configuraciones que realizar para dejar las Raspberry Pi completamente preparadas para poder instalar Hadoop más adelante.

Como mencionamos anteriormente, usamos un teclado USB y un ratón Bluetooth, para el cual también hay que conectar el dispositivo bluetooth a un USB. Raspberry Pi OS tiene un problema inicial con los ratones Bluetooth, y es que el tiempo de respuesta es muy lento y se hace muy tedioso poder usar el ratón. Por tanto, para solucionar este problema y que el ratón se comporte correctamente tenemos que añadir el texto "usbhid.mousepoll=0" al final del fichero "/boot/cmdline.txt". Una vez editado y guardado (hemos usado "nano" para ello a través de la consola), podemos reiniciar nuestra Raspberry Pi y el ratón funcionará con normalidad.

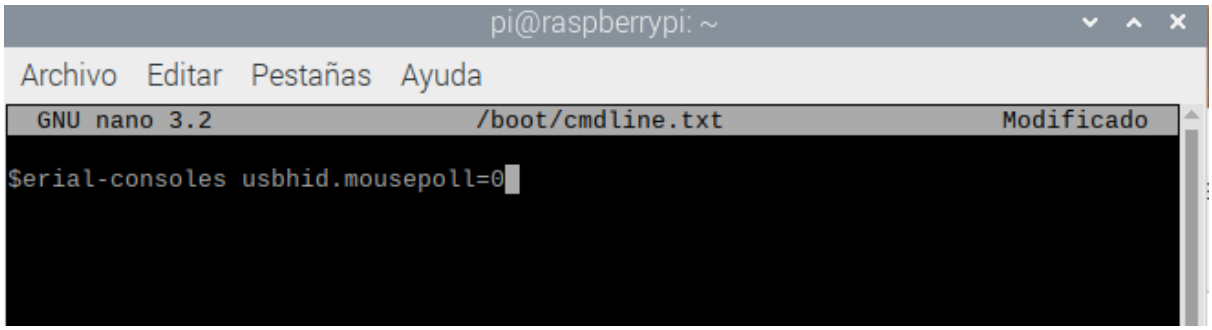


Ilustración 58: Modificación en el fichero "/boot/cmdline.txt"

Todos estos pasos, tanto de instalación de Raspberry Pi OS como la solución a los problemas con el ratón Bluetooth tenemos que realizarlos en cada una de las cuatro Raspberry Pi.

Durante el resto del proyecto no vamos a estar usando las Rasperry Pi a través del monitor, esto sólo ocurrirá en este apartado ya que era necesario para instalar y configurar Rasperry Pi OS. Durante el proyecto usaremos las Raspberry Pi a través de la consola, conectándonos a través de SSH desde el ordenador portátil con la consola de Windows 10. La conexión será vía ethernet gracias al conmutador al que estarán conectadas todas las Raspberry Pi y el ordenador portátil. Por esto, lo último que tenemos que realizar es la configuración IP de cada una de las Raspberry Pi. La forma más cómoda es proporcionando una IP estática a cada una de las Raspberry Pi, será más sencillo conectarse a ellas a través de SSH y también para que se comuniquen entre sí al mantener siempre la misma dirección IP.

Para poder realizar esto nos vamos al archivo "/etc/dhcpcd.conf", y en él descomentamos las líneas de "interface eth0" y la de "static ip_address". En el parámetro "static ip_address" asignamos la dirección IP que queramos. En este caso para la Raspberry Pi 1 le damos la dirección 192.162.1.10, a la Raspberry Pi 2 la 192.168.1.20, a la Raspberry Pi 3 la 192.168.1.30 y a la Raspberry Pi 4 la 192.168.1.40.

```
# Example static IP configuration:
interface eth0
static ip_address=192.168.1.10/24
#static ip6_address=fd51:42f8:caae:d92e::ff/64
#static routers=192.168.0.1
#static domain_name_servers=192.168.0.1 8.8.8.8 fd51:42f8:caae:d92e::1
```

Ilustración 59: Configuración IP estática en Rasperry Pi 1

Una vez modificado el archivo, lo guardamos. Por último, queda habilitar SSH, por tanto, para habilitarlo usamos los comandos "sudo systemctl enable ssh" y "sudo systemctl start ssh". Reiniciamos una a una las Raspberry Pi.

También hay que darle una dirección IP correcta al ordenador portátil para que pueda conectarse correctamente. Para dar una dirección IP estática a la interfaz ethernet del ordenador con Windows 10 nos vamos a "Panel del control" -> "Redes e Internet" -> "Centro de redes y recursos compartidos" -> "Cambiar configuración del adaptador". En este apartado veremos que aparece la interfaz "ethernet", hacemos click derecho sobre ella y seleccionamos "Propiedades". Seleccionamos "Protocolo de Internet versión 4 (TCP/IPv4)" y "Propiedades". En esta nueva ventana ya podemos introducir la IP versión 4 deseada, en nuestro caso será la 192.168.1.105 como podemos ver en la Ilustración 60.

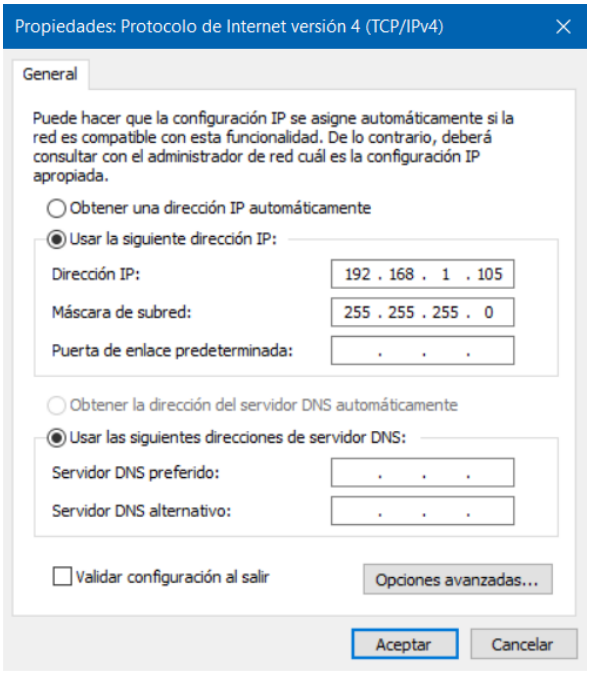


Ilustración 60: Configuración IP estática en Windows 10

Para poder usar las consolas de las Raspberry a través de Windows 10, sólo tenemos que abrir la consola de Windows y usar el comando "ssh nombre@ip". Por ejemplo, para conectarnos a la Raspberry 2 sólo tenemos que utilizar el comando "ssh pi@192.168.0.20".

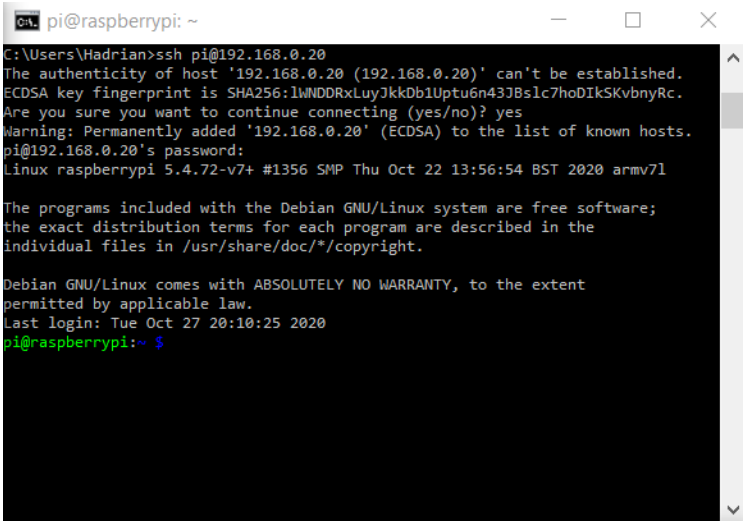


Ilustración 61: Conexión SSH desde el ordenador portátil a Raspberry Pi 2

2. Instalación y configuración del clúster Hadoop

En este apartado vamos a ver la instalación y configuración completa de Hadoop en nuestro clúster de Raspberry Pi [49] [50] [51]. Antes de comenzar con la descarga e instalación vamos a configurar los "hostname", es decir, el nombre de cada Raspberry Pi dentro del clúster. Así, a lo largo del trabajo nos resultará más fácil y cómodo identificarlas.

La Raspberry Pi 1 será el maestro, por ello la llamaremos "master", y a las Raspberry Pi 2, 3 y 4, que serán los esclavos, las llamaremos "worker1", "worker2" y "worker3", respectivamente. Esto es muy sencillo de realizar. Como indicamos en el Anexo 1, a partir de ahora las configuraciones de realizarán a través del ordenador portátil con una conexión SSH. Por tanto, conectamos el ordenador al conmutador e iniciamos con cada Raspberry Pi una sesión SSH. El primer archivo que vamos a modificar es el "/etc/hostname" para cambiar el nombre de cada Raspberry Pi. Cuando lo abramos nos encontraremos que el hostname actual es "raspberry", lo único que tenemos que hacer es borrar ese nombre y cambiarlo por el nombre correspondiente a cada Raspberry Pi como se ha explicado.

A continuación, editamos el fichero "etc/hosts", aquí asociaremos el hostname de cada Raspberry Pi a las diferentes direcciones IP. En este fichero comentamos la dirección 127.0.1.1 para que no nos de problemas en el futuro e introducimos las direcciones y los nombres de todas las Raspberry Pi del clúster. Una vez completados todos los pasos, reiniciamos cada una de las Raspberry Pi.

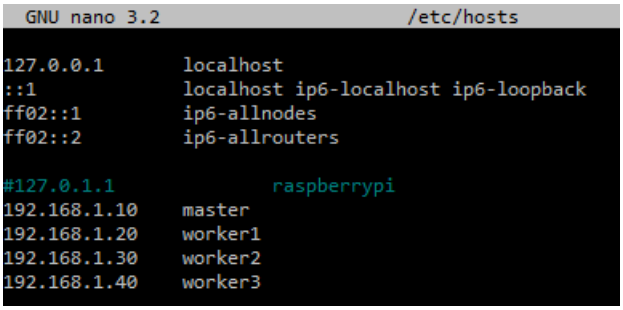


Ilustración 62: Configuración del fichero "/etc/hosts/" en la Raspberry Pi "master"

Ahora vamos a crear un usuario específico para usar Hadoop, de esta forma es mucho más cómodo gestionar el clúster. Para ello tenemos que introducir los comandos de Ilustración 63. Con ellos lo que haremos es crear un grupo llamado "hadoop" al que hemos añadiremos el usuario "hduser", que también lo hemos añadido al grupo

"sudo". El usuario "hduser" será el que utilicemos para instalar y usar Hadoop, así que accedemos a él usando el comando "su hduser". Esta configuración la realizamos en todas las Raspberry Pi.

```
pi@worker1:~ $ sudo addgroup hadoop
Añadiendo el grupo `hadoop' (GID 1001) ...
Hecho.
pi@worker1:~ $ sudo adduser --ingroup hadoop hduser
Añadiendo el usuario `hduser' ...
Añadiendo el nuevo usuario `hduser' (1001) con grupo `hadoop' ...
Creando el directorio personal `/home/hduser' ...
Copiando los ficheros desde `/etc/skel' ...
Nueva contraseña:
Vuelva a escribir la nueva contraseña:
passwd: contraseña actualizada correctamente
Cambiando la información de usuario para hduser
Introduzca el nuevo valor, o pulse INTRO para usar el valor predeterminado
        Nombre completo []:
        Número de habitación []:
        Teléfono del trabajo []:
        Teléfono de casa []:
        Otro []:
¿Es correcta la información? [S/n] s
pi@worker1:~ $ sudo adduser hduser sudo
Añadiendo al usuario `hduser' al grupo `sudo' ...
Añadiendo al usuario hduser al grupo sudo
Hecho.
```

Ilustración 63: Creación del usuario "hduser"

Una vez creado los usuarios en cada Raspberry Pi, necesitamos configurar ssh en "master", ya que cuando el master inicia el clúster, inicia sesiones SSH con cada uno de los esclavos. Para evitar que cada vez que se establezcan cada una de esas conexiones, vamos a usar un sistema de clave pública y privada que evitarán que se pregunte por ningún tipo de contraseña cada vez que se use esa sesión SSH. Vamos a usar para ello el algoritmo ed25519. Todo esto lo realizamos de la siguiente forma.

Primero creamos en todas las Raspberry Pi el directorio "/home/hduser/.ssh". En "master" creamos y editamos el fichero "~/.ssh/config" con los hostname, usuario e IP de cada Raspberry Pi como podemos ver en la Ilustración 64.

```
  GNU nano 3.2                  /home/hduser/.ssh/config

Host master
User hduser
Hostname 192.168.1.10

Host worker1
User hduser
Hostname 192.168.1.20

Host worker2
User hduser
Hostname 192.168.1.30

Host worker3
User hduser
Hostname 192.168.1.40
```

Ilustración 64: Configuración del fichero "~/.ssh/config"

Para las claves públicas y privadas tenemos que generar un par de ellas en cada Raspberry Pi. Para ello ejecutamos el comando "ssh-keygen -t ed25519" en el directorio "/home/hduser/.ssh/" en cada Raspberry Pi. Nos generará dos archivos, "id_ed25519" que será la clave privada, y "id_ed25519.pub" que será la clave pública. Ahora tenemos que copiar la clave pública generada en una Raspberry Pi en en el resto de las Raspberry Pi, y así para cada una. Esto se hace copiando esa clave en el fichero "~/.ssh/authorized_keys". Por ejemplo, para "master" tendremos que ejecutar:

- "cat ~/.ssh/id_ed25519.pub >> ~/.ssh/authorized_keys": Se copia su propia clave pública porque también establece una conexión SSH consigo mismo.
- "cat ~/.ssh/id_ed25519.pub | ssh hduser@192.168.1.20 'cat >> .ssh/authorized_keys'": Concatenamos la clave pública de "worker1" en el fichero "authorized_keys" de "master.
- "cat ~/.ssh/id_ed25519.pub | ssh hduser@192.168.1.30 'cat >> .ssh/authorized_keys'": Concatenamos la clave pública de "worker2" en el fichero "authorized_keys" de "master.
- "cat ~/.ssh/id_ed25519.pub | ssh hduser@192.168.1.40 'cat >> .ssh/authorized_keys'": Concatenamos la clave pública de "worker3" en el fichero "authorized_keys" de "master.

Esto mismo tenemos que hacerlo en cada Raspberry Pi, pero variando las direcciónes IP, por ejemplo "worker2" tendrá que copiarse su propia clave pública, y la de "master", "worker2" y "worker3", y así el resto.

Ahora tenemos que definir las variables de entorno necesarias para Hadoop y Java. Estas variables las tenemos que definir al final del fichero "~/.bashrc" y son las que podemos ver en la Ilustración 65. Para que los cambios sean efectivos tenemos que poner el comando "source ~/.bashrc". Lo hacemos también con todas las Raspberry Pi.

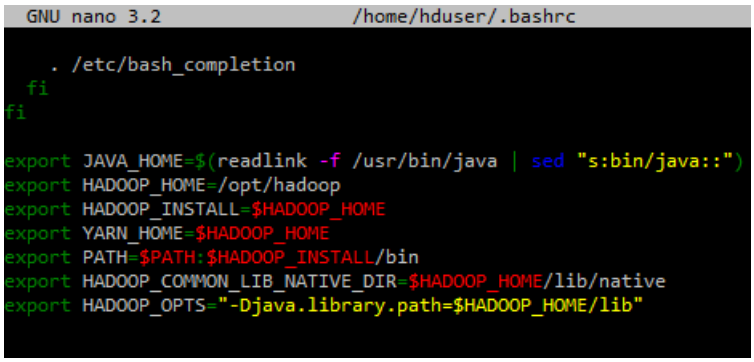


Ilustración 65: Definición de las variables de entorno de Java y Hadoop en el fichero "~/.bashrc"

Después de todo esto, ya sólo queda descargar Java 8. Hadoop 2.7.7 sólo soporta las versiones 7 u 8 de Java, por lo que nosotros usaremos la versión 8. Para descargarlo, sólo tenemos que ejecutar el comando "sudo apt install openjdk-8-jdk" en todas las Rasberry Pi. Una vez finalice la descarga e instalación, tenemos que seleccionar que se use Java 8 por defecto, ya que en Raspberry Pi OS viene Java 11 por defecto. Con el comando "sudo update-alternatives --config java" podemos cambiar la versión. Como podemos ver en la Ilustración 66, al ejecutarlo nos muestra las opciones disponibles y con el número que le corresponde a cada una podemos seleccionarla. Entonces, con el número 2, podemos seleccionar la versión 8.

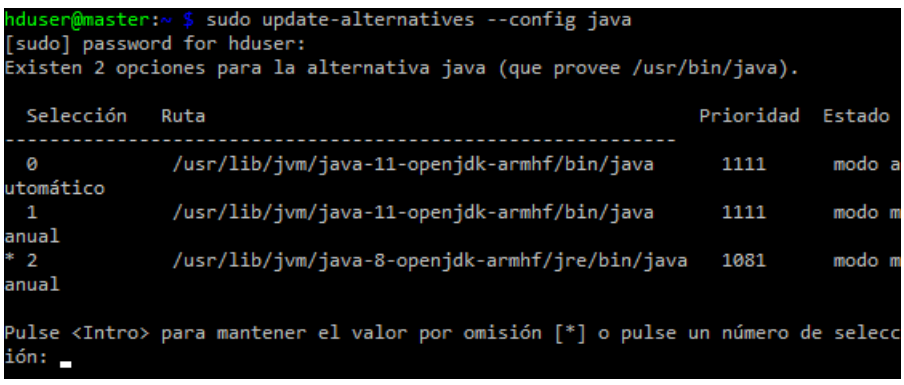


Ilustración 66: Selección de Java 8 por defecto

Una vez tengamos toda la configuración esencial para que nuestro clúster pueda funcionar correctamente, podemos empezar a descargar Hadoop e instalarlo. Para ello tenemos que descargar el fichero ".tar.gz" desde la página web oficial de Hadoop [52]. Como sabemos en este proyecto usaremos la versión 2.7.7 de Hadoop.

La instalación de Hadoop la realizaremos en el directorio "/opt/hadoop". Por tanto, con la sesión iniciada con el usuario "hduser" creado anteriormente creamos el directorio "/opt/hadoop" con el comando "sudo mkdir hadoop" en el directorio "/opt". A este directorio le cambiamos los permisos para el usuario "hduser" con el comando "sudo chown hduser:hadoop -R /opt/hadoop". En este directorio ya podemos realizar la descarga de Hadoop 2.7.7 con el comando "wget https://archive.apache.org/dist/hadoop/common/hadoop-2.7.7/hadoop-2.7.7.tar.gz". Con introducirlo en una de las Raspberry Pi basta, ya que podemos copiarlo en el resto de Raspberry Pi, además que ahorrará tiempo, ya que es un archivo de unos 200 MB, pero que dada la velocidad de descarga puede tardar unos 5 minutos en completar la descarga, y una descarga simultánea en las cuatro Raspberry Pi ralentizaría este proceso.

En nuestro caso hemos usado la Raspberry Pi "master" para realizar la descarga. Creamos los directorios "/opt/hadoop/" en el resto de Raspberry Pi y cambiamos los permisos y una vez termine de descargarse, desde "master" utilizamos el comando "scp hadoop-2.7.7.tar.gz hduser@worker1:/opt/hadoop". Sustituimos "worker1", por "worker2" y por "worker3" y así tendremos el fichero copiado en las cuatro Raspberry Pi.

Como el archivo descargado es un archivo comprimido en formato ".tar.gz" tenemos que descomprimirlo. Para ello usamos el comando "tar -xvzf /opt/hadoop/hadoop-2.7.7.tar.gz". Este comando nos descomprimirá el contenido en una carpeta llamada "hadoop-2.7.7", pero queremos la ubicación "/opt/hadoop", por tanto, copiamos el contenido a esa carpeta con el comando "cp -a /opt/hadoop/hadoop-2.7.7/. /opt/hadoop/". Por último, borramos los ficheros que ya no necesitaremos, que son "/opt/hadoop/hadoop-2.7.7.tar.gz" y la carpeta "/opt/hadoop/hadoop-2.7.7". Con esto ya tendremos instalado Hadoop en nuestras cuatro Raspberry Pi.

Ahora tenemos que realizar la configuración del clúster. Vamos a comenzar por el fichero "/opt/hadoop/etc/hadoop/slaves". Este fichero definirá los esclavos del clúster, por tanto, solo debemos de modificarlo en el "master", borrar su contenido y escribir el nombre de los 3 nodos esclavos en líneas separadas.

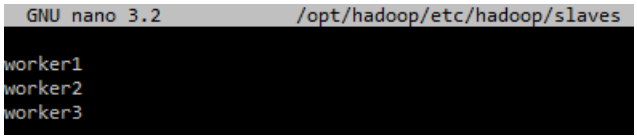


Ilustración 67: Configuración del fichero "/opt/hadoop/etc/hadoop/slaves"

Ahora vamos a configurar los siguientes 4 ficheros en todas las Raspberry Pi :

- "opt/hadoop/etc/hadoop/core-site.xml" [53]: En este fichero tenemos que introducir la dirección IP y el puerto del NameNode. En este caso el NameNode es las Raspberry Pi a la que hemos llamado "master" y el puerto que se usa es el número 9000. Para esto tenemos el parámetro "fs.defaultFS" se define la dirección del NameNode y el puerto al que los DataNodes enviarán los "heartbeats" al NameNode. Además, este parámetro nos permitirá usar los comandos de HDFS sin tener que introducir la dirección completa de un directorio en HDFS. Por ejemplo, para un "ls" pasamos de "hdfs dfs -ls hdfs://hdfs/nombre_directorio" a "hdfs dfs – ls nombre_directorio" Por tanto, el fichero quedaría como podemos ver en la Ilustración 68.

```
<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://master:9000/</value>
    </property>
</configuration>
```

Ilustración 68: Configuración del fichero "core-site.xml" de Hadoop

- "/opt/hadoop/etc/hadoop/hdfs-site.xml" [54]: Este fichero contiene la configuración para los demonios de HDFS, que son los NameNodes, DataNodes y Secondary NameNode. Los parámetros que vamos a configurar son los siguientes:
    - "dfs.datanode.data.dir": Especifica el directorio donde los DataNode deben almacenar los bloques de información. Para ello crearemos el el directorio "/opt/hadoop_env", le asignamos los permisos para hduser y creamos en él los directorios "/opt/hadoop_env/hdfs/datanode". Esto lo hacemos en "worker1", "worker2" y "worker3"
    - "dfs.namenode.name.dir": Especifica el directorio donde el NameNode debe almacenar el fichero "fsImage". Para ello crearemos el el directorio "/opt/hadoop_env", le asignamos los permisos para hduser y creamos en él los directorios "/opt/hadoop_env/hdfs/namenode". Esto lo hacemos en "master".
    - "dfs.replication": Especifica el número de réplicas que se van a realizar por cada bloque de información.
    - "dfs.namenode.http-address": Contiene la dirección IP y el puerto donde va a estar escuchando la interfaz web de Hadoop.
    - "dfs.namenode.secondary.http-address": Contiene la dirección IP y el puerto donde se va a ejecutar el Secondary NameNode.

  Por tanto, sabiendo esto, este fichero quedaría como podemos ver en la Ilustración 69.

```
<configuration>
    <property>
        <name>dfs.datanode.data.dir</name>
        <value>/opt/hadoop_tmp/hdfs/datanode</value>
    </property>
    <property>
        <name>dfs.namenode.name.dir</name>
        <value>/opt/hadoop_tmp/hdfs/namenode</value>
    </property>
    <property>
        <name>dfs.replication</name>
        <value>3</value>
    </property>
    <property>
        <name>dfs.namenode.http-address</name>
        <value>master:50070</value>
    </property>
    <property>
        <name>dfs.namenode.secondary.http-address</name>
        <value>master:50090</value>
    </property>
</configuration>
```

Ilustración 69:Configuración del fichero "hdfs-site.xml" de Hadoop

- "/opt/hadoop/etc/hadoop/yarn-site.xml" [55]: Este fichero contiene la configuración necesaria para Yarn. En este caso vamos a configurar 5 parámetros que son los siguientes:
    - "yarn.resourcemanager.resource-tracker.address": La dirección y puerto de la interfaz del ResourceTracker.
    - "yarn.resourcemanager.scheduler.address": La dirección y puerto de la interfaz del Scheduler.
    - "yarn.resourcemanager.address": La dirección y puerto de la interfaz del Application Manager en el ResourceManager.
    - "yarn.nodemanager.aux-services" y "yarn.nodemaganer.aux-services.mapreduce_shuffle.class" que sirven para definir cual es la implementación o el servicio que vamos a usar para la operación "shuffle" de MapReduce

  Así, el fichero quedará como podemos ver en la Ilustración 70.

```
<configuration>
    <property>
        <name>yarn.resourcemanager.resource-tracker.address</name>
        <value>master:8031</value>
    </property>
    <property>
        <name>yarn.resourcemanager.scheduler.address</name>
        <value>master:8030</value>
    </property>
    <property>
        <name>yarn.resourcemanager.address</name>
        <value>master:8032</value>
    </property>
    <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>
```

Ilustración 70: Configuración del fichero "yarn-site.xml" de Hadoop

- "/opt/hadoop/etc/hadoop/mapred-site.xml" [56]: Este fichero tenemos que copiarlo de otro que ya vienen creado, que es el fichero "/opt/hadoop/etc/hadoop/mapred-site.xml.template". Por tanto, lo copiamos con el comando "cp /opt/hadoop/etc/hadoop/mapred-site.xml.template opt/hadoop/etc/hadoop/mapred-site.xml". El fichero contiene la configuración necesaria para MapReduce. En este caso hemos definido la variable "mapred.framework.name". Esta variable define el framework que se usará para realizar las tareas MapReduce, que será Yarn. El fichero quedaría como en la Ilustración 71.

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

Ilustración 71: Configuración del fichero "mapred-site.xml" de Hadoop

Por último, definimos el entorno de java en el fichero "/opt/hadoop/etc/hadoop/hadoop-env.sh":

```
 GNU nano 3.2          /opt/hadoop/etc/hadoop/hadoop-env.sh

# The only required environment variable is JAVA_HOME.  All others are
# optional.  When running a distributed configuration it is best to
# set JAVA_HOME in this file, so that it is correctly defined on
# remote nodes.

# The java implementation to use.
export JAVA_HOME=$(readlink -f /usr/bin/java | sed "s:bin/java::")

# The jsvc implementation to use. Jsvc is required to run secure datanodes
# that bind to privileged ports to provide authentication of data transfer
# protocol.  Jsvc is not required if SASL is configured for authentication of
# data transfer protocol using non-privileged ports.
#export JSVC_HOME=${JSVC_HOME}
```

Ilustración 72: Configuración en el fichero "hadoop-env.sh"

Una vez hayamos completado todos estos pasos, ya tendremos completamente configurado nuestro clúster Hadoop y podemos iniciarlo. Primero ejecutamos "hdfs namenode -format". Una vez ejecutado tenemos que

iniciar HDFS con los scripts “/opt/hadoop/sbin/start-dfs.sh y Yarn con “/opt/hadoop/sbin/start-yarn.sh”.

Para comprobar el correcto funcionamiento del clúster podemos ejecutar el comando “/opt/hadoop/bin/hdfs dfsadmin -report”, cuya salida podemos ver en la Ilustración 73. En esta Ilustración 73 podemos ver al inicio la capacidad del clúster y a continuación información de cada uno de sus nodos esclavos, como su dirección IP, puerto usado, capacidad disponible o usada, así como otros datos.

```
hduser@master:~ $ /opt/hadoop/bin/hdfs dfsadmin -report
20/11/29 21:17:05 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
Configured Capacity: 84607598592 (78.80 GB)
Present Capacity: 57752526848 (53.79 GB)
DFS Remaining: 57734893568 (53.77 GB)
DFS Used: 17633280 (16.82 MB)
DFS Used%: 0.03%
Under replicated blocks: 4
Blocks with corrupt replicas: 0
Missing blocks: 0
Missing blocks (with replication factor 1): 0

-------------------------------------------------
Live datanodes (3):

Name: 192.168.1.40:50010 (worker3)
Hostname: worker3
Decommission Status : Normal
Configured Capacity: 28468338688 (26.51 GB)
DFS Used: 5877760 (5.61 MB)
Non DFS Used: 9067290624 (8.44 GB)
DFS Remaining: 17925459968 (16.69 GB)
DFS Used%: 0.02%
DFS Remaining%: 62.97%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Sun Nov 29 21:17:08 CET 2020


Name: 192.168.1.20:50010 (worker1)
Hostname: worker1
Decommission Status : Normal
Configured Capacity: 27670921216 (25.77 GB)
DFS Used: 5877760 (5.61 MB)
Non DFS Used: 6624940032 (6.17 GB)
DFS Remaining: 19610894336 (18.26 GB)
DFS Used%: 0.02%
DFS Remaining%: 70.87%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Sun Nov 29 21:17:08 CET 2020


Name: 192.168.1.30:50010 (worker2)
Hostname: worker2
Decommission Status : Normal
Configured Capacity: 28468338688 (26.51 GB)
DFS Used: 5877760 (5.61 MB)
Non DFS Used: 6794211328 (6.33 GB)
DFS Remaining: 20198539264 (18.81 GB)
DFS Used%: 0.02%
DFS Remaining%: 70.95%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Sun Nov 29 21:17:08 CET 2020
```

Ilustración 73: Ejecución del comando “/opt/hadoop/bin/hdfs dfsadmin -report”

3. Instalación de Pig

La instalación de Pig [57] sólo será necesaria en la Raspberry Pi "master", ya que se encargará de realizar la tarea MapReduce. La versión que usaremos será la última versión de Pig lanzada en 2017 que es la 0.17.0.

Para comenzar la instalación abrimos una conexión ssh con el ordenador portátil y el nodo "master". Iniciamos sesión con el usuario "hduser" anteriormente creado con el comando "su hduser". A continuación, vamos a crear el directorio donde tendremos instalado PIG, este directorio será "/usr/lib/pig". Por tanto, lo cremos con el comando "sudo mkdir /usr/lib/pig" y cambiamos los permisos con "sudo chown hduser:hadoop /usr/lib/pig".

Una vez creado el directorio de instalación, accedemos a él y descargamos Pig desde la web de Apache con el comando "wget https://archive.apache.org/dist/pig/pig-0.17.0/pig-0.17.0.tar.gz". Esto nos descargará un archivo comprimido en formato ".tar.gz". Lo descomprimimos con "tar -xvf pig-0.17.0.tar.gz" y borramos el archivo comprimido que ya no necesitaremos con el comando "rm pig-0.17.0.tar.gz".

Con esto ya tendríamos instalado Pig en nuestra Raspberry Pi "master", pero para poder usarlo necesitamos definir las variables de entorno de Pig en el fichero "~/.bashrc". Por tanto, lo editamos y añadimos lo siguiente:

```
export PIG_HOME="/usr/lib/pig/pig-0.17.0"
export PIG_CONF_DIR="$PIG_HOME/conf"
export PIG_CLASSPATH="$PIG_CONF_DIR"
export PATH="$PIG_HOME/bin:$PATH"
```

Ilustración 74: Definición variables de entorno de Pig en el fichero "~/.bashrc"

Podemos comprobar que está correctamente instalado podemos escribir "pig -version":

```
hduser@master:/ $ pig -version
Apache Pig version 0.17.0 (r1797386)
compiled Jun 02 2017, 15:41:58
```

Ilustración 75: Ejecución del comando "pig -version"

4. Configuración MPU6050

Para poder obtener la información del MPU6050, tenemos que tenerlo conectado a los pines GPIO de la Raspberry Pi como se indicaba en el apartado "Diseño de la solución", y seguir los siguientes pasos.

Los datos se obtendrán usando un script de Python. Al haber instalado Raspbian OS usando NOOBS ya tenemos instalado Python y el resto de componentes necesarios para hacer que el sensor funcione. Estos componentes son la librería de GPIO y la librería de smbus. Lo único que tenemos que hacer es habilitar la interfaz I2C ejecutando el comando "sudo raspi-config" y accedemos a "Interfacing Options".

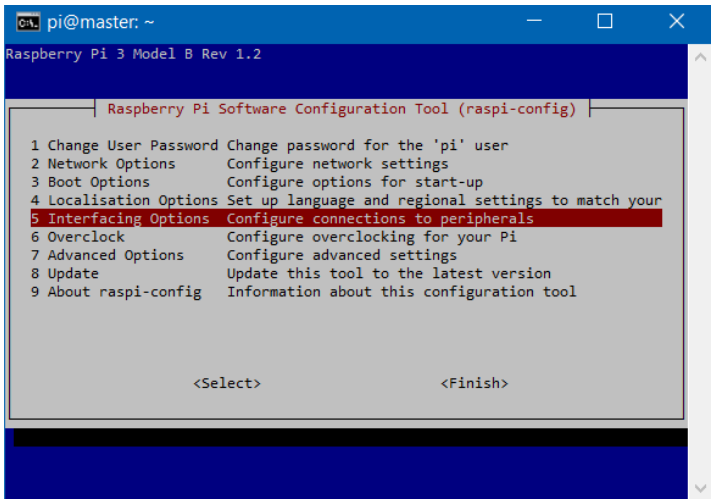


Ilustración 76: Selección de “Interfacing Options”

A continuación, nos vamos a “I2C”.

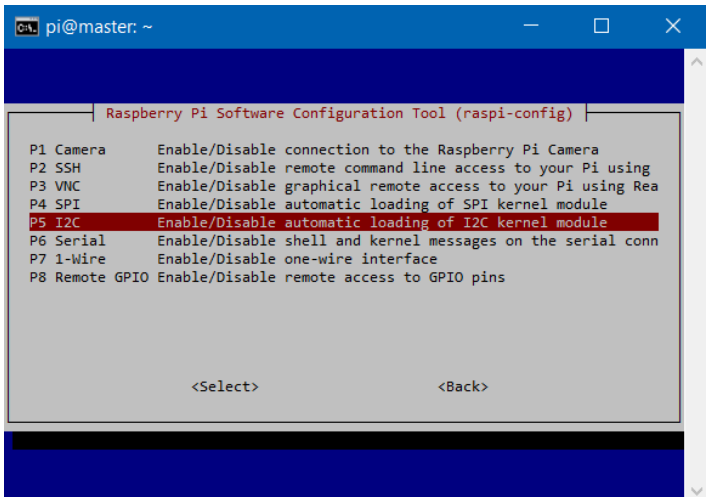


Ilustración 77: Selección de “I2C”

Y finalmente, hacemos click en “Yes”. Reiniciamos, instalamos la librería del mpu6050 con “sudo pip install mpu6050-raspberrypi” y ya tendremos todo listo para poner en funcionamiento el sensor.

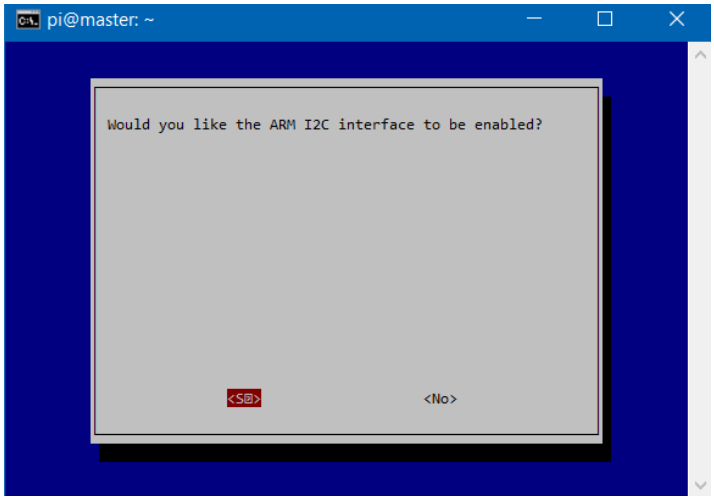


Ilustración 78: Selección de "Yes" para habilitar I2C

Para la lectura de los datos necesitaremos un script [58] [59] [60]. En este caso hemos elegido el lenguaje Python, y crearemos el script "mpu.py". El MPU6050 tiene unos registros y direcciones que necesitaremos. Estos registros serán almacenados en variables como se puede ver en el código en la Ilustración 79. A continuación, se definen dos funciones llamadas "MPU_init" y "read_raw_data". La primera se encargará de inicializar los registros necesarios para el funcionamiento del sensor. La segunda se encargará de la lectura de los valores "en bruto" (raw) de los sensores.

Los datos obtenidos por los sensores consisten en 16 bits en 2 componentes para el giroscopio y el acelerómetro, y de 16 bits pero no en 2 componentes para el sensor de temperatura. Es por eso que en el caso de los sensores del giroscopio y del acelerómetro, primero restamos el valor obtenido menos 65536, y después lo dividimos por un Factor de escala de sensibilidad. Este factor es de 16384 para el giroscopio y de 131 para el acelerómetro. Los valores de la temperatura también necesitan ser divididos por un factor de escala de sensibilidad de 240 y luego sumarle 36.53 para obtener el valor de la temperatura en grados centígrados.

Tras la lectura de los datos, los resultados se imprimen por la salida estándar. El tiempo entre lecturas lo definimos con la función "sleep()", que en este caso es de 0.1 segundos. El resultado completo del script lo tenemos en la Ilustración 79.

```
import smbus
from time import sleep
import sys

#Definimos los registros y direcciones que necesitaremos
PWR_MGMT_1   = 0x6B
SMPLRT_DIV   = 0x19
CONFIG       = 0x1A
GYRO_CONFIG  = 0x1B
INT_ENABLE   = 0x38
ACCEL_XOUT_H = 0x3B
ACCEL_YOUT_H = 0x3D
ACCEL_ZOUT_H = 0x3F
GYRO_XOUT_H  = 0x43
GYRO_YOUT_H  = 0x45
GYRO_ZOUT_H  = 0x47
TEMP = 0x41

def MPU_Init():
        #Escribimos en el registro sample rate
        bus.write_byte_data(Device_Address, SMPLRT_DIV, 7)

        #Escribimos en el registro power management
        bus.write_byte_data(Device_Address, PWR_MGMT_1, 1)

        #Escribimos en el registro Configuration
        bus.write_byte_data(Device_Address, CONFIG, 0)

        #Escribimos en el Gyro configuration
        bus.write_byte_data(Device_Address, GYRO_CONFIG, 24)

        #Escribimos en el registro interrupt
        bus.write_byte_data(Device_Address, INT_ENABLE, 1)

def read_raw_data(addr):
        #Los valores del acelerometro y del giroscopio son de 16 bits
        high = bus.read_byte_data(Device_Address, addr)
        low = bus.read_byte_data(Device_Address, addr+1)

        #Concatenamos valores altos y bajos
        value = ((high << 8) | low)

        #Obtenemos el valor firmado del mpu6050
        if(value > 32768):
                value = value - 65536
        return value


bus = smbus.SMBus(1)    # o bus = smbus.SMBus(0)
Device_Address = 0x68   # Direccion del MPU6050

MPU_Init()

while True:

        #Leemos los datos raw del acelerometro
        acc_x = read_raw_data(ACCEL_XOUT_H)
        acc_y = read_raw_data(ACCEL_YOUT_H)
        acc_z = read_raw_data(ACCEL_ZOUT_H)

        #Leemos los valores raw del giroscopio
        gyro_x = read_raw_data(GYRO_XOUT_H)
        gyro_y = read_raw_data(GYRO_YOUT_H)
        gyro_z = read_raw_data(GYRO_ZOUT_H)

        #Rango de escala completa +/- 250 grados celsius segun el factor de escala de sensibilidad
        Ax = acc_x/16384.0
        Ay = acc_y/16384.0
        Az = acc_z/16384.0

        Gx = gyro_x/131.0
        Gy = gyro_y/131.0
        Gz = gyro_z/131.0

        #Leemos los valores raw del sensor de temperatura
        tempRow=read_raw_data(TEMP)
        tempC=(tempRow / 340.0) + 36.53

        print("%.2f" %Gx + "\t%.2f" %Gy + "\t%.2f" %Gz + "\t%.2f" %Ax + "\t%.2f" %Ay + "\t%.2f" %Az + "\t%.2f" %tempC)
        sleep(0.1)
```

Ilustración 79: Script "mpu.py" [59]