Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Instalación y configuración de herramientas software para Big Data

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Ignacio Davó Escrivá

Tutor: Jon Ander Gómez Adrian

2015/2016

Resumen

Este Trabajo Fin de Grado consiste en la instalación y configuración de dos de las herramientas para almacenamiento y procesamiento de grandes cantidades de datos, que se están convirtiendo en los líderes en software libre para Big Data, Hadoop y Spark. Estas herramientas proporcionan, tanto de forma conjunta como por separado, extraordinarias capacidades de almacenamiento y procesamiento de datos, a la vez que proporcionan características como escalabilidad, distribución, seguridad, redundancia, tolerancia a fallos, etc.

Palabras clave: instalación, configuración, Big Data, Hadoop, Spark, almacenamiento, datos, sistemas distribuidos.

Abstract

This Degree Final Project involves the installation and configuration of two of the middleware tools for storing and processing large amounts of data, which are quickly becoming the leaders in free software for Big Data, Hadoop and Spark. These tools provide both, together or standalone, extraordinary storage capacity and data processing, while providing characteristics such as scalability, distribution, security, redundancy, fault tolerance, etc.

Keywords : installation, configuration, Big Data, Hadoop, Spark, storage, data, distributed systems.

Índice

1. Introducción

> "Si se pusieran en fila todos los nuevos libros publicados, nos deberíamos desplazar a 150 kilómetros por hora para mantenernos al frente de la hilera" (Stephen Hawking) (1)

Actualmente, vivimos en un mundo altamente demandante de almacenamiento de datos. La forma en que los usuarios interactúan con la tecnología debido, fundamentalmente, a la rápida y constante evolución de la misma, el rápido crecimiento del uso de los teléfonos inteligentes y sus aplicaciones (Apps), a la utilización exhaustiva de las redes sociales (con imágenes y videos de alta resolución), al almacenamiento de todo tipo de documentos y a la explosión del Internet de las Cosas (IoT)[1], están incrementando esa demanda día a día. Además, la creciente generación de datos de todo tipo por parte de las empresas e instituciones, así como el almacenamiento de datos generados por parte de diferentes dispositivos, hace que el crecimiento de los volúmenes de datos que necesitan ser almacenados sea exponencial.

Las estadísticas indican que en el año 2010, existían del entorno de 80×10^6 dispositivos M2M[2]. La "AAA" (Asociación de Electrónica de Consumo) espera que en 2015 sean 25×10^9 dispositivos M2M y que en 2020 la cifra alcance los 50 billones americanos. Además, según la misma fuente, para el año 2017 existirán 42 millones de coches inteligentes en el mundo, en 2020 serán 85 millones y su cifra no parará de crecer. (2)

Por otro lado, algunas fuentes hablan de que en el año 2020, 1/3 de los datos mundiales estarán almacenados o pasarán por la nube en algún momento. Además, para dicho año la producción mundial de datos será 44 veces superior a lo que era en el año 2009. De todo ese volumen de datos, los particulares crearán el 70% de los mismos, pero será responsabilidad de las empresas el almacenamiento del 80% de esos datos, o bien porque son propietarias de los mismos o porque los particulares utilizan sus sistemas de almacenamiento para hacerlo (Google Drive, DropBox, Facebook, etc.). (3)

De hecho, algunas de las fuentes también dan las siguientes cifras como muestra de la magnitud de la evolución de la cantidad de información generada:

- El 90% de los datos mundiales ha sido generado en los dos últimos años. (4)
- El 90% de los datos generados por los diferentes dispositivos (smartphones, tablets, vehículos, etc.) nunca llegan a ser analizados y el 60% de estos datos empiezan a estar obsoletos en milisegundos. (4)
- Todos los días creamos 2.5 trillones de bytes en datos. (5)
- "En los últimos cinco años, se ha generado más información científica que en toda la historia de la humanidad". (6)
- Los científicos del CERN[3] pueden generar 40 terabytes de datos por segundo durante la experimentación con el Gran Colisionador de Hadrones (LHC). (7)

[1] IoT (Internet of Things): Concepto referido a la interconexión de objetos de uso diario con internet para su gestión y control o para proporcionar información.
[2] M2M: del Ingles Machine To Machine (máquina a máquina), intercambio de información entre dos máquinas remotas.
[3] CERN: Organización Europea para la Investigación Nuclear.

- El motor de un avión *Boeing*, puede producir hasta 10 terabytes de información de funcionamiento por cada 30 minutos de vuelo. Esto significa que un *Jumbo* con cuatro motores genera 640 terabytes de datos cada vez que cruza el Atlántico. (7)
- Twitter gestiona más de 200 millones de usuarios que generan más de 90 millones de tuits diarios, esto produce más de 8 terabytes diarios de información. (7)
- Facebook tiene más de 800 millones de usuarios que generan infinidad de tipos diferentes de información (links, fotos, textos, etc.). Esto genera mensualmente más de 30 billones de elementos de información diferentes. (7)

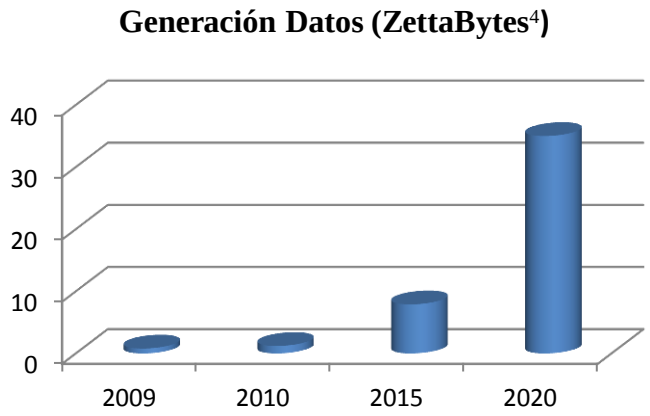


Tabla 1 - Evolución creación datos globales

Todos estos volúmenes de datos hacen que los expertos resuman todos los aspectos del Big Data en las cinco V´s. Inicialmente fueron tres V´s y luego fueron ampliadas a cuatro o cinco V´s en función del grupo de expertos consultado. Las cinco V´s principales son:

- **Volumen**: El tamaño de los datos que se generan cada día. Los sistemas de Big Data son capaces de almacenar grandes cantidades de datos en sistemas escalables y distribuidos.
- **Velocidad**: Se refiere tanto a la velocidad a la que se crean los datos como a la velocidad en que se propagan entre los usuarios. También se refiere a la velocidad en que estos datos pierden su valor.
- **Variedad**: Actualmente los datos proceden de muy diversas fuentes como sensores, smartphones, cámaras, vehículos, redes sociales, documentos, etc., por lo que los sistemas de Big Data necesitan ser capaces de procesar y almacenar datos en múltiples y diferentes formatos (estructurados, no estructurados, etc.).
- **Veracidad**: Los datos llegan en "bruto", en diferentes formatos y en ocasiones incompletos y las tecnologías Big Data deben ser lo suficientemente flexibles como para procesar y reflejar de manera fiel estos datos.
- **Valor**: La capacidad para convertir todas esas cantidades de datos en algo de "valor" para las propias empresas.

[4] Un ZettaByte es igual a 10^{12} GB o 10^{9} TB de información.

1.1. Motivación

Creo que uno de los mayores retos actuales y futuros para la tecnología va a ser el manejo y la gestión de los grandes volúmenes de datos que se generan. Estos volúmenes ingentes de datos van a necesitar ser almacenados de manera que los usuarios puedan acceder a ellos casi en cualquier momento y desde cualquier lugar. Además, a medida que pase el tiempo irá cobrando más importancia la capacidad y facilidad para recuperar dichos datos y poder filtrar, analizar y eliminar aquellos datos superfluos o que no aportan ninguna información de valor.

Hace poco alguien me dijo que, cualquier administrador de sistemas que se precie, debe desenvolverse con soltura manejando el sistema operativo Linux en su modo de comandos y, en especial, su editor Vi[5]. Pues bien, yo creo que cualquier Arquitecto de Sistemas Informáticos del presente y sobre todo del futuro, debe ser capaz de implementar, configurar, optimizar y administrar sistemas de almacenamiento para grandes volúmenes de datos en configuraciones de alta disponibilidad.

Por todo ello, existe una creciente demanda de profesionales especializados en la gestión, el almacenamiento y el análisis de grandes volúmenes de datos hasta el punto de que la Comisión Europea lo ha incluido como uno de sus Objetivos Estratégicos en la Agenda Digital para Europa en su programa de financiación de proyectos de investigación e innovación Horizonte 2020.

1.2. Objetivos del Proyecto

El presente proyecto tiene como principal objetivo la implementación y configuración de un clúster[6] de Big Data en uno de los laboratorios de la ETSINF (concretamente el laboratorio Anita Borg del edificio 1G), utilizando para ello diferentes herramientas middleware[7] como son Apache Hadoop y Apache Spark.

Para ello se utilizarán dos de los equipos pertenecientes al laboratorio como Maestros del sistema de Big Data y el resto de los equipos existentes en dicho laboratorio como Workers del mismo según el esquema representado en la Ilustración 1.

Existen 2 equipos actuando como Maestros para coordinación, configuración y administración. Uno de ellos como Master para la gestión de los nodos (NameNode) y el otro como Master Secundario para la gestión de recursos (ResourceManager). El resto de equipos del laboratorio se configura como trabajadores (Slaves) para utilizar sus capacidades para realizar todos los trabajos, tanto los de almacenamiento (con Hadoop) como los de computo (Spark).

[5] Vi: Editor de textos básico incorporado en las diferentes versiones de Linux
[6] Clúster: Conjunto de ordenadores que se comportan como si fueran uno solo.
[7] Middleware: Niveles de software y servicios que existen entre las aplicaciones “de usuario” y las aplicaciones de Sistema Operativo, aportando una capa de abstracción de software distribuida.

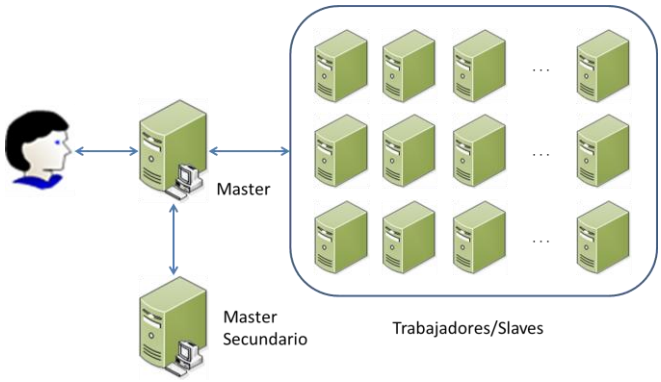


Imagen 1 - Esquema de Clúster Hadoop

Los equipos utilizarán todos la misma configuración y entre las opciones de Hadoop, elegimos la de persistencia en 3 de los nodos del clúster, es decir, cada elemento de información estará replicado en 3 de los nodos del clúster de manera que garantizamos la persistencia en caso de fallo de uno de los trabajadores/slaves.

2. Planificación Proyecto

La planificación de un proyecto es una de las partes más importantes del mismo, y en ella se detalla cuáles son las tareas que se van a realizar y cuáles son los tiempos y costes estimados para las mismas. Realizar una planificación del proyecto es absolutamente necesario para poder controlar la evolución y ejecución del proyecto y poder determinar las posibles desviaciones existentes. En base a la planificación, podremos tomar medidas correctoras (tanto en cuestión de tiempo como en cuestiones económicas) que nos eviten "sorpresas" el ultimo día.

La planificación inicial del proyecto contempla cinco tareas básicas que se desglosan en la tabla siguiente. Su duración se ha estimado en horas, ya que su traslación a días es compleja al no tener una dedicación total al proyecto. Las dos tareas que cobran mayor importancia (en cuanto a tiempo) son las de instalación y configuración del sistema y la redacción del proyecto.

Tarea		Duración
Tarea 1	**Preparación y planificación**	**15**
	Preparación y planificación del proyecto	15
Tarea 2	**Documentación inicial**	**50**
	Documentación sobre Big Data	10
	Documentación sobre Hadoop	20
	Documentación sobre Spark	20
Tarea 3	**Instalación y Configuración**	**105**
	Instalación Hadoop	20
	Instalación Spark	20
	Comprobación funcionamiento	25
	Resolución incidencias	40
Tarea 4	**Pruebas y casos de uso**	**54**
	Pruebas caso de uso 1	12
	Pruebas caso de uso 2	12
	Resolución incidencias y conclusiones	30
Tarea 5	**Redacción proyecto**	**140**
	Busqueda documentación adicional	25
	Realización tablas e imágenes	20
	Redacción proyecto	70
	Revisión y corrección	25

Tabla 2 - Tabla de tareas

En esta planificación del proyecto no se ha tenido en cuenta ningún tipo de coste económico debido a los siguientes motivos:

- Se van a utilizar instalaciones y equipamientos de la ETSINF, por lo que no es necesario contemplar el coste ya que se utilizan de forma gratuita (en otras condiciones, sería conveniente establecer tanto el coste de los equipamientos e infraestructuras como los costes de uso de los mismos).

- Todos los productos software que se van a utilizar son de software libre (tanto las versiones de Linux como Hadoop y Spark), por lo que no se debe realizar ningún tipo de gasto por su utilización.
- La parte correspondiente de coste de recursos humanos por las horas de dedicación deberían tenerse en cuenta si se tratara de una empresa o institución pública para su correcta imputación, pero considero que no es necesario en este caso.

Sin embargo, la realidad de la ejecución del proyecto ha sido un poco diferente, ya que se han ido aprovechando algunos momentos entre tareas para iniciar la redacción del mismo, por lo que la última tarea de redacción del proyecto se ha ido solapando en el tiempo de manera que, a medida que se finalizaba cada una de las tareas, se realizaba la redacción del apartado oportuno en la memoria del proyecto. También ha ocurrido esto con las tareas relacionadas con la búsqueda de documentación e información, que han sido realizadas en ocasiones de forma simultánea a otras tareas relacionadas.

Además, alguna de las tareas indicadas en la planificación inicial, a pesar de necesitar el tiempo estimado, han tenido que ser retrasadas algunos días, por lo que la finalización del proyecto ha sido un poco más tardía de lo previsto inicialmente. Los motivos en los retrasos de algunas tareas han sido de todo tipo pero, en general, por causas ajenas a la ejecución del mismo, como pueden ser necesidades de tiempo para estudio de otras asignaturas, tiempo para labores profesionales o temas personales diversos.

	Nombre de tarea	Duración	Comienzo	Fin
1	**Preparación y planificación**	**15 horas**	**lun 21/12/15**	**mar 22/12/15**
2	Preparación y planificación del proyecto	15 horas	lun 21/12/15	mar 22/12/15
3	**Documentación Inicial**	**60 horas**	**mar 22/12/15**	**lun 18/01/16**
4	Documentación sobre Big Data	10 horas	mar 22/12/15	jue 24/12/15
5	Documentación sobre Hadoop	20 horas	jue 24/12/15	mar 12/01/16
6	Documentación sobre Spark	20 horas	mar 12/01/16	vie 15/01/16
7	Redacción proyecto	10 horas	vie 15/01/16	lun 18/01/16
8	**Instalación y Configuración**	**130 horas**	**lun 18/01/16**	**mar 23/02/16**
9	Instalación Hadoop	20 horas	lun 18/01/16	mié 20/01/16
10	Instalación Spark	20 horas	mié 20/01/16	lun 25/01/16
11	Comprobación funcionamiento	25 horas	lun 25/01/16	jue 28/01/16
12	Resolución incidencias	40 horas	jue 28/01/16	jue 04/02/16
13	Redacción proyecto	25 horas	jue 04/02/16	mar 23/02/16
14	**Pruebas y casos de uso**	**69 horas**	**mar 23/02/16**	**lun 07/03/16**
15	Pruebas caso de uso 1	12 horas	mar 23/02/16	jue 25/02/16
16	Pruebas caso de uso 2	12 horas	jue 25/02/16	vie 26/02/16
17	Resolución incidencias y conclusiones	30 horas	vie 26/02/16	jue 03/03/16
18	Redacción proyecto	15 horas	jue 03/03/16	lun 07/03/16
19	**Redacción proyecto**	**90 horas**	**lun 07/03/16**	**mar 22/03/16**
20	Busqueda documentación adicional	25 horas	lun 07/03/16	jue 10/03/16
21	Realización tablas e imágenes	20 horas	jue 10/03/16	lun 14/03/16
22	Redacción proyecto	20 horas	lun 14/03/16	jue 17/03/16
23	Revisión y corrección	25 horas	jue 17/03/16	mar 22/03/16

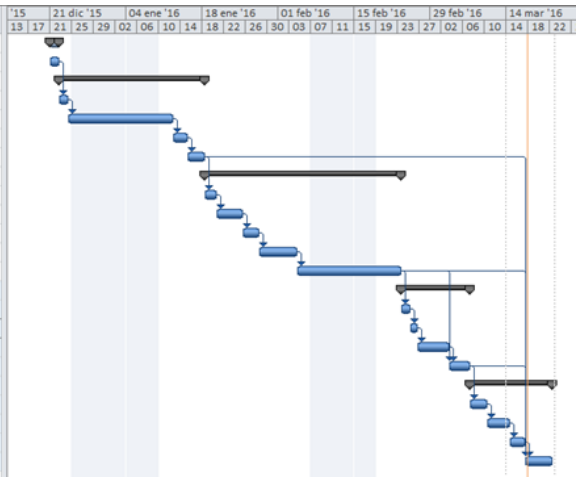


Tabla 3 - Diagrama de Gantt

3. Tecnologías

En este apartado, describiré las diferentes tecnologías que se van a utilizar en el proyecto y sus principales características.

3.1. Linux Mint 1.7

Todos los equipos del laboratorio están provistos de Linux Mint v. 17. "Qiana" Esta es una versión estable de Linux cuyo propósito es producir un sistema operativo que sea a la vez potente y sencillo de utilizar. Algunas de las razones del éxito de Linux Mint son (8):

- Tiene soporte multimedia completo y es sencillo de utilizar.
- Es gratuito y *Open Source*[8]
- Está impulsado por la comunidad de desarrolladores, aprovechando sus sugerencias e ideas para la mejora continua.
- Está basado en Debian y Ubuntu, y provee más de 30.000 paquetes de software.
- Es seguro y fiable, requiere muy poco mantenimiento.

3.2. Apache Hadoop 2.6.0

La librería de software Apache Hadoop es un framework [9] que permite el tratamiento y gestión distribuidos de grandes volúmenes de información. Está diseñado para ser escalable desde un pequeño número de máquinas hasta miles de ellas con alta disponibilidad. (9)

Los componentes del proyecto Hadoop son los siguientes:

- **Hadoop Common**: Utilidades comunes que dan soporte al resto de módulos.
- **Hadoop Distributed File System (HDFS)**: Sistema de archivos distribuido que ofrece un alto rendimiento en los accesos a los datos.
- **Hadoop Yarn**: Un framework para gestión de recursos y planificación de tareas.
- **Hadoop MapReduce**: Sistema para el procesamiento en paralelo de grandes volúmenes de información basado en Hadoop Yarn.

3.2.1 Hadoop Distributed File System

El HDFS (Hadoop Distributed File System) es el sistema de ficheros distribuido y escalable que da soporte a Hadoop (10). Está diseñado para ser ejecutado en hardware de bajo coste y para que, su conjunto, sea tolerante a fallos. Es muy similar a otros sistemas de ficheros distribuidos. Sus principales características, totalmente transparentes para el usuario, son:

- **Tolerancia a fallos**: Los sistemas de almacenamiento de Big Data pueden tener miles de máquinas trabajando en sus clústeres, lo que hace que los fallos de hardware sean la norma en lugar de la excepción, por ello, la

[8] Open Source: Código abierto. Software desarrollado y distribuido libremente.
[9] Framework: Estructura tecnológica de soporte que sirve de base para otras organizaciones tecnológicas o desarrollos superiores

detección y la rápida y automática recuperación de los mismos es un objetivo prioritario de la arquitectura HDFS.

- **Acceso a datos en streaming**[10]: En las aplicaciones en que se utiliza HDFS, es necesario el acceso con un rendimiento constante y elevado.
- **Grandes volúmenes de datos**: Un fichero típico de HDFS puede ser de varios gigabytes hasta varios terabytes de información, por lo que debe disponer de suficiente ancho de banda para la comunicación entre nodos.
- **"Mover la computación es más barato que mover los datos"**: Es más sencillo y minimiza la congestión de las redes de comunicaciones el trasladar las solicitudes de computación al lugar donde están los datos que trasladar los datos a los equipos que realizarán el computo.
- **Portabilidad**: El sistema HDFS ha sido diseñado para ser fácilmente portable de una plataforma a otra, aun siendo heterogéneas en cuanto a hardware o software.
- **Modelo simple**: Las aplicaciones HDFS utilizan un modelo de escritura única/lectura múltiple de ficheros.

HDFS utiliza una arquitectura de tipo maestro/esclavo. Un clúster HDFS consiste en un servidor maestro (NameNode) que gestiona el sistema de ficheros y regula el acceso a los mismos por parte de los clientes. Además, existen un numero de DataNodes (normalmente, uno por cada nodo existente en el clúster), que gestionan el almacenamiento en el nodo en el que residen.

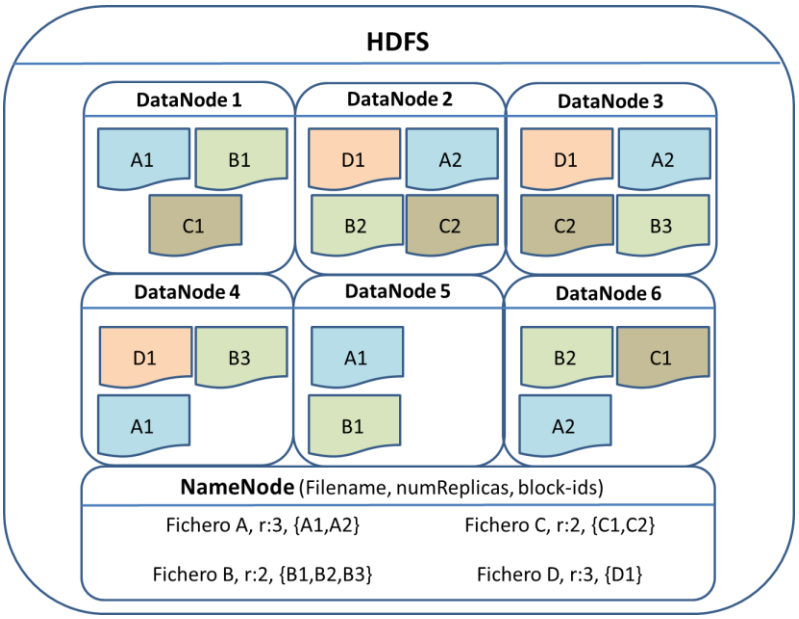


Imagen 2 - Replicación y división de ficheros en DataNodes y NameNode

[10] Streaming: Distribución digital en la que el usuario consume el producto descargado en paralelo (de forma simultánea) a su descarga.

Existen dos elementos básicos en la configuración de un sistema HDFS:

- **Tamaño de bloque**: indica el tamaño que debe tener cada bloque de un fichero (normalmente, 64 MB o 128 MB).
- **Factor replicación**: número de veces que cada bloque de un fichero se replica en disco.

El sistema NameNode se encarga de todos los trabajos relacionados con la detección de errores, los balanceos de carga y de datos, comprobación e integridad del sistema, etc. Para trabajar con el sistema HDFS se puede utilizar la línea de comandos, un interfaz web o una API[11] para aplicaciones Java.

En la versión 2.0 de HDFS se introdujeron algunos cambios para mejorar la alta disponibilidad del sistema de NameNode, para evitar su punto de fallo único al existir una única copia del NameNode, lo que en caso de fallo haría caer todo el sistema. Esto se puede conseguir por dos vías, el Quorum Journal Manager o utilizando el Network File System.

3.2.2 Map Reduce

MapReduce es un framework de software cuyo objetivo es la mejora del procesamiento de grandes volúmenes de datos trabajando sobre sistemas distribuidos. Está diseñado para trabajar con ficheros de gran tamaño con información tanto estructurada como no estructurada. (11)

El paradigma MapReduce se emplea para resolver algunos algoritmos que son susceptibles de ser paralelizados (12). Se basa en realizar el procesamiento de la información en el mismo lugar en que ésta reside. Al lanzar un proceso de MapReduce, las tareas son distribuidas entre los diferentes nodos del clúster. La parte de computación se realiza de forma local en el mismo nodo que contiene los datos, por lo que se minimiza el tráfico de datos por la red.

El framework de Hadoop se encarga tanto de la distribución de los datos a cada uno de los nodos del clúster como de la distribución de las tareas que estos tienen que realizar.

El término MapReduce en realidad se refiere a dos tareas separadas y distintas que los programas de Hadoop realizan. La primera de las tareas es la de Map, que toma un conjunto de datos y lo convierte en un nuevo conjunto donde los elementos individuales se transforman en tuplas (clave/valor). La segunda tarea Reduce, toma como entrada la salida de la tarea Map, combinando las tuplas de datos en conjuntos más pequeños de tuplas. La tarea Reduce siempre se realiza después de la tarea Map.

Veamos un ejemplo. Supongamos que tenemos dos archivos, y cada uno de ellos contiene dos columnas (una clave y un valor) que representan una ciudad y la

[11] API: Conjunto de subrutinas y funciones (métodos) que ofrece una biblioteca de software de aplicaciones para ser utilizada por otro software.

temperatura de esa ciudad en un día determinado. En la siguiente tabla tenemos el contenido de los archivos:

Archivo 1 - Nodo 1	
Clave	Valor
Barcelona	23
Valencia	29
Alicante	26
Barcelona	15
Madrid	21
Valencia	24
Sevilla	38

Archivo 2 - Nodo 2	
Clave	Valor
Valencia	18
Alicante	32
Barcelona	20
Sevilla	26
Valencia	23
Madrid	28
Sevilla	35

Tabla 4 - Tablas con tuplas (Clave,Valor)

Lo que queremos obtener como resultado, es la temperatura máxima para cada una de las ciudades de entre todos los archivos (cada ciudad puede aparecer en varios archivos y en cada archivo puede aparecer varias veces la misma ciudad). Podemos dividir el trabajo en dos tareas de Map (una para cada archivo) que devolverá la temperatura máxima para cada ciudad en ese archivo. En el ejemplo de nuestros archivos, el resultado será:

Archivo 1 - Nodo 1	
Clave	Valor
Barcelona	23
Valencia	29
Alicante	26
Madrid	21
Sevilla	38

Archivo 2 - Nodo 2	
Clave	Valor
Valencia	23
Alicante	32
Barcelona	20
Sevilla	35
Madrid	28

Tabla 5 - Resultado de función Map

Una vez finalizadas las dos tareas de Map sobre nuestros archivos, estos dos conjuntos de resultados serán la entrada para la tarea Reduce, que combinaría todas las entradas para obtener un único dato de salida para cada una de las ciudades.

El resultado de la función Reduce para estos dos archivos de datos lo podemos ver en la tabla siguiente:

Resultado Reduce	
Clave	Valor
Barcelona	23
Valencia	29
Alicante	32
Madrid	28
Sevilla	38

Tabla 6 - Resultado de función Reduce

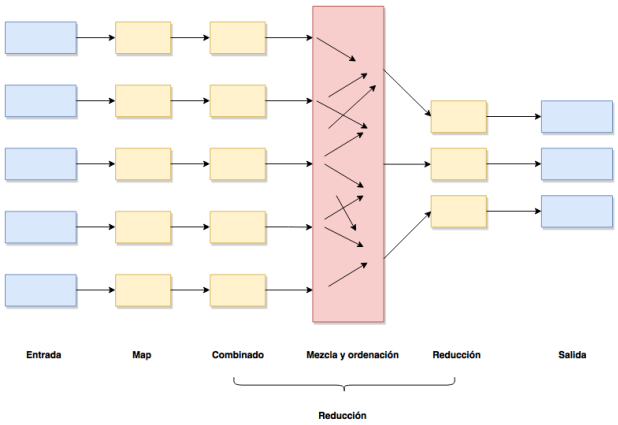


Imagen 3 - Esquema de MapReduce

De esta forma, se consigue un procesamiento más eficiente, ya que el cálculo de la temperatura máxima en cada uno de los ficheros se realiza en paralelo en los diferentes nodos del clúster, mezclando luego los resultados de dichos cálculos para obtener el resultado final.

3.2.3 Hadoop Yarn

Apache Hadoop Yarn[12] es una tecnología de gestión de clúster, también conocida como la versión 2.0 de MapReduce. Lo que Apache definía inicialmente como un gestor de recursos rediseñado, es hoy un sistema operativo distribuido para aplicaciones de Big Data. (13)

En 2012, Yarn se convirtió en un sub-proyecto del proyecto original de Hadoop. Es una reescritura del código que separa las capacidades de gestión y programación de recursos de las capacidades de proceso de datos, lo que hace que Hadoop sea capaz de soportar una mayor variedad de tipos de procesamiento y una mayor gama de aplicaciones. Por ejemplo, los clústeres de Hadoop ahora son capaces de ejecutar aplicaciones para consulta de datos o streaming interactivos al mismo tiempo que realizan trabajos de MapReduce.

Yarn combina un gestor de recursos central que gestiona la forma en que las aplicaciones utilizan los recursos del sistema, con agentes gestores en los nodos que gestionan las operaciones de proceso en los nodos individuales del clúster. La separación de HDFS del MapReduce con Yarn, hace a Hadoop más operativo para aplicaciones que no pueden esperar a la finalización de los trabajos por lotes en cada nodo.

3.3. Apache Spark 1.5.2

"Apache Spark es un motor rápido y de uso general para el procesamiento de datos a gran escala" (14). Es un framework de código abierto para computación en clúster. Desarrollado para el rendimiento, Spark puede resultar hasta 100x veces más rápido que Hadoop para el proceso

[12] YARN: Yet Another Resource Negotiator. Otro negociador (gestor) de recursos más

de grandes cantidades de información. Proporciona API´s de alto nivel con más de 100 operadores para Java, Python, Scala y R, además de un motor optimizado.

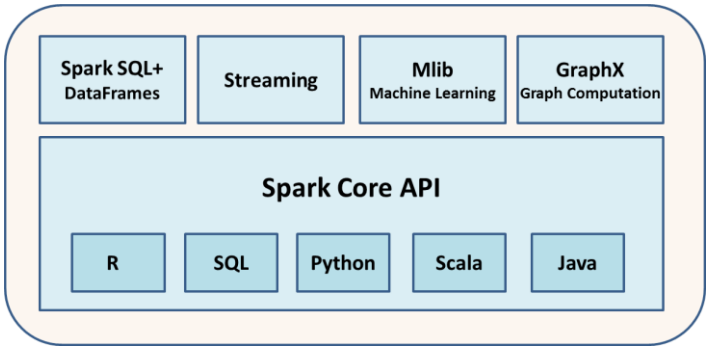


Imagen 4 - Ecosistema de Spark

Al contrario de lo que se cree comúnmente, Spark no es una versión modificada de Hadoop y no depende de él, ya que posee su propio sistema de gestión del clúster. Dado que Spark tiene su propio sistema de gestión de clúster de cálculo, se utiliza el HDFS de Hadoop únicamente para el almacenamiento.

El Core de Spark es el motor de funcionamiento general sobre el que se construyen todas las demás funcionalidades. Proporciona capacidades de computación *in-memory*[13] para obtener velocidad, y un modelo de ejecución para soportar una amplia variedad de aplicaciones, además de API´s para Java, Python y Scala. También soporta un amplio conjunto de herramientas de alto nivel entre las que se incluyen: (14)

- **Spark SQL**: Para el procesamiento de datos estructurados y SQL[14]. Proporciona una sólida integración con el resto del sistema.
- **MLib**: Es una biblioteca de aprendizaje automático escalable que ofrece algoritmos de alta calidad y una gran velocidad.
- **Graphx**: Es un motor de cálculo para el procesamiento gráfico.
- **Spark Streaming**: Para procesos de streaming en directo y tiempo real.

Spark está diseñado para cubrir una amplia gama de formas de trabajo tales como aplicaciones por lotes, algoritmos iterativos, consultas interactivas y streaming. Apache Spark tiene las siguientes características:

- **Velocidad:** Spark ayuda a la ejecución de una aplicación en un clúster Hadoop hasta 100 veces más rápido si se hace sobre la memoria RAM y hasta 10 veces más rápido si se realiza sobre el disco duro.
- **Múltiples lenguajes:** Spark ofrece API´s integrados para Java, Python, Scala y R, con más de 100 comandos de alto nivel.

[13] In-Memory: En memoria RAM
[14] SQL: Structured Query Language, lenguaje estándar de acceso y consulta de bases de datos estructuradas.

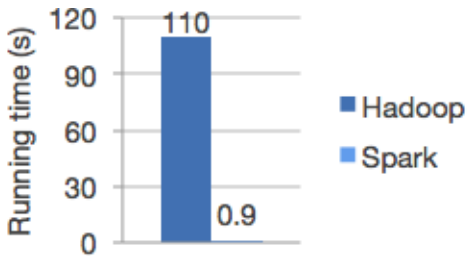


Imagen 5 - Comparativa velocidad Hadoop-Spark

Existen tres formas de realizar un despliegue con Apache Spark: (15)

- **Independiente**: En esta forma de despliegue, Spark se sitúa sobre el sistema HDFS (visto anteriormente) y el espacio se asigna de forma explícita. Aquí Spark y el sistema MapReduce trabajarán de forma conjunta para realizar todas las tareas del clúster.
- **Hadoop+Yarn**: Spark se ejecuta sobre Yarn sin necesidad de instalación previa. Ayuda a integrar Spark en el ecosistema de Hadoop y permite que otros componentes se ejecuten sobre él.
- **MapReduce**: Se utiliza para lanzar los trabajos aparte de la instalación independiente. El usuario puede trabajar directamente sobre el *Shell*[15] de Spark.

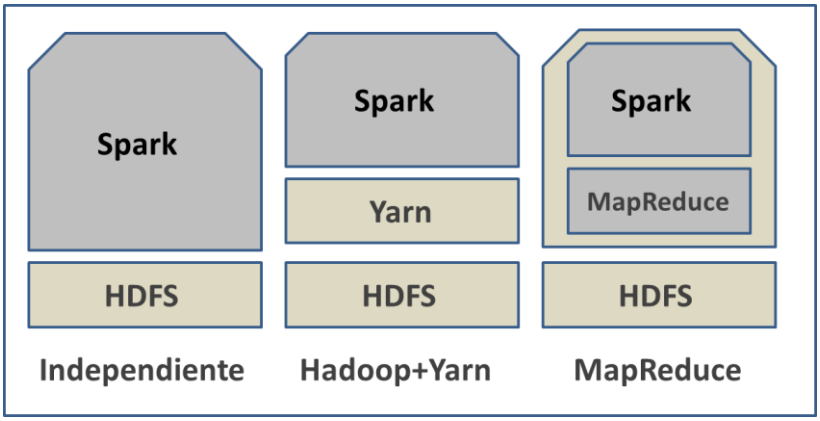


Imagen 6 - Diferentes modos de despliegue de Spark

[15] Shell: Interprete de comandos que proporciona un interface de usuario para acceder a los servicios de un sistema.

4. Instalación de sistema Big Data

A lo largo de este proyecto voy a realizar la instalación y configuración de un clúster de Big Data utilizando para ello el software descrito anteriormente (Apache Hadoop y Apache Spark), en uno de los laboratorios de la Escuela Técnica Superior de Ingeniería Informática de la Universidad Politécnica de Valencia.

4.1. Descripción del entorno

Una de las ventajas del software Apache Hadoop es que no se necesitan unos requerimientos de hardware elevados para realizar la instalación, lo que quiere decir que se puede instalar en sistemas informáticos de *bajo-coste*, es decir, PC´s normales de sobremesa sin necesidad de que sean servidores específicos de alto rendimiento.

La Escuela dispone de un laboratorio, el Anita Borg (en el edificio 1G), dotado con 32 ordenadores con las siguientes características:

- **Procesador**: Procesadores de última generación dotados con 4 núcleos (cores) cada uno de ellos. (La marca y modelo no son relevantes).
- **RAM**: Están dotados con 8Gb de memoria RAM.
- **Disco duro**: Aunque disponen de un disco de mayor capacidad, está particionado[16] para los sistemas operativos Windows y Linux. La capacidad disponible para el sistema Linux (que es el que vamos a utilizar) es de 180 Gb cada equipo, aunque el espacio libre es de aproximadamente 140 Gb en cada uno de ellos.
- **Sistema Operativo**: Aunque todos los equipos disponen de dos particiones (una para Windows y otra para Linux), nosotros vamos a trabajar exclusivamente sobre la partición de Linux donde tenemos instalado la versión 1.7 de Linux Mint "Qiana"

Imagen 7 - Características de los equipos

[16] Partición: Cada una de las divisiones existentes en una única unidad física de almacenamiento.

En cuanto a requerimientos adicionales a tener en cuenta, hemos de asegurarnos de que todas las máquinas tengan instalados los siguientes componentes "mínimos" para poder realizar la instalación correcta:

- Java en una versión superior a la 1.6.0
- SSH[17] para poder acceder a todos los equipos desde el NameNode

Disponemos de una LAN[18] que conecta todos los equipos a través de un Switch y que nos proporciona conexión con la WAN[19] de la Universidad y conectividad con Internet. Las características técnicas de la LAN no son relevantes para este proyecto por lo que no haré hincapié en ellas.

Los equipos están distribuidos en dos bloques separados tal y como se refleja en la tabla 4. Se utilizará el equipo con dirección IP[20] 158.42.215.3 como equipo Namenode, en él se ejecutaran tanto el servidor de Hadoop como el de Spark. En el equipo con IP 158.42.215.12 se ejecutará el Resource Manager (Yarn) para la gestión de los recursos del sistema.

El resto de equipos serán configurados como trabajadores (workers, slaves, etc.) de nuestro clúster y serán los encargados de almacenar la información (a través de HDFS y Hadoop) y de ejecutar en sus unidades de memoria y con los núcleos (cores) disponibles los trabajos lanzados por Spark.

158.42.215.3 Namenode	**158.42.215.12 Resourcenode**	158.42.215.14	158.42.215.28	158.42.215.29
158.42.215.197	158.42.215.30	158.42.215.39	158.42.215.47	158.42.215.48
158.42.215.221	158.42.215.49	158.42.215.56	158.42.215.57	158.42.215.58
158.42.215.228	158.42.215.60	158.42.215.59	158.42.215.61	158.42.215.62

158.42.215.63	158.42.215.64	158.42.215.65	158.42.215.66
158.42.214.88	158.42.214.42	158.42.215.132	158.42.214.91
158.42.214.24	158.42.215.99	158.42.215.54	158.42.215.66

Tabla 7 - Relación de direcciones IP

Existen tres formas diferentes de instalar y configurar nuestro sistema Hadoop en función de cuáles son nuestras necesidades: (9)

- **Single Node**: Instalación en un único nodo local, solo tiene sentido como ejemplo y para la realización de pruebas.
- **Pseudo-distribuido**: sobre una única máquina pero con varios nodos que se ejecutan sobre la misma máquina JVM.[21]
- **Multi-node**: Totalmente distribuido, cada máquina alberga un único nodo. Esta es la forma habitual para instalaciones de Big-Data y es la que vamos a utilizar.

[17] Ssh; Protocolo seguro para acceso a máquinas remotas a través de una red.
[18] LAN: Local Área Network, Red de Área Local.
[19] WAN: Wide Area Network, Red de Area Amplia.
[20] IP: Numero que identifica unívocamente a cada dispositivo dentro de una red.
[21] JVM: Java Virtual Machine

4.2. Preparar los Equipos

En primer lugar, es indispensable que en todos los equipos exista un mismo usuario que sea el que vamos a utilizar para trabajar con Hadoop. En nuestro caso, ese usuario es ***usulocal***, que está configurado en todos los equipos del aula y ha sido creado por los administradores de sistemas de la ETSINF.

El primer paso que debemos realizar es configurar el acceso *ssh* en todas las máquinas de manera que nuestros equipos Namenode y ResourceManager sean capaces de conectar con todas las demás sin necesidad de proporcionar la password en cada conexión, y de esta forma se puedan arrancar los servicios de Hadoop de forma automática. Para ello debemos realizar los siguientes pasos desde nuestro equipo Namenode:

1. Crear el directorio .ssh. Para ello, nos situamos en el directorio *home* del usuario usulocal y, si no existe, creamos el directorio .ssh y nos situamos en el

```
usulocal@lxeip6029 ~ $ mkdir -p .ssh
usulocal@lxeip6029 ~ $ cd .ssh
usulocal@lxeip6029 ~/.ssh $
```

2. Crear la clave de nuestro equipo Namenode con la siguiente instrucción:

```
usulocal@lxeip6029 ~/.ssh $
usulocal@lxeip6029 ~/.ssh $ ssh-keygen -t dsa -P '' -f id_dsa
Generating public/private dsa key pair.
Your identification has been saved in id_dsa.
Your public key has been saved in id_dsa.pub.
The key fingerprint is:
d6:3a:21:42:b2:22:49:37:be:f9:88:82:5d:97:20:1b usulocal@lxeip6029
The key's randomart image is:
+--[ DSA 1024]----+
|                 |
|                 |
| ..o.            |
|..E+o    .       |
|+ .=....S .      |
|... +.oo o       |
|.. + .  o        |
|o o o    .       |
|.. . .           |
+-----------------+
usulocal@lxeip6029 ~/.ssh $
```

lo que nos genera un fichero *id_dsa.pub* con nuestra clave pública. Esto debemos realizarlo también en el equipo Resource Manager.

3. A continuación, y para cada uno de los worker nodes, debemos añadir el fichero generado *id_dsa.pub* a su fichero *authorized_keys* con la siguiente instrucción (donde hay que sustituir la IP 158.42.215.14 por cada una de las direcciones IP de los workers):

```
usulocal@lxeip6029 ~/.ssh $ cat id_dsa.pub | ssh usulocal@158.42.215.14 '' cat -
>>.ssh/authorized_keys
```

   Esta instrucción debemos ejecutarla tanto desde el equipo Namenode como desde el equipo Resource manager.

4. Necesitamos, también en todos los worker nodes, subsanar los permisos de acceso al fichero *authorized_keys*. Para ello, accedemos a cada uno de los workers y ejecutamos las siguientes instrucciones:

```
usulocal@lxeip6029 ~/.ssh $ cd ~/.ssh
usulocal@lxeip6029 ~/.ssh $ chmod go-rwx .
usulocal@lxeip6029 ~/.ssh $ chmod go-rw authorized_keys
```

Los pasos 3 y 4 deben ser realizados para todos y cada uno de los worker nodes que tengamos instalados en el Sistema. A partir de aquí, podemos comprobar si podemos acceder a los workers sin necesidad de utilizar la password de acceso con la siguiente instrucción:

```
usulocal@lxeip6029 ~/.ssh $ ssh usulocal@158.42.215.14
Linux Mint 17 Qiana lxeip6002 ssh-pty
El sistema Linux Mint 17 esta en fase de proves. Si detectes un funcionament erroni o una mancança important, pots avisar-nos
enviant un missatge a tecnicos-inf@upv.es. Gracies!
mkdir: cannot create directory '/home/usulocal/Escritorio/pig': File exists
receiving incremental file list

sent 27 bytes  received 66 bytes  186.00 bytes/sec
total size is 363  speedup is 3.90
Dockerfile
docker_build.sh
docker_run.sh
receiving incremental file list

sent 27 bytes  received 67 bytes  188.00 bytes/sec
total size is 551  speedup is 5.86
Vagrantfile
bootstrap.sh
-sh: 16: [: -gt: unexpected operator
$ bash
usulocal@lxeip6002 ~/Escritorio/pig $
```

Si lo hemos realizado todo correctamente, el sistema conectara con el worker sin solicitarnos la password de conexión del usuario *usulocal*.

4.3. Preparar Hadoop y Spark

Una vez preparados todos los equipos para conectar por *ssh* sin necesidad de introducir la password, podemos preparar la instalación tanto de Hadoop como de Spark. Para ello, debemos realizar los siguientes pasos en el equipo Namenode y siempre trabajando con el usuario *usulocal*:

1. Descargar las versiones binarias más recientes y estables tanto de Hadoop como de Spark.
   a. Nosotros hemos descargado la versión 2.6.2 de Hadoop, lo que se puede realizar desde la siguiente url:
   http://www.apache.org/dyn/closer.cgi/hadoop/common/hadoop-2.6.2/hadoop-2.6.2.tar.gz

   b. Para Spark, hemos descargado la versión 1.5.2, lo que se puede realizar desde la url siguiente:
   http://www.apache.org/dyn/closer.lua/spark/spark-1.5.2/spark-1.5.2-bin-hadoop2.6.tgz

2. Creamos un directorio donde ubicar todo nuestro sistema. En nuestro caso, creamos la carpeta *cluster*, además, dentro de ella creamos otra carpeta *opt* donde desplegaremos todo el sistema:

```
usulocal@lxeip6029 ~/cluster $ mkdir cluster
usulocal@lxeip6029 ~/cluster $ cd ~/cluster
usulocal@lxeip6029 ~/cluster $ mkdir opt
```

3. Desplegamos Hadoop dentro del directorio que acabamos de crear:

```
usulocal@lxeip6029 ~ $ cd ~/cluster/opt
usulocal@lxeip6029 ~/cluster/opt $ tar zxvf /tmp/hadoop2.6.2.tar.gz
```

   (donde hemos de sustituir el 2.6.2 por la versión que corresponda a nuestra descarga).

4. Ahora creamos un enlace simbólico para trabajar con la carpeta *cluster/opt/hadoop* que apunte a la de la versión que hemos descargado

```
usulocal@lxeip6029 ~ $ cd ~/cluster/opt
usulocal@lxeip6029 ~/cluster/opt $ ln -s hadoop-2.6.2 hadoop
```

5. De la misma forma que en el punto 3, desplegamos Spark

```
usulocal@lxeip6029 ~ $ cd ~/cluster/opt
usulocal@lxeip6029 ~/cluster/opt $ tar zxvf /tmp/spark-1.5.2-bin-hadoop2.6.tgz
```

   (sustituyendo también las versiones 1.5.2 y 2.6 por las versiones correspondientes).

6. Tal y como hemos hecho en el punto 4 para Hadoop, ahora hacemos lo mismo con el enlace simbólico de Spark para trabajar con la carpeta *cluster/opt/spark* y que apunte a la de la versión que hemos descargado

```
usulocal@lxeip6029 ~ $ cd ~/cluster/opt
usulocal@lxeip6029 ~/cluster/opt $ ln -s spark-1.5.2-bin-hadoop2.6 spark
```

4.4. Configurar Hadoop

El sistema Hadoop, necesita una serie de directorios de trabajo, tanto en el Namenode como en el Resource Manager y en los worker nodes. Estos directorios se deben crear dentro del directorio *cluster* que hemos creado anteriormente con las siguientes instrucciones:

```
#!/bin/bash

echo "creating hadoop directories"

mkdir -p "${HOME}/cluster/var/hadoop"
mkdir -p "${HOME}/cluster/var/hadoop/hadoop-datanode"
mkdir -p "${HOME}/cluster/var/hadoop/hadoop-namenode"
mkdir -p "${HOME}/cluster/var/hadoop/mr-history"
mkdir -p "${HOME}/cluster/var/hadoop/mr-history/done"
mkdir -p "${HOME}/cluster/var/hadoop/mr-history/tmp"
```

Imagen 8 - Creación directorios

Necesitamos también decirle al Sistema cuales son los nodos que tiene disponible para trabajar. Esto lo hacemos creando un fichero llamado *slaves* y ubicado en las carpetas *~/cluster/opt/hadoop/etc/hadoop/slaves* y en *~/cluster/opt/spark/conf/slaves*. Este fichero contendrá el nombre (o dirección IP) de cada uno de los worker nodes por línea.

4.4.1. core-site-xml

Ahora ya podemos proceder a editar los ficheros de configuración del sistema Hadoop. En primer lugar, debemos configurar el fichero *~/cluster/opt/hadoop/etc/hadoop/core-site.xml*. Su principal función es la de definir el tamaño del buffer de lectura/escritura y el nombre y ubicación del Namenode que vamos a utilizar. (16)

Parámetro	Valor	Notas
fs.defaultFS	NameNode URI	hdfs://host:port/
io.file.buffer.size	131072	Tamaño del buffer de Entrada/Salida utilizado en SequenceFiles.

Tabla 8 - Parámetros de core-site.xml

En nuestro caso, el fichero *core-site.xml* queda de la siguiente manera:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>


<configuration>
        <property>
                <name>fs.defaultFS</name>
                <value>hdfs://158.42.215.3:9000</value>
        </property>
        <property>
                <name>io.file.buffer.size</name>
                <value>131072</value>
        </property>
</configuration>
```

Imagen 9 - Fichero core-site.xml

4.4.2. hdfs-site.xml

El siguiente fichero a configurar es el *~/cluster/opt/hadoop/etc/hadoop/hdfs-site.xml*. En este fichero aparece la configuración para la gestión de nuestro sistema de archivos HDFS, donde podemos configurar dos tipos de parámetros, unos para el Namenode y otros para los data nodes (workers) según el siguiente esquema (16):

	Parámetro	Valor	Notas
NameNode	dfs.replication	Número de replicas	Número de replicas de cada bloque que realizará el sistema HDFS
	dfs.namenode.name.dir	NameNode URI	Lista de directorios separados por comas donde se replicaran los nombres de las tablas para redundancia.
	dfs.namenode.hosts /dfs.namenode.hosts.exclude	Lista de nodos permitidos/excluidos	Si es necesario, se pueden utilizar este fichero para realizar listas de nodos permitidos o excluidos.
	dfs.blocksize	268435456	Tamaño del bloque del HDFS de 256MB para grandes sistemas de ficheros.
	dfs.namenode.handler.count	100	Máximo numero de hilos del servidor Namenode para gestionar RPC de un gran número de nodos.
Data Node	dfs.namenode.handler.count	Lista de rutas, separadas por comas, en el sistema de almacenamiento local donde almacenar los bloques.	Si hay una lista de directorios separadas por coma, los datos serán almacenados en todos los directorios, normalmente en dispositivos diferentes.

Tabla 9 - Parámetros de hdfs-site.xml

No es necesario utilizar todos los parámetros del fichero de configuración, solo aquellos que deseemos modificar para nuestra instalación concreta. De todos los parámetros, el más interesante son el de *dfs.replication*, que define el número de réplicas de cada bloque que se realizarán sobre el sistema y el *dfs.blocksize* que determina el tamaño de bloque de almacenamiento. Nuestro fichero *hdfs-site.xml* quedaría de la siguiente manera en nuestra instalación:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
        <property>
                <name>dfs.replication</name>
                <value>3</value>
        </property >

        <property>
                <name>dfs.namenode.name.dir</name>
                <value>file:///home/usulocal/cluster/var/hadoop/hadoop-namenode</value>
        </property>


    <property>
        <name>dfs.webhdfs.enabled</name>
        <value>true</value>
    </property>
    <property>
        <name>dfs.permissions.enabled</name>
        <value>false</value>
    </property>
    <property>
        <name>dfs.namenode.handler.count</name>
        <value>100</value>
    </property>

        <property>
                <name>dfs.datanode.data.dir</name>
        <value>file:///home/usulocal/cluster/var/hadoop/hadoop-datanode</value>
        </property>
</configuration>
```

Imagen 10 - Fichero hdfs-site.xml

4.4.3. mapred-site.xml

A continuación tenemos el fichero de configuración del sistema de MapReduce, *~/cluster/opt/hadoop/etc/hadoop/mapred-site.xml*, que nos ayuda a configurar el comportamiento y el uso de recursos en los procesos Map Reduce y la gestión de los históricos de dichos trabajos (16).

	Parámetro	Valor	Notas
Aplicaciones MapReduce	mapreduce.framework.name	yarn	Establece el framework de ejecución en Hadoop Yarn
	mapreduce.map.memory.mb	1536	Limite superior de recursos de memoria para mapeo
	mapreduce.map.java.opts	-Xmx1024M	Tamaño maximo de pila para trabajos hijos de JVM en mapeo.
	mapreduce.reduce.memory.mb	3072	Limite máximo de recursos de memoria para reduce.
	mapreduce.reduce.java.opts	-Xmx2560M	Tamaño máximo de pila paratrabajos hijos de JVMpara Reduce.
	mapreduce.task.io.sort.mb	512	Limite superior de memoria para ordenación de datos.
	mapreduce.task.io.sort.factor	100	Número de cadenas fusionadas simultaneamente para ordenación de archivos
	mapreduce.reduce.shuffle.parallelcopies	50	Mayor número de copias paralelas ejecutadas por tareas Reduce para números elevados de mapeos
Histórico de MapReduce	mapreduce.jobhistory.address	MapReduce JobHistory Server host:port	El puerto por defecto es el 10020
	mapreduce.jobhistory.webapp.address	MapReduce JobHistory Server Web UI host:port	El puerto por defecto es 19888
	mapreduce.jobhistory.intermediate-done-dir	/mr-history/tmp	Directorio donde se escriben los archivos de histórico de trabajos MapReduce
	mapreduce.jobhistory.done-dir	/mr-history/done	Directorio donde son gestionados los archivos del servidor histórico

Tabla 10 - Parámetros de mapred-site.xml

En nuestro caso, el fichero mapred-site.xml queda configurado de la siguiente manera:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
        <property>
                <name>mapreduce.framework.name</name>
                <value>yarn</value>
        </property>
        <property>
        <name>mapreduce.map.memory.mb</name>
        <value>1536</value>
        </property>
    <property>
        <name>mapreduce.map.java.opts</name>
        <value>-Xmx1024M</value>
        </property>
        <property>
        <name>mapreduce.reduce.memory.mb</name>
        <value>3072</value>
        </property>
        <property>
        <name>mapreduce.reduce.java.opts</name>
        <value>-Xmx2560M</value>
        </property>
        <property>
        <name>mapreduce.task.io.sort.mb</name>
        <value>512</value>
        </property>
        <property>
        <name>mapreduce.task.io.sort.factor</name>
        <value>100</value>
        </property>
        <property>
        <name>mapreduce.reduce.shuffle.parallelcopies</name>
        <value>50</value>
        </property>

    <property>
        <name>mapreduce.jobhistory.address</name>
        <value>158.42.215.12:10020</value>
    </property>
    <property>
        <name>mapreduce.jobhistory.webapp.address</name>
        <value>158.42.215.12:19888</value>
    </property>
    <property>
        <name>mapreduce.jobhistory.intermediate-done-dir</name>
        <value>/home/usulocal/cluster/var/hadoop/mr-history/tmp</value>
    </property>
    <property>
        <name>mapreduce.jobhistory.done-dir</name>
        <value>/home/usulocal/cluster/var/hadoop/mr-history/done</value>
    </property>

</configuration>
```

Imagen 11 - Fichero mapred-site.xml

4.4.4. yarn-site.xml

Por su parte, para la configuración tanto del gestor de recursos Yarn como de los ficheros de *log*[22] generados, disponemos de las siguientes opciones para establecer en el fichero *~/cluster/opt/hadoop/etc/hadoop/yarn-site.xml* (16).

[22] Log: Registro de eventos en un lapso de tiempo

	Parámetro	Valor	Notas
Resource and Node Manager	yarn.acl.enable	true /false	Habilita el ACL (Access Control List). Valor por defecto *false*
	yarn.admin.acl	Admin ACL	ACL para usuarios/grupos separados por comas. El valor por defecto es * que significa que todos administran. Un espacio significa que nadie tiene acceso.
	yarn.log-aggregation-enable	true /false	Configuración para habilitar/deshabilitar la agregación de logs.
ResourceManager	yarn.resourcemanager.address	ResourceManager host:port para el envio de trabajos.	host:port Si aparece, sobreescribe el hostname establecido en yarn.resourcemanager.hostname.
	yarn.resourcemanager.scheduler.address	ResourceManager host:port para asignación de recursos por parte del Scheduler.	host:port Si aparece, sobreescribe el hostname establecido en yarn.resourcemanager.hostname.
	yarn.resourcemanager.resource-tracker.address	ResourceManager host:port para los Node Managers	host:port Si aparece, sobreescribe el hostname establecido en yarn.resourcemanager.hostname.
	yarn.resourcemanager.admin.address	ResourceManager host:port para los comandos de administración	host:port Si aparece, sobreescribe el hostname establecido en yarn.resourcemanager.hostname.
	yarn.resourcemanager.webapp.address	ResourceManager web-ui host:port.	host:port Si aparece, sobreescribe el hostname establecido en yarn.resourcemanager.hostname.
	yarn.resourcemanager.hostname	ResourceManager host.	Nombre de host que se puede utilizar en lugar de los yarn.resourcemanager*address. Utiliza los puertos estandar.
	yarn.resourcemanager.scheduler.class	ResourceManager Scheduler class	Puede tomar los valores CapacityScheduler (recomendado), FairScheduler (tambien recomendado), or FifoScheduler
	yarn.scheduler.minimum-allocation-mb	Minima cantidad de memoria a asignar a cada solicitud del RM	En MBs.
	yarn.scheduler.maximum-allocation-mb	Máxima cantidad de memoria a asignar a cada solicitud del RM	En MBs.
	yarn.resourcemanager.nodes.include-path /yarn.resourcemanager.nodes.exclude-path	Lista de NodeManagers permitidos/excluidos	Si es necesario, se puede utilizar esta lista para determinar los NodeManagers disponibles
NodeManager	yarn.nodemanager.resource.memory-mb	Memoria fisica disponible, en MB, para un Node Manager.	Define los recursos totales disponibles en el Node Manager para ejecutar los contenedores.
	yarn.nodemanager.vmem-pmem-ratio	Ratio máximo en que la memoria virtual puede exceder a la memoria fisica	La memoria viirtual utilizada por cada tarea puede exceder a la memoria fisica disponible en porporción al ratio establecido.
	yarn.nodemanager.local-dirs	Lista de rutas separadas por comas	Lista de rutas del sistema de almacenamiento local donde se escribe la información y datos intermedios
	yarn.nodemanager.log-dirs	Lista de rutas separadas por comas	Lista de rutas del sistema de almacenamiento local donde se escribe los logs
	yarn.nodemanager.log.retain-seconds	10800	Tiempo predeterminado (en segundos) para conservar los logs del NodeManager (si está desactivada la agregación)
	yarn.nodemanager.remote-app-log-dir	/logs	Directorio de HDFS donde se mueven los ficheros de log cuando se finaliza una aplicación. Necesita permisos
	yarn.nodemanager.remote-app-log-dir-suffix	logs	Sufijo anexado al directorio de registro remoto. Los logs se agregan a ${yarn.nodemanager.remote-app-log-dir}/${user}/${thisParam} Sólo si el registro de agregación está habilitado
	yarn.nodemanager.aux-services	mapreduce_shuffle	Servicio shuffle que necesita ser ajustado para tareas deMapReduce

Tabla 11 - Parámetros de yarn-site.xml

```
<?xml version="1.0"?>

<configuration>

        <property>
                <name>yarn.acl.enable</name>
                <value>false</value>
        </property>
        <property>
                <name>yarn.admin.acl</name>
                <value>*</value>
        </property>
        <property>
                <name>yarn.log-aggregation-enable</name>
                <value>false</value>
        </property>


        <property>
                <name>yarn.resourcemanager.hostname</name>
                <value>158.42.215.12</value>
        </property>
        <property>
                <name>yarn.resourcemanager.scheduler.class</name>
                <value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler</value>
        </property>
        <property>
                <name>yarn.scheduler.minimum-allocation-mb</name>
                <value>1024</value>
        </property>
```

```
        <property>
                <name>yarn.scheduler.maximum-allocation-mb</name>
                <value>8192</value>
        </property>

        <property>
                <name>yarn.nodemanager.resource.memory-mb</name>
                <value>8192</value>
        </property>
        <property>
                <name>yarn.nodemanager.vmem-pmem-ratio</name>
                <value>2.1</value>
        </property>
        <property>
                <name>yarn.nodemanager.log.retain-seconds</name>
                <value>10800</value>
        </property>
        <property>
                <name>yarn.nodemanager.remote-app-log-dir</name>
                <value>/tmp/logs</value>
        </property>
        <property>
                <name>yarn.nodemanager.remote-app-log-dir-suffix</name>
                <value>logs</value>
        </property>
```

```
        <property>
                <name>yarn.nodemanager.aux-services</name>
                <value>mapreduce_shuffle</value>
        </property>
        <property>
                <name>yarn.log-aggregation.retain-seconds</name>
                <value>-1</value>
        </property>
        <property>
                <name>yarn.log-aggregation.retain-check-interval-seconds</name>
                <value>-1</value>
        </property>

    <property>
        <name>yarn.web-proxy.address</name>
        <value>158.42.215.12:8089</value>
    </property>
    <property>
        <name>yarn.timeline-service.hostname</name>
        <value>158.42.215.12</value>
    </property>


</configuration>
```

Imagen 12 - Fichero yarn-site.xml

4.4.5. Configuración Spark

El sistema Spark está diseñado para ser gestionado desde programas (Java, Python, etc.) y utilizando sus directivas de configuración. Sin embargo, el sistema dispone de un archivo *spark-defaults.conf* donde podemos establecer los parámetros de ejecución por defecto de nuestro sistema. En nuestro caso hemos establecido únicamente la dirección del sistema maestro de spark (en nuestro Namenode), hemos habilitado la generación de logs, en el fichero siguiente: (17)

```
# Default system properties included when running spark-submit.
# This is useful for setting default environmental settings.

# Example:
spark.master                     spark://158.42.215.3:7077
spark.eventLog.enabled           true
#spark.eventLog.dir               hdfs://158.42.215.3:8021/directory
spark.serializer                 org.apache.spark.serializer.KryoSerializer
spark.driver.memory              5g
spark.executor.extraJavaOptions  -XX:+PrintGCDetails -Dkey=value -Dnumbers="one two three"
```

Imagen 13 - Fichero spark-defaults.conf

4.5. Configurar variables de entorno

Además, tenemos que configurar las variables de entorno, de manera que el sistema sea capaz de encontrar nuestra instalación de Hadoop y de Spark y pueda encontrar cada uno de sus directorios cuando lo necesite. Debemos crear el fichero *~/.cluster_rc* con las siguientes líneas:

```
export HADOOP_PREFIX="${HOME}/cluster/opt/hadoop"
export HADOOP_HOME="${HADOOP_PREFIX}"
export HADOOP_CONF_DIR="${HADOOP_PREFIX}/etc/hadoop"

PATH="${HADOOP_HOME}/bin:${HADOOP_HOME}/sbin:${PATH}"
export PATH

export SPARK_PREFIX="${HOME}/cluster/opt/spark"
export SPARK_HOME="${SPARK_PREFIX}"
export SPARK_CONF_DIR="${SPARK_PREFIX}/conf"

PATH="${SPARK_HOME}/bin:${SPARK_HOME}/sbin:${PATH}"
export PATH

export JAVA_HOME="/opt/jdk1.8.0_25"
```

Imagen 14 - Variables de entorno en cluster_rc

Por supuesto, los directorios que ponemos en las variables de entorno van en función de nuestra instalación concreta, tanto para Hadoop como para Spark y Java. Una vez creado este fichero, debemos añadir a los ficheros ~/.bashrc y ~/.profile la siguiente línea para que el entorno sea cargado de forma automática:

```
. ~/.cluster_rc
```

4.6. Despliegue en los workers y Resource Manager

Ya hemos finalizado la instalación de nuestro Namenode o servidor principal de Hadoop, pero ahora necesitamos realizar el despliegue en todo el resto de equipos (workers) y en el Resource Manager. Para ello nos bastaría copiar la configuración que hemos realizado en el Namenode

sobre cada uno de los equipos con los siguientes pasos sustituyendo <worker> por la dirección IP o nombre de cada uno de los equipos:

- Copiar toda la instalación que hemos realizado sin descargar cada vez los binarios de Hadoop y de Spark. Para ello podemos utilizar el comando de Linux:

```
# rsync –avH ~/cluster  usulocal@<worker>:~/cluster
```

- Copiar la configuración de las variables de entorno a cada uno de los workers. Lo podemos realizar utilizando estas instrucciones para copiar ficheros por nuestra conexión SSH:

```
# scp ~/.cluster_rc  usulocal@$<worker>:.cluster_rc
# scp ~/.bashrc      usulocal@$<worker>:.bashrc
# scp ~/.profile     usulocal@$<worker>:.profile
```

Como en nuestro caso estamos hablando de 31 equipos (1 Resource Manager + 30 workers), el trabajo puede ser realmente tedioso, pero si, además, en nuestra instalación tenemos un número elevado de equipos (digamos 200), el trabajo puede ser casi "eterno". Para solucionarlo, creamos un script[23] que realice todo el trabajo por nosotros.

```
#!/bin/bash

i=0
IPS[i++]=215.14
IPS[i++]=215.28
IPS[i++]=215.29
IPS[i++]=215.197
IPS[i++]=215.68

for ip in ${IPS[@]}
do
      workernode="158.42.${ip}"

      echo ${workernode}

      # Despliega la configuración del servidor en cada worker
      rsync ~/cluster  usulocal@${workernode}:~/cluster

      # Despliega la configuracion local de entorno a cada worker
      scp ~/.cluster_rc  usulocal@${workernode}:.cluster_rc
      scp ~/.bashrc      usulocal@${workernode}:.bashrc
      scp ~/.profile     usulocal@${workernode}:.profile

done
```

Imagen 15 - Script para realizar despliegue

En este fichero de script, hemos dejado solo las 5 primeras direcciones IP de nuestro sistema para que la imagen no fuera demasiado larga, pero deberíamos tener una línea del tipo *IPS[i++]=215.14* por cada una de las direcciones IP de nuestro sistema. El script lo que hace es un bucle donde, para cada una de las direcciones IP reflejadas, realiza las dos operaciones descritas anteriormente, la operación *rsync* para el copiado de los sistemas Hadoop y Spark y las operaciones *scp* para copiado de los ficheros de entorno.

[23] Script: archivo de órdenes, archivo de procesamiento por lotes.

5. Operación del sistema

Una vez realizados todos los pasos anteriores, y si no hemos tenido ningún problema, ya estamos en condiciones de poner en marcha nuestro sistema Big Data con Hadoop (HDFS) como sistema de almacenamiento, Yarn como nuestro gestor de recursos y Spark como nuestro sistema de cómputo de grandes trabajos. Para comprobar que todo ha sido instalado correctamente, podemos utilizar el comando *hadoop version* que nos debe dar la versión del sistema que acabamos de instalar:

Imagen 16 - Versión instalada Hadoop

5.1. Puesta en marcha

Antes de poder realizar la puesta en marcha de nuestro sistema, hemos de realizar el formateo del mismo. Esta operación debe ser realizada únicamente una vez. El sistema genera un *id* (identificador) que graba tanto en el servidor Namenode como en cada uno de los worker nodes. Este identificador tiene que ser el mismo para todos los equipos. Si reformateamos un sistema, se regenera el *id* en el Namenode pero no en los worker nodes (interpreta que pertenecen a otra instalación de HDFS), por lo que a partir de este momento los *id* son diferentes y el sistema deja de funcionar (la solución pasa por borrar los ficheros con los *id* en los worker nodes y reformatear, pero se pierde la información almacenada). La instrucción para formatear el sistema es la siguiente:

```
# ~/cluster/opt/hadoop/bin/hdfs namenode -format
```

La puesta en marcha es sencilla y necesita de tres pasos (en este mismo orden) que tendrán que ser realizados cada vez que se ponga en marcha el sistema:

- En primer lugar, hemos de poner en marcha el sistema de Hadoop. Para ello, en nuestra máquina Namenode, ejecutamos el script que está en el directorio de ejecutables *~/cluster/opt/hadoop/sbin/start-dfs.sh*, que pone en marcha el cluster HDFS.

```
usulocal@lxeip6029 ~/cluster/opt/hadoop $ sbin/start-dfs.sh
Starting namenodes on [lxeip6029]
lxeip6029: Linux Mint 17 Qiana lxeip6029 ssh-pty
lxeip6029: starting namenode, logging to /home/usulocal/cluster/opt/hadoop-2.6.2
/logs/hadoop-usulocal-namenode-lxeip6029.out
158.42.215.221: Linux Mint 17 Qiana lxeip6031 ssh-pty
158.42.215.28: Linux Mint 17 Qiana lxeip6003 ssh-pty
158.42.215.65: Linux Mint 17 Qiana lxeip6019 ssh-pty
158.42.215.48: Linux Mint 17 Qiana lxeip6008 ssh-pty
158.42.215.132: Linux Mint 17 Qiana lxeip6023 ssh-pty
158.42.214.91: Linux Mint 17 Qiana lxeip6024 ssh-pty
158.42.215.49: Linux Mint 17 Qiana lxeip6009 ssh-pty
158.42.215.39: Linux Mint 17 Qiana lxeip6006 ssh-pty
158.42.215.57: Linux Mint 17 Qiana lxeip6011 ssh-pty
158.42.214.24: Linux Mint 17 Qiana lxeip6025 ssh-pty
158.42.215.62: Linux Mint 17 Qiana lxeip6016 ssh-pty
158.42.215.66: Linux Mint 17 Qiana lxeip6020 ssh-pty
```

Imagen 17 - Arranque sistema Hadoop

- Una vez arrancado el sistema Hadoop, en el equipo que hace de Resource Manager, debemos arrancar el sistema Yarn con la instrucción *~/cluster/opt/hadoop/sbin/start-yarn.sh*

```
usulocal@lxeip6001 ~/cluster/opt/hadoop $
usulocal@lxeip6001 ~/cluster/opt/hadoop $ sbin/start-yarn.sh
starting yarn daemons
starting resourcemanager, logging to /home/usulocal/cluster/opt/hadoop/logs/yarn
-usulocal-resourcemanager-lxeip6001.out
158.42.215.14: Linux Mint 17 Qiana lxeip6002 ssh-pty
158.42.215.197: Linux Mint 17 Qiana lxeip6030 ssh-pty
158.42.215.228: Linux Mint 17 Qiana lxeip6032 ssh-pty
158.42.215.68: Linux Mint 17 Qiana lxeip6028 ssh-pty
158.42.214.42: Linux Mint 17 Qiana lxeip6022 ssh-pty
158.42.215.64: Linux Mint 17 Qiana lxeip6018 ssh-pty
158.42.215.221: Linux Mint 17 Qiana lxeip6031 ssh-pty
158.42.215.62: Linux Mint 17 Qiana lxeip6016 ssh-pty
158.42.215.28: Linux Mint 17 Qiana lxeip6003 ssh-pty
158.42.215.29: Linux Mint 17 Qiana lxeip6004 ssh-pty
158.42.214.88: Linux Mint 17 Qiana lxeip6021 ssh-pty
158.42.215.47: Linux Mint 17 Qiana lxeip6007 ssh-pty
158.42.215.58: Linux Mint 17 Qiana lxeip6012 ssh-pty
158.42.215.99: Linux Mint 17 Qiana lxeip6026 ssh-pty
```

Imagen 18 - Arranque sistema Yarn

- Por último, tenemos que poner en marcha el sistema Spark, de nuevo desde el equipo Namenode desde el que hemos arrancado Hadoop, con la instrucción *~/cluster/opt/spark/sbin/start-all.sh*

5.2. Parada del sistema

La parada del sistema es igual de sencilla que la puesta en marcha y hemos de realizar las operaciones en orden inverso:

- En primer lugar, necesitamos parar el sistema Spark en el equipo Namenode con la instrucción *~/cluster/opt/spark/sbin/stop-all.sh*
- A continuación, hemos de para el Yarn. Esto lo realizamos en el equipo ResourceManager con la instrucción *~/cluster/opt/hadoop/sbin/stop-yarn.sh*
- Por último, hemos de parar el sistema Hadoop y HDFS en el Namenode utilizando *~/cluster/opt/hadoop/sbin/stop-dfs.sh*

5.3. Comandos HDFS

El sistema de almacenamiento HDFS de Hadoop dispone de un sistema de comandos similar al *shell*, que nos permite realizar las operaciones más frecuentes que podemos necesitar sobre archivos. El uso de estos comandos es con la línea de comandos: (18)

```
# hdfs [--config confdir] [COMMAND] [GENERIC_OPTIONS] [COMMAND_OPTIONS]
```

- **appendToFile** Anexo uno o varios ficheros del sistema local de archivos al fichero de destino. También lee la entrada del sistema estándar y lo añade al destino.

  ```
  # hdfs dfs -appendToFile <local> ... <destino>
  ```

- **cat** copia los directorios de entrada a la salida estándar stdout

  ```
  # hdfs dfs -cat URI [URI ...]
  ```

- **chgrp** cambia la asociación de ficheros a un grupo

  ```
  # hdfs dfs -chgrp [-R] GROUP URI [URI ...]
  ```

- **chmod** cambia los permisos de los ficheros. Con –R, se realiza de forma recursiva a través de la estructura de directorios.

  ```
  # hdfs dfs -chmod [-R] <MODE[,MODE]... | OCTALMODE> URI [URI ...]
  ```

- **chown** cambia el propietario de los ficheros. Con –R, se realiza de forma recursiva a través de la estructura de directorios.

  ```
  # hdfs dfs -chown [-R] [OWNER][:[GROUP]] URI [URI ]
  ```

- **copyFromLocal** Similar al comando put (comentado más adelante) solo que el fichero de origen debe ser local.

  ```
  # hdfs dfs -copyFromLocal <local> URI
  ```

- **copyToLocal** similar al comando get (comentado más adelante), solo que el fichero de destino debe ser local.

  ```
  # hdfs dfs -copyToLocal [-ignorecrc] [-crc] URI <localdestino>
  ```

- **count** cuenta el número de ficheros, directorios y bytes en el *path* especificado y que coinciden con el patrón establecido.

  ```
  # hdfs dfs -count [-q] [-h] <path>
  ```

- **cp** copia ficheros desde el origen al destino.

  ```
  # hdfs dfs -cp [-f] [-p | -p[topax]] URI [URI ...] <destino>
  ```

- **du** muestra los tamaños de ficheros y directorios en el directorio especificado.

  ```
  # hdfs dfs -du [-s] [-h] URI [URI ...]
  ```

- **expunge** realiza el vaciado de la papelera.

```
# hdfs dfs -expunge
```

- **get** copia ficheros al sistema de ficheros local

```
# hdfs dfs -get [-ignorecrc] [-crc] <src> <localdestino>
```

- **getfacl** muestra la Lista de Control de Accesos (ACL) de los ficheros y directorios.

```
# hdfs dfs -getfacl [-R] <path>
```

- **getfattr** muestra los nombres y valores de los atributos extendidos (si existen) de ficheros o directorios.

```
# hdfs dfs -getfattr [-R] -n name | -d [-e en] <path>
```

- **getmerge** toma los ficheros de un directorio de origen y los concatena sobre el fichero de destino.

```
# hdfs dfs -getmerge <src> <localdestino> [addnl]
```

- **ls** devuelve un listado con el estado de los ficheros o de los directorios.

```
# hdfs dfs -ls [-R] <argumentoss>
```

- **mkdir** crea un directorio.

```
# hdfs dfs -mkdir [-p] <paths>
```

- **moveFromLocal** similar al comando put, solo que aquí el fichero de origen *localsrc* es borrado al finalizar.

```
# hdfs dfs -moveFromLocal <localsrc> <destino>
```

- **mv** mueve ficheros del origen al destino.

```
# hdfs dfs -mv URI [URI ...] <destino>
```

- **put** copia uno o varios ficheros del origen local al directorio de destino. También, toma la entrada estándar y la escribe en un fichero.

```
# hdfs dfs -put <localsrc> ... <destino>
```

- **rm** borra los ficheros especificados como argumentos.

```
# hdfs dfs -rm [-f] [-r|-R] [-skipTrash] URI [URI ...]
```

- **setfacl** establece la lista de control de accesos (ACL) de un fichero o directorio.

```
# hdfs dfs -setfacl [-R] [-b|-k -m|-x <acl_spec> <path>]|[--
set <acl_spec> <path>]]
```

- **setrep** cambia el factor de replicación para un fichero o directorio.

```
# hdfs dfs -setrep [-R] [-w] <numReplicas> <path>
```

- **stat** devuelve la información estadística del path.

  ```
  # hdfs dfs -stat URI [URI ...]
  ```

- **test** realiza diversas pruebas sobre el fichero, con el argumento *–e* comprueba si el fichero existe, con *–z* comprueba si tiene longitud cero y con *–d* comprueba si es un directorio.

  ```
  # hdfs dfs -test -[ezd] URI
  ```

6. Funcionamiento HDFS

El sistema HDFS proporciona una interface web que nos puede ayudar para realizar las operaciones básicas de monitorización y la navegación por los archivos de nuestro cluster. Por defecto, viene configurado en el puerto 50070 de nuestro servidor Namenode y podemos acceder con cualquier navegador web (en nuestro caso, en *http://158.42.215.3:50070/*).

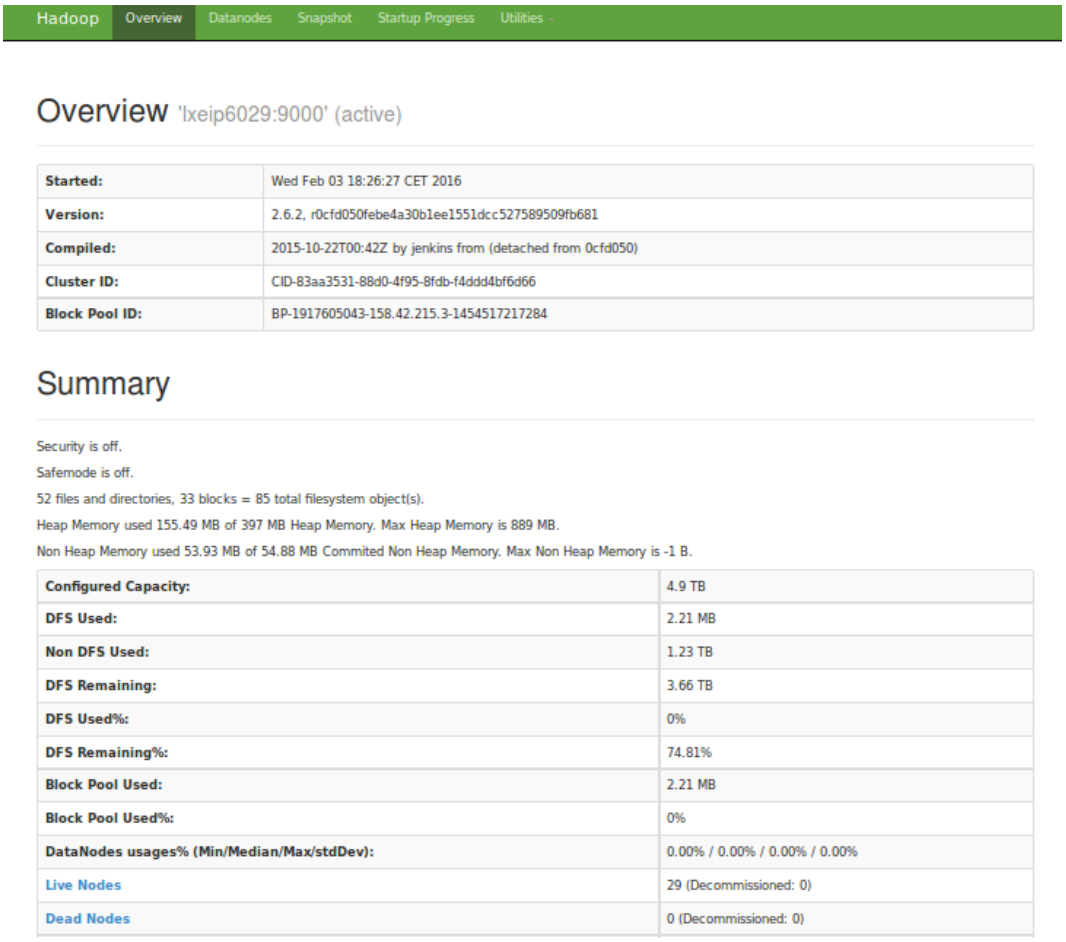
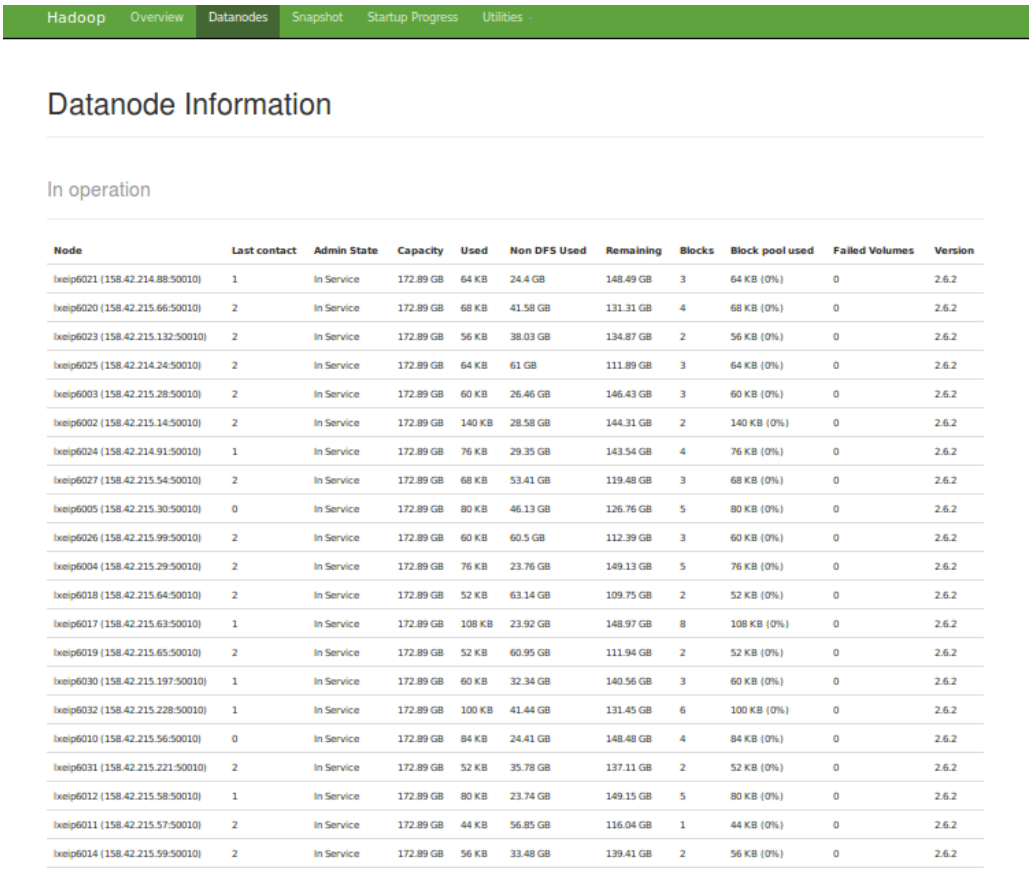


Imagen 19 - Información general cluster HDFS

La dirección y el puerto de la interface web puede ser modificado cambiando los valores del parámetro de configuración *dfs.http.address* en el archivo *hadoop-site.xml*. Además, cada worker dispone de su propia interface web en el puerto 50075(http://<worker>:50075/).

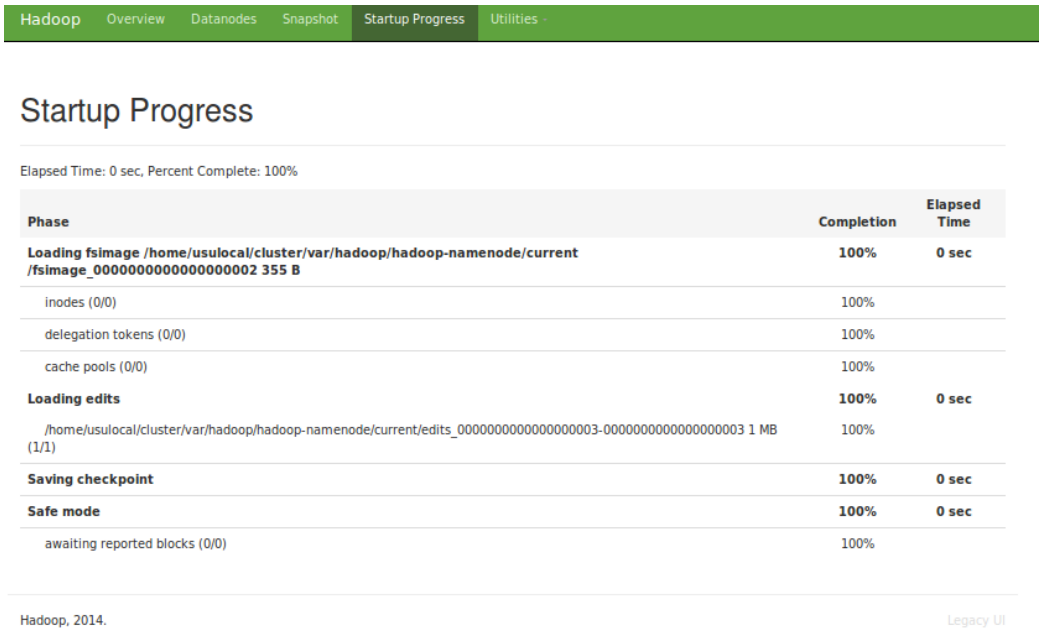
Esta interface web nos proporciona un menú donde podemos ver diferentes parámetros de nuestro cluster, por ejemplo, la relación de los workers que tenemos disponibles en nuestro sistema (pestaña *datanodes*), con información como su estado, la capacidad de memoria de disco que tiene disponible, la memoria que tiene utilizada, etc.



Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
lxeip6021 (158.42.214.88:50010)	1	In Service	172.89 GB	64 KB	24.4 GB	148.49 GB	3	64 KB (0%)	0	2.6.2
lxeip6020 (158.42.215.66:50010)	2	In Service	172.89 GB	68 KB	41.58 GB	131.31 GB	4	68 KB (0%)	0	2.6.2
lxeip6023 (158.42.215.132:50010)	2	In Service	172.89 GB	56 KB	38.03 GB	134.87 GB	2	56 KB (0%)	0	2.6.2
lxeip6025 (158.42.214.24:50010)	2	In Service	172.89 GB	64 KB	61 GB	111.89 GB	3	64 KB (0%)	0	2.6.2
lxeip6003 (158.42.215.28:50010)	2	In Service	172.89 GB	60 KB	26.46 GB	146.43 GB	3	60 KB (0%)	0	2.6.2
lxeip6002 (158.42.215.14:50010)	2	In Service	172.89 GB	140 KB	28.58 GB	144.31 GB	2	140 KB (0%)	0	2.6.2
lxeip6024 (158.42.214.91:50010)	1	In Service	172.89 GB	76 KB	29.35 GB	143.54 GB	4	76 KB (0%)	0	2.6.2
lxeip6027 (158.42.215.54:50010)	2	In Service	172.89 GB	68 KB	53.41 GB	119.48 GB	3	68 KB (0%)	0	2.6.2
lxeip6005 (158.42.215.30:50010)	0	In Service	172.89 GB	80 KB	46.13 GB	126.76 GB	5	80 KB (0%)	0	2.6.2
lxeip6026 (158.42.215.99:50010)	2	In Service	172.89 GB	60 KB	60.5 GB	112.39 GB	3	60 KB (0%)	0	2.6.2
lxeip6004 (158.42.215.29:50010)	2	In Service	172.89 GB	76 KB	23.76 GB	149.13 GB	5	76 KB (0%)	0	2.6.2
lxeip6018 (158.42.215.64:50010)	2	In Service	172.89 GB	52 KB	63.14 GB	109.75 GB	2	52 KB (0%)	0	2.6.2
lxeip6017 (158.42.215.63:50010)	1	In Service	172.89 GB	108 KB	23.92 GB	148.97 GB	8	108 KB (0%)	0	2.6.2
lxeip6019 (158.42.215.65:50010)	2	In Service	172.89 GB	52 KB	60.95 GB	111.94 GB	2	52 KB (0%)	0	2.6.2
lxeip6030 (158.42.215.197:50010)	1	In Service	172.89 GB	60 KB	32.34 GB	140.56 GB	3	60 KB (0%)	0	2.6.2
lxeip6032 (158.42.215.228:50010)	1	In Service	172.89 GB	100 KB	41.44 GB	131.45 GB	6	100 KB (0%)	0	2.6.2
lxeip6010 (158.42.215.56:50010)	0	In Service	172.89 GB	84 KB	24.41 GB	148.48 GB	4	84 KB (0%)	0	2.6.2
lxeip6031 (158.42.215.221:50010)	2	In Service	172.89 GB	52 KB	35.78 GB	137.11 GB	2	52 KB (0%)	0	2.6.2
lxeip6012 (158.42.215.58:50010)	1	In Service	172.89 GB	80 KB	23.74 GB	149.15 GB	5	80 KB (0%)	0	2.6.2
lxeip6011 (158.42.215.57:50010)	2	In Service	172.89 GB	44 KB	56.85 GB	116.04 GB	1	44 KB (0%)	0	2.6.2
lxeip6014 (158.42.215.59:50010)	2	In Service	172.89 GB	56 KB	33.48 GB	139.41 GB	2	56 KB (0%)	0	2.6.2

Imagen 20 - Información de worker nodes

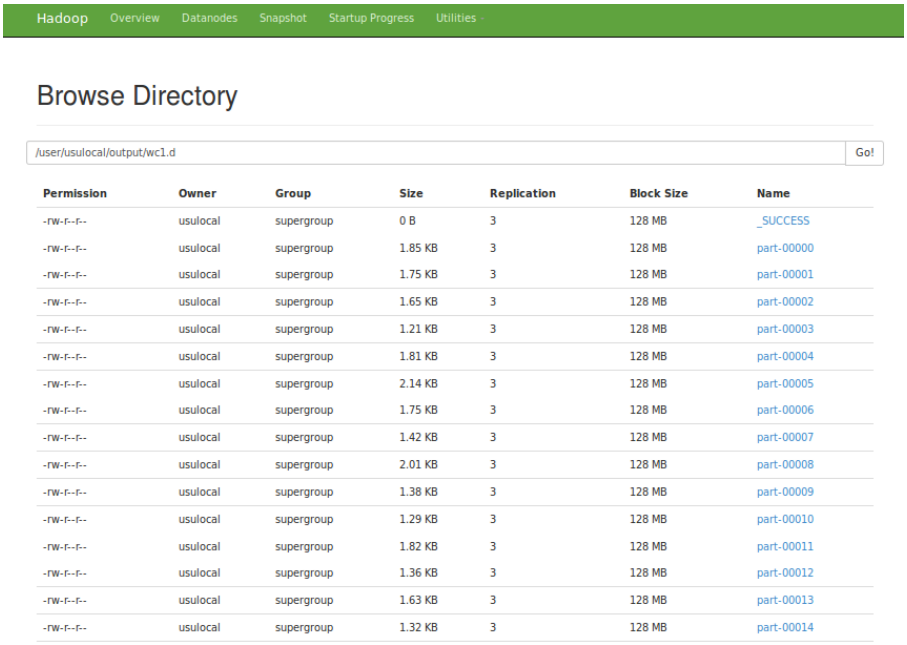
Podemos ver cuál ha sido el proceso de arranque del Cluster (Startup progress). En él se puede observar como primero se carga el *fsimage* y después se aplican los cambios en *loading edits*.



Phase	Completion	Elapsed Time
Loading fsimage /home/usulocal/cluster/var/hadoop/hadoop-namenode/current /fsimage_0000000000000000002 355 B	**100%**	**0 sec**
inodes (0/0)	100%	
delegation tokens (0/0)	100%	
cache pools (0/0)	100%	
Loading edits	**100%**	**0 sec**
/home/usulocal/cluster/var/hadoop/hadoop-namenode/current/edits_0000000000000000003-0000000000000000003 1 MB (1/1)	100%	
Saving checkpoint	**100%**	**0 sec**
Safe mode	**100%**	**0 sec**
awaiting reported blocks (0/0)	100%	

Imagen 21 - Proceso de carga Hadoop

También podemos acceder a los directorios, carpetas y archivos de nuestro cluster con una herramienta grafica fácil de utilizar



Permission	Owner	Group	Size	Replication	Block Size	Name
-rw-r--r--	usulocal	supergroup	0 B	3	128 MB	_SUCCESS
-rw-r--r--	usulocal	supergroup	1.85 KB	3	128 MB	part-00000
-rw-r--r--	usulocal	supergroup	1.75 KB	3	128 MB	part-00001
-rw-r--r--	usulocal	supergroup	1.65 KB	3	128 MB	part-00002
-rw-r--r--	usulocal	supergroup	1.21 KB	3	128 MB	part-00003
-rw-r--r--	usulocal	supergroup	1.81 KB	3	128 MB	part-00004
-rw-r--r--	usulocal	supergroup	2.14 KB	3	128 MB	part-00005
-rw-r--r--	usulocal	supergroup	1.75 KB	3	128 MB	part-00006
-rw-r--r--	usulocal	supergroup	1.42 KB	3	128 MB	part-00007
-rw-r--r--	usulocal	supergroup	2.01 KB	3	128 MB	part-00008
-rw-r--r--	usulocal	supergroup	1.38 KB	3	128 MB	part-00009
-rw-r--r--	usulocal	supergroup	1.29 KB	3	128 MB	part-00010
-rw-r--r--	usulocal	supergroup	1.82 KB	3	128 MB	part-00011
-rw-r--r--	usulocal	supergroup	1.36 KB	3	128 MB	part-00012
-rw-r--r--	usulocal	supergroup	1.63 KB	3	128 MB	part-00013
-rw-r--r--	usulocal	supergroup	1.32 KB	3	128 MB	part-00014

Imagen 22 - Sistema de carpetas HDFS

Además, para cada uno de los bloques de cualquiera de nuestros archivos, podemos comprobar en qué worker-nodes se encuentra almacenado. Recordemos que, en función del índice de réplica que tengamos definido en el parámetro *dfs.replication* en nuestro fichero de configuración *hdfs-site.xml*, se realizarán un número determinado de copias de cada bloque en varios de los workers tal y como aparece en la imagen.

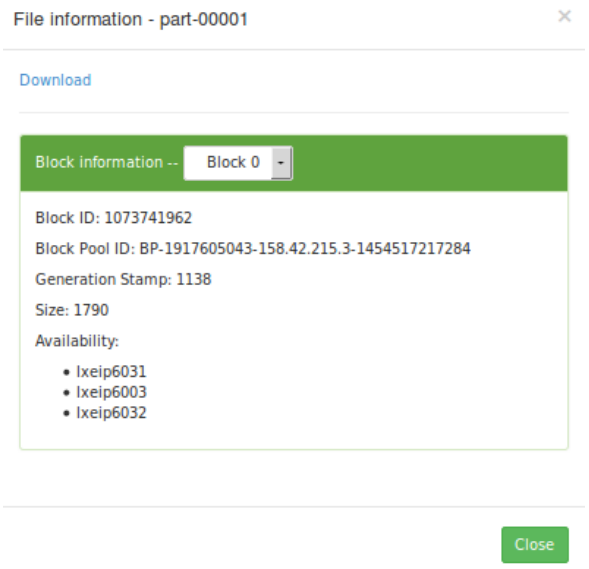


Imagen 23 - Ubicación de bloques de archivo

De la misma manera, para el control de nuestro Resource Manager con Yarn, también disponemos de una interface web que nos facilita el trabajo. En este caso, su dirección es http://<ResourceManager>:8088, lo que en nuestro caso se traduce por http://158.42.215.12:8088

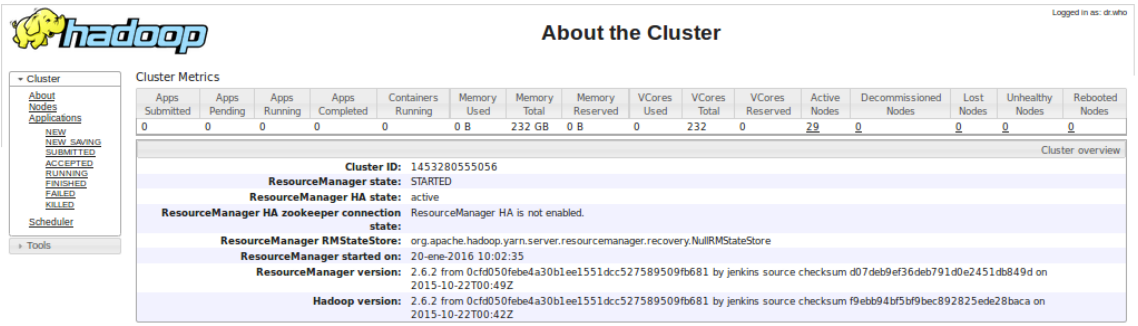


Imagen 24 - Características del cluster

Desde esta misma URL podemos ver también todos los nodos pertenecientes al clúster y el estado actual de cada uno de ellos.

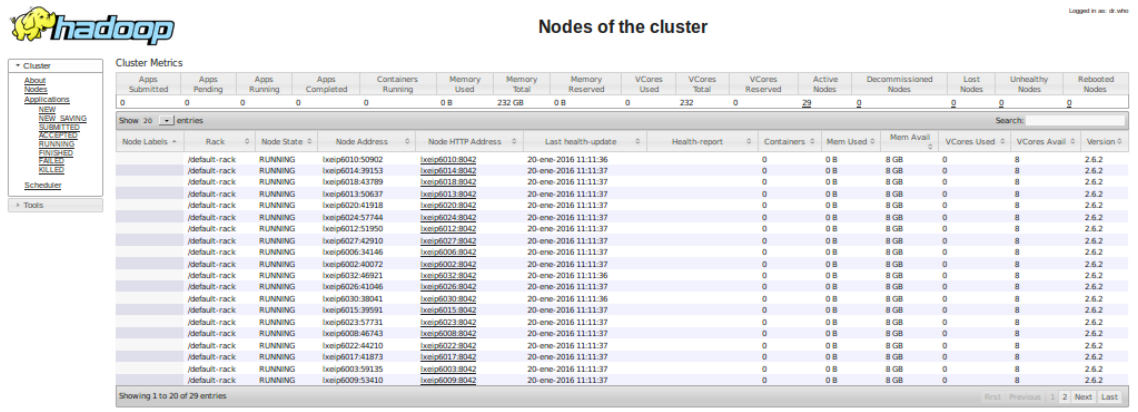


Imagen 25 - Recursos del clúster disponibles

Y podemos también ver la relación de trabajos aceptados por el Resource Manager y su estado de ejecución actual.

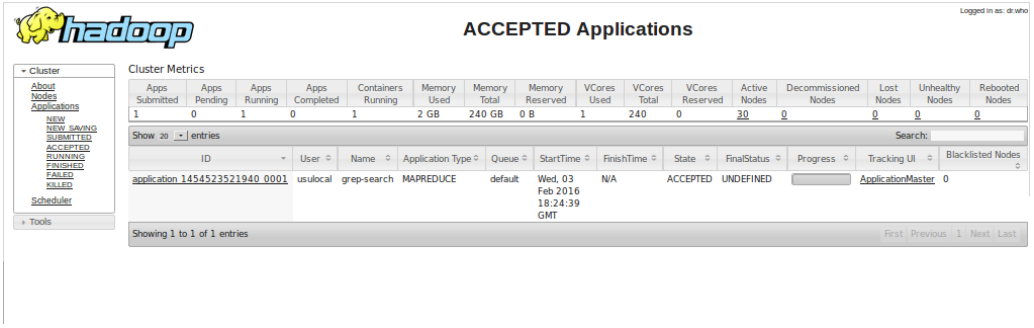
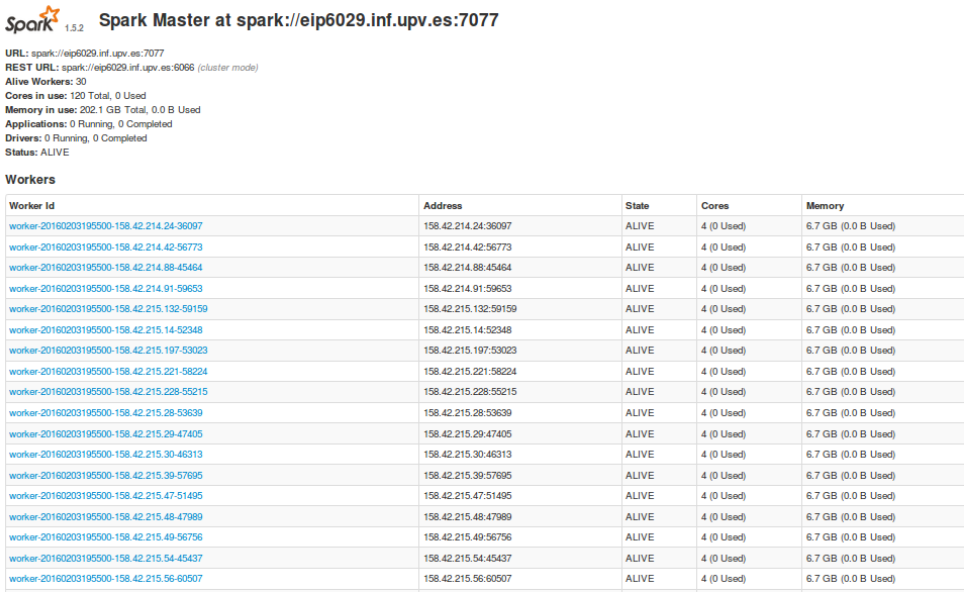


Imagen 26 - Trabajos aceptados por Yarn

7. Funcionamiento de Spark

Una vez operativo el sistema Spark, también disponemos de un interface web para controlar su evolución y funcionamiento. Este interface es accesible vía navegador web a través de la dirección http://158.42.215.3:8080, y nos proporciona, inicialmente, información de los workers que tenemos disponibles en nuestro cluster y de su estado actual:



Spark 1.5.2 **Spark Master at spark://eip6029.inf.upv.es:7077**

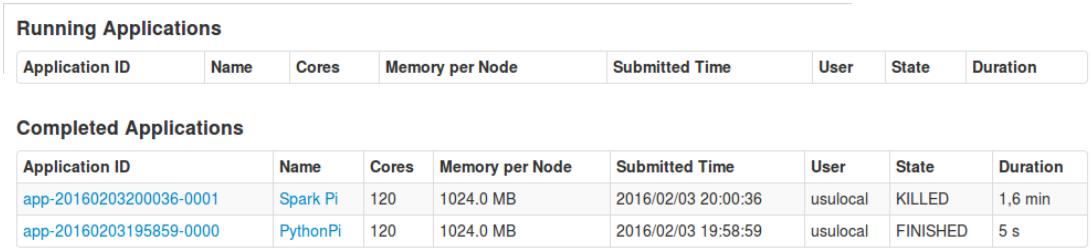
URL: spark://eip6029.inf.upv.es:7077
REST URL: spark://eip6029.inf.upv.es:6066 *(cluster mode)*
Alive Workers: 30
Cores in use: 120 Total, 0 Used
Memory in use: 202.1 GB Total, 0.0 B Used
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers

Worker Id	Address	State	Cores	Memory
worker-20160203195500-158.42.214.24-36097	158.42.214.24:36097	ALIVE	4 (0 Used)	6.7 GB (0.0 B Used)
worker-20160203195500-158.42.214.42-56773	158.42.214.42:56773	ALIVE	4 (0 Used)	6.7 GB (0.0 B Used)
worker-20160203195500-158.42.214.88-45464	158.42.214.88:45464	ALIVE	4 (0 Used)	6.7 GB (0.0 B Used)
worker-20160203195500-158.42.214.91-59653	158.42.214.91:59653	ALIVE	4 (0 Used)	6.7 GB (0.0 B Used)
worker-20160203195500-158.42.215.132-59159	158.42.215.132:59159	ALIVE	4 (0 Used)	6.7 GB (0.0 B Used)
worker-20160203195500-158.42.215.14-52348	158.42.215.14:52348	ALIVE	4 (0 Used)	6.7 GB (0.0 B Used)
worker-20160203195500-158.42.215.197-53023	158.42.215.197:53023	ALIVE	4 (0 Used)	6.7 GB (0.0 B Used)
worker-20160203195500-158.42.215.221-58224	158.42.215.221:58224	ALIVE	4 (0 Used)	6.7 GB (0.0 B Used)
worker-20160203195500-158.42.215.228-55215	158.42.215.228:55215	ALIVE	4 (0 Used)	6.7 GB (0.0 B Used)
worker-20160203195500-158.42.215.28-53639	158.42.215.28:53639	ALIVE	4 (0 Used)	6.7 GB (0.0 B Used)
worker-20160203195500-158.42.215.29-47405	158.42.215.29:47405	ALIVE	4 (0 Used)	6.7 GB (0.0 B Used)
worker-20160203195500-158.42.215.30-46313	158.42.215.30:46313	ALIVE	4 (0 Used)	6.7 GB (0.0 B Used)
worker-20160203195500-158.42.215.39-57695	158.42.215.39:57695	ALIVE	4 (0 Used)	6.7 GB (0.0 B Used)
worker-20160203195500-158.42.215.47-51495	158.42.215.47:51495	ALIVE	4 (0 Used)	6.7 GB (0.0 B Used)
worker-20160203195500-158.42.215.48-47989	158.42.215.48:47989	ALIVE	4 (0 Used)	6.7 GB (0.0 B Used)
worker-20160203195500-158.42.215.49-56756	158.42.215.49:56756	ALIVE	4 (0 Used)	6.7 GB (0.0 B Used)
worker-20160203195500-158.42.215.54-45437	158.42.215.54:45437	ALIVE	4 (0 Used)	6.7 GB (0.0 B Used)
worker-20160203195500-158.42.215.56-60507	158.42.215.56:60507	ALIVE	4 (0 Used)	6.7 GB (0.0 B Used)

Imagen 27 - Workers disponible Spark

También podemos ver en la parte inferior, tanto los trabajos que se están ejecutando actualmente como los trabajos que han sido ejecutados con anterioridad.



Running Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration

Completed Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20160203200036-0001	Spark Pi	120	1024.0 MB	2016/02/03 20:00:36	usulocal	KILLED	1,6 min
app-20160203195859-0000	PythonPi	120	1024.0 MB	2016/02/03 19:58:59	usulocal	FINISHED	5 s

Imagen 28 - Relación trabajos cluster

Además, para cada uno de los trabajos podemos ir ampliando la información suministrada para ver qué workers han intervenido en su ejecución, etc. pudiendo incluso llegar a los detalles de que RDD[24] han sido enviados a cada uno de los workers.

[24] Resilient Distributed Dataset: Conjunto de datos distribuidos tolerantes a fallos

Spark 1.5.2 **Application: GMM-02**

ID: app-20160203203520-0013
Name: GMM-02
User: usulocal
Cores: Unlimited (120 granted)
Executor Memory: 1024.0 MB
Submit Date: Wed Feb 03 20:35:20 CET 2016
State: RUNNING
Application Detail UI

Executor Summary

ExecutorID	Worker	Cores	Memory	State	Logs
22	worker-20160203195500-158.42.215.59-51342	4	1024	RUNNING	stdout stderr
26	worker-20160203195500-158.42.214.42-56773	4	1024	RUNNING	stdout stderr
15	worker-20160203195500-158.42.215.60-34221	4	1024	RUNNING	stdout stderr
8	worker-20160203195500-158.42.215.62-54141	4	1024	RUNNING	stdout stderr
11	worker-20160203195500-158.42.215.54-45437	4	1024	RUNNING	stdout stderr
1	worker-20160203195500-158.42.214.24-36097	4	1024	RUNNING	stdout stderr
23	worker-20160203195500-158.42.215.29-47405	4	1024	RUNNING	stdout stderr
19	worker-20160203195500-158.42.215.49-56756	4	1024	RUNNING	stdout stderr
5	worker-20160203195500-158.42.215.221-58224	4	1024	RUNNING	stdout stderr

Imagen 29 - Relación de workers para un Job

Spark dispone de dos sistemas diferenciados de funcionamiento:

- API´s. Spark proporciona API´s de programación para los lenguajes Scala. Python, Java y R. Como se puede ver en la gráfica, las líneas de código producidas para Spark han ido creciendo durante los últimos años, de manera que en Septiembre de 2015, se estimaban más de 600.000 líneas de código para Spark. (19)

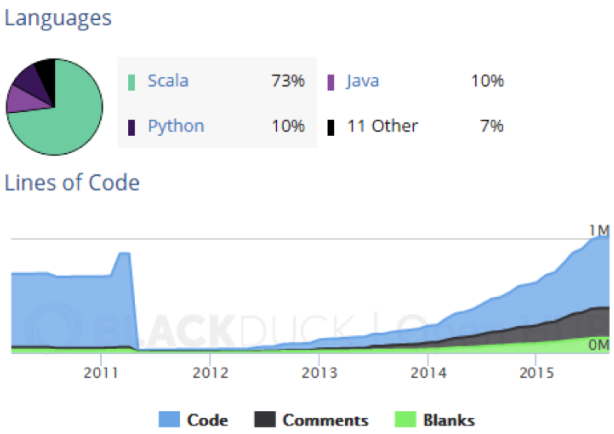


Tabla 12 - Proporción código Spark

- El otro método para lanzar ejecuciones a Spark es a través de su Spark-shell al que podemos llamar con la instrucción *bin/spark-shell*. En el Shell podemos trabajar directamente con lenguaje Scala

```
16/03/08 12:02:39 INFO util.Utils: Successfully started service 'HTTP class serv
er' on port 44741.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.5.2
      /_/

Using Scala version 2.10.4 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_25)
Type in expressions to have them evaluated.
Type :help for more information.
```

Imagen 30 - Spark-shell

7.1. Lanzamiento trabajos en Spark

Para lanzar trabajos en nuestro clúster Spark lo podemos realizar a través de la línea de comandos. En la instrucción de lanzamiento podemos modificar algunos parámetros por defecto como el número de nodos que ejecutarán el trabajo, los cores que utilizarán o las cantidades de memoria RAM que cada uno de ellos dedicará a nuestro trabajo. Por supuesto, los resultados obtenidos, en cuanto a velocidad y rendimiento, dependen de estos parámetros. Estos parámetros son también configurables en el propio programa a través de funciones incluidas en el API. Para que todo funcione correctamente y el sistema sea capaz de localizar todos los ficheros de configuración del clúster, es necesario que nos aseguremos de tener bien definida la variable de entorno *HADOOP_CONF_DIR*. (20)

```
$ ./bin/spark-submit --class path --master yarn-cluster [options]
<app jar> [app options]
```

Por ejemplo:

```
$ ./bin/spark-submit --class org.apache.spark.examples.SparkPi \
    --master yarn-cluster \
    --num-executors 3 \
    --driver-memory 4g \
    --executor-memory 2g \
    --executor-cores 1 \
    lib/spark-examples*.jar \
    10
```

donde los parámetros significan:

- Class → Punto de entrada de la aplicación a ejecutar.
- Master → La dirección *url* del clúster Spark
    - *spark://node1:7077* para el modo Standalone
    - *yarn-cluster* para el modo clúster sobre yarn
    - *yarn-client* para el modo cliente sobre yarn
    - *local[n]* Este modo se utiliza para la ejecución de las aplicaciones en el nodo local y con *n* número de cores.
- Conf → Los parámetros de configuración que necesitemos establecer. En nuestro caso
    - Num-executors → Numero de workers que participaran en la realización del trabajo.
    - Driver-memory → Cantidad de memoria del proceso que va a controlar la ejecución.
    - Executor-memory → Cantidad de memoria que destinará el worker a la realización del trabajo.
    - Executor-cores → Numero de cores que destinará el worker a realizar el trabajo.
- App-jar → El path al paquete *jar*[25], o a la aplicación *.py* para Python, de nuestra aplicación y todas sus dependencias.
- App-options → Cualquier parámetro que debamos mandar a nuestra aplicación.

[25] Jar: Archivo Java (**J**ava **AR**chive) que permite ejecutar aplicaciones escritas en Java

De todos los parámetros posibles, el más significativo es el parámetro –*master*. Este nos indica cual va a ser el modo de lanzamiento del trabajo sobre nuestro clúster. Este parámetro es básico y depende totalmente del tipo de trabajo que vamos a realizar y dónde y cómo están los datos que vamos a utilizar para el mismo.

Las aplicaciones distribuidas con las que vamos a trabajar realizan tres trabajos diferentes. En primer lugar, distribuyen un conjunto de datos entre todos sus trabajadores. A continuación, mandan a los trabajadores ejecutar algún tipo de código sobre esos datos que les han enviado. Por último, recogen los resultados de todos y los fusionan en un único resultado. La principal diferencia entre los modos de lanzamiento de Spark radica en quién y cómo se realiza la gestión de los recursos del cluster.

En MapReduce, la unidad de computación a más alto nivel es un trabajo (en adelante *job*) (21). El sistema carga los datos, se aplica una función *map*, se mezclan (*shuffle*), se aplica una función *reduce*, y se escriben los resultados en el sistema de almacenamiento. Los *jobs* en Spark tienen un funcionamiento similar (aunque un trabajo puede constar de más procesos y no solo *map* y *reduce*), pero añade una entidad superior que es la aplicación, que puede ejecutar varios *jobs* secuenciales o en paralelo. A diferencia de MapReduce, una aplicación tendrá procesos (*Executors*), que se ejecutan en los workers del cluster, incluso cuando no están ejecutando ninguna tarea, de esta manera, se pueden almacenar los datos en la memoria principal de los equipos para un acceso rápido a los mismos, y el inicio de las tareas es mucho más rápido.

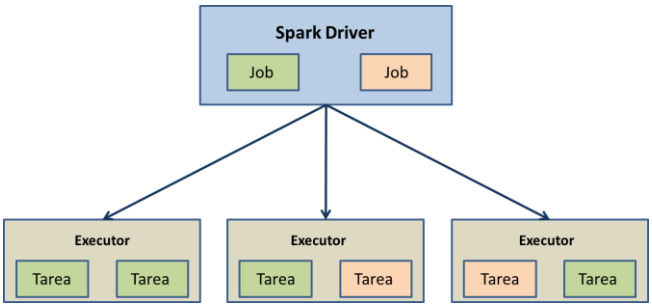


Imagen 31 - Reparto de tareas Spark

Sin embargo, la mayor desventaja es la gestión de recursos a alto nivel. Como el número de *executors* es fijo y cada *executor* tiene asignados unos recursos de forma fija, cada aplicación utiliza el mismo número de recursos para toda su duración. Sin embargo, Yarn es capaz de redimensionar los contenedores de ejecución y gestionar los recursos de forma dinámica.

El gestor del cluster es el encargado de iniciar todos los procesos *executor*. El desarrollador de aplicaciones para Spark no necesita preocuparse de cuál es el gestor de recursos sobre el que se está ejecutando Spark.

Spark puede funcionar sobre tres gestores de recursos, Mesos (del que no vamos a hablar en este documento), YARN y su propio modo *Standalone*. Los tres frameworks utilizan dos componentes, un servicio central (Resource Manager para Yarn, Spark Master para standalone) que determina qué aplicaciones lanzan procesos *executor*, así como cuándo y dónde se lanza su ejecución. También disponen de un servicio esclavo que se ejecuta en cada uno de los nodos (workers) del cluster (Node Manager para Yarn y Spark Slave para standalone) que son los que inician la ejecución de los *executors*.

La utilización de Yarn como gestor de recursos proporciona algunas ventajas sobre el modo *standalone* o incluso sobre el modo Mesos:

- Yarn permite compartir y configurar de forma centralizada todos los recursos de que dispone el cluster para todas las aplicaciones que se ejecuten en él. Se puede utilizar todo el cluster para una tarea de MapReduce, y luego solo una parte para una aplicación de Spark sin realizar ningún cambio en la configuración.
- El modo *standalone* de Spark, requiere que las aplicaciones lancen un *executor* en cada uno de los nodos del cluster, mientras que con Yarn puedes elegir el número de *executors* a utilizar.
- Por último, Yarn es el único gestor de recursos para Spark que soporta la gestión de la seguridad, utilizando Kerberos[26] y la autenticación segura entre procesos.

Cuando una aplicación Spark se ejecuta sobre Yarn, cada *executor* funciona como un *contenedor* de Yarn. Cuando MapReduce gestiona un contenedor y lanza una máquina virtual de Java (JVM) para cada una de las tareas, Spark puede alojar múltiples tareas en un solo contenedor. Esto le permite a Spark iniciar las tareas a mucha más velocidad que a MapReduce.

Por último, en el modo yarn-cluster, el driver se ejecuta sobre el Application Master, por lo que el mismo proceso es responsable tanto de la gestión del programa como de la solicitud de los recursos necesarios, y el proceso se ejecuta dentro del primer contenedor de Yarn. Es muy útil para procesos batch, pero no funciona bien para procesos interactivos.

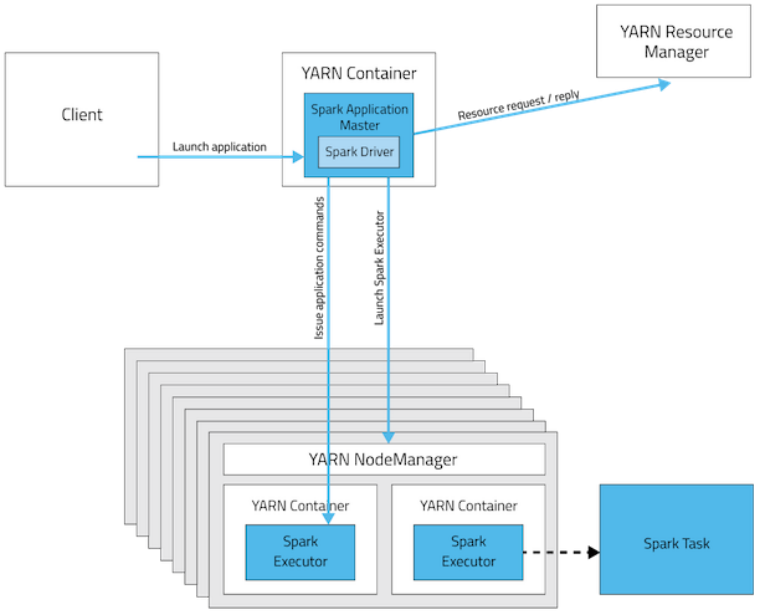


Imagen 32 - Funcionamiento yarn-cluster

En el modo `yarn-client`, el Application Master se limita a solicitar a Yarn los recursos necesarios, y el driver se ejecuta de forma independiente sobre el equipo cliente. Es muy útil para procesos de depurado o interactivos que requieran entrada del usuario.

[26] Kerberos: Un protocolo de autenticación de redes de ordenador que utiliza un tercero de confianza.

- Client (*yarn-client*): la aplicación (driver) se ejecutará sobre el equipo en el que se ha lanzado y será quien ejecute las labores de solicitud de recursos al Resource Manager, etc. Este equipo puede no pertenecer al clúster.
- Cluster (*yarn-cluster*): la aplicación (driver) será ejecutada sobre el Resource Manager de Yarn. Es la forma más frecuente de uso.

	Yarn Cluster	Yarn Client	Spark Standalone
El *driver* se ejecuta en	Application Master	Cliente	Cliente
Quien solicita recursos?	Application Master	Application Master	Cliente
Quien inicia los procesos de ejecución?	Yarn NodeManager	Yarn NodeManager	Slaves de Spark
Persistencia	Yarn Resource Manager	Yarn Resource Manager	Spark Master
Spark Shell ?	No	Si	Si

Imagen 33 – Gestión recursos en Spark

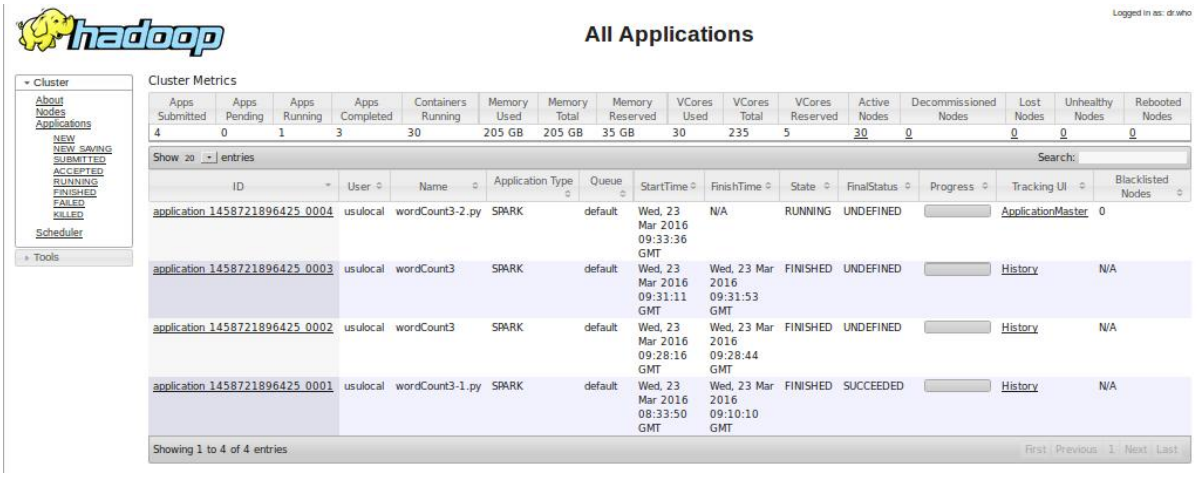


Imagen 34 - Cola de trabajos en Yarn

7.2. RDD´s

A diferencia de lo que ocurre con Hadoop HDFS (sistema de almacenamiento en disco distribuido), Spark proporciona a los programadores un conjunto de API´s para diferentes lenguajes de programación que se basan en el uso de estructuras de datos, llamadas RDD (Resilient Distributed Dataset).

Los RDD son colecciones de datos particionadas y distribuidas a través de un clúster que pueden ser reconstruidas si alguna de ellas se pierde (son tolerantes a fallos). Con ellas, las aplicaciones pueden crear conjuntos de datos que serán repartidos por los diferentes nodos del clúster para realizar las operaciones o cálculos necesarios para cada uno de ellos (paralelismo de datos).

Existen dos tipos básicos de operaciones sobre los RDD´s, el primero para realizar transformaciones sobre los mismos, y el segundo para realizar operaciones paralelas con ellos. Estas operaciones las realizaremos por programa utilizando la API proporcionada por Spark.

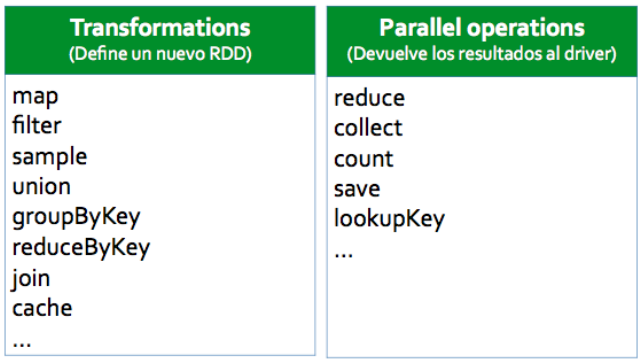


Imagen 35 - Operaciones con RDD´s

Los RDD se pueden crear a partir de datos en memoria o a partir de datos que se encuentren distribuidos en disco en nuestro sistema de almacenamiento HDFS (o cualquier otro sistema de archivos soportado por Hadoop, incluso el sistema de archivos local).

7.3. Programación con Spark (22)

Como ya he comentado en varias ocasiones en el presente documento, Spark proporciona API´s de programación para los lenguajes Java, Scala, Python y R, además de proporcionar un Shell interactivo para Python. La versión de Spark que hemos utilizado (la 1.5.2) soporta Java a partir de la versión 7, Scala a partir de la versión 2.10 y con las versiones 2.6+ y 3.4+ de Python. Por simplificar, los ejemplos que se utilicen serán siempre en Python.

A alto nivel, las aplicaciones Spark consisten en un programa *driver* que ejecuta la función principal de la aplicación de usuario y lanza varias operaciones paralelas en el clúster. El principal elemento abstracto que proporciona Spark son los RDD (de los que ya he hablado anteriormente) y que son colecciones de elementos de datos repartidos por los worker (nodos) de un clúster y que pueden ser tratados en paralelo. Los RDD´s son creados mediante la transformación de datos a partir de un fichero alojado en el sistema HDFS de Hadoop o a partir de una colección de datos en el programa *driver*. El usuario puede activar la persistencia de un RDD de manera que se reutilice de forma eficiente en las diferentes operaciones paralelas. Los RDD se recuperan de forma automática en caso de caída del nodo en el que están alojados.

En un siguiente nivel, nos encontramos con las variables compartidas que se pueden utilizar en las operaciones paralelas. Por defecto, cuando Spark ejecuta una función en paralelo como un conjunto de tareas en workers diferentes, envía una copia de cada variable utilizada en la función a cada una de las tareas. En ocasiones, una variable tiene que ser compartida entre las diferentes tareas o entre las tareas y el programa *driver*. Spark soporta dos tipos diferentes de variables compartidas, las variables de *broadcast* que pueden utilizarse para enviar un mismo

valor a la memoria de todos los nodos, y los *acumuladores* que son variables que se utilizan para los contadores, sumas, etc.

El primer paso que debe seguir cualquier aplicación para funcionar en Spark es crear el *SparkContext* que le proporciona al sistema información de cómo conectar con el clúster. Para ello, primero es necesario crear el objeto *SparkConf* que contiene información sobre la aplicación.

```
sc = SparkContext( appName="Mi Aplicacion" )
```

Existen dos formas de crear los RDD´s para una aplicación Spark, por un lado, paralelizando una colección de datos existente en memoria en nuestro programa *driver*, y por otro accediendo a un conjunto de datos en un sistema externo de almacenamiento, como algún sistema distribuido HDFS o compatible con Hadoop.

Las colecciones de datos se pueden paralelizar utilizando el método *parallelize* de nuestro SparkContext sobre un conjunto de datos en nuestro *driver*. Los elementos de una colección se copian para formar un conjunto distribuido que puede ser ejecutado en paralelo.

```
data = [1, 2, 3, 4, 5]
distData = sc.parallelize(data)
```

Una vez creado, este conjunto de datos distribuido ya puede ser ejecutado en paralelo. Podemos llamar a la función `distData.reduce(lambda a, b: a + b)` para sumar los elementos de la lista.

Un parámetro muy importante para trabajar con RDD es el número de particiones en que se divide un conjunto de datos. Spark lanza una tarea para cada partición de datos. Normalmente, necesitaremos entre 2 y 4 particiones para cada CPU del clúster (para optimizar los recursos). Spark intenta realizar de forma automática el número de particiones más adecuado en función de nuestro clúster, de todas formas, podemos forzar con un parámetro el número de particiones que queremos realizar `sc.parallelize(data, 10)`), donde el 10 indica el número de particiones. Con ello podemos conseguir que, determinados trabajos con pocos datos, sean repartidos de forma correcta y no se ejecuten en un único nodo.

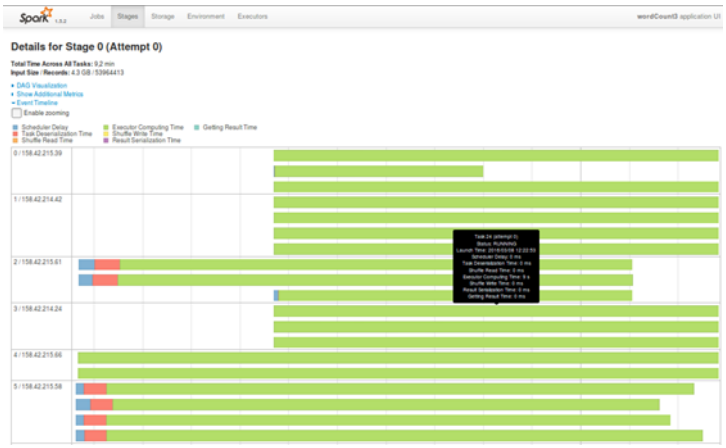


Imagen 36 - Planificación de Workers en Spark

También podemos crear nuestros RDD a partir de fuentes de datos externas que sean soportadas por Hadoop, como los sistemas de ficheros locales, HDFS, Cassandra[27], etc. Los RDD de fuentes de datos externas se crean usando el método *textFile* del *SparkContext*. Este método toma un fichero y lo lee como una colección de líneas. Una vez creada la colección de líneas, podemos operar sobre ellas con operaciones para conjuntos de datos como *map* y *reduce*.

```
distFile = sc.textFile("data.txt",numParticiones)
```

Para trabajar con ficheros en Spark, hemos de tener en cuenta los siguientes puntos:

- Si estamos utilizando un path del sistema de almacenamiento local, este fichero debe estar accesible en el mismo path local de los workers. Para ello deberemos copiar el fichero a los workers o utilizar una unidad de red.
- Todos los métodos de entrada relativos a ficheros soportan su ejecución en directorios, ficheros comprimidos y admiten caracteres comodín.
- El método *textFile* soporta un segundo argumento para controlar el número de particiones del fichero, `numParticiones`. Por defecto Spark crea una partición para cada bloque del fichero (de 64 o 125 MB para HDFS), pero puedes forzar para realizar un mayor número de particiones, pero nunca menos que el número de bloques.

Tal y como he comentado en el punto anterior, una vez creados los RDD, sobre ellos podemos realizar dos tipos de operaciones, las de *transformación* que crean un conjunto de datos a partir de otro ya existente y las de *acción* que devuelven algún valor al programa *driver* después de realizar alguna serie de operaciones. Por ejemplo, la función *map* es una transformación que ejecuta una función sobre cada conjunto de datos y devuelve un nuevo RDD con los resultados. Por el otro lado, *resume* es una acción que agrega todos los elementos de un RDD utilizando alguna función y devuelve el resultado final al programa *driver*.

Todas las transformaciones en Spark son *lazy*[28]y no calculan los resultados en el momento, solo recuerdan las transformaciones que deben aplicar a un conjunto de datos y las ejecutarán cuando una acción necesite su resultado, de esta forma se puede funcionar de forma más eficiente. Por defecto, cada transformación de un RDD se debe recalcular cada vez que le apliquemos una acción, pero podemos hacerla permanente utilizando el método *persist*, de manera que no hemos de calcularla cada vez.

En la tabla siguiente podemos ver algunas de los principales métodos de Spark tanto para transformación de RDD como para realizar acciones sobre los mismos. Las diferentes funciones serán llamadas de forma específica en cada uno de los lenguajes soportados.

Normalmente, cuando se ejecuta una función de Spark en un nodo remoto, se generan copias independientes de las variables que serán utilizadas en la función y se envían a la memoria local del nodo remoto. Si estas variables se modifican en el nodo remoto, no son reenviadas de vuelta al programa *driver*. Si necesitamos utilizar variables compartidas, podemos utilizar los tipos *broadcast* y *accumulator*.

[27] Cassandra: Base de datos distribuida y NoSQL (Not only SQL).
[28] Lazy: Perezoso. En informática, ejecución de algo solo en el momento en que es necesario. Lo más tarde posible.

	Método	Significado
Transformación	map(func)	Devuelve un nuevo RDD que resulta de aplicar a cada elemento de origen la función *func*
	filter(func)	Devuelve un nuevo RDD con los elementos que devuelven true en la función *func*
	flatMap(func)	Similar a map pero en lugar de un elemento devuelve 0 o más elementos
	union(otherDataset)	Devuelve un RDD resultado de unir el dataset original con *otherDataset*
	intersection(otherDataset)	Devuelve un RDD resultado de la intersección entre el dataset originaly *otherDataset*
	distinct([numTasks]))	Devuelve un nuevo RDD que contiene los elementos distintos del dataset original
	groupByKey([numTasks])	Si se llama sobre un RDD del tipo <clave,valor>, devuelve un nuevo RDD agrupado por el valor clave
	reduceByKey(func, [numTasks])	Cuando se realiza sobre un RDD de tipo <clave,valor>, devuelve otro dataset con formato <clave,valor> donde los valores de cadaclave son agregados según la
Acción	reduce(func)	Agregar los elementos del dataset utilizando la funcion *func*
	collect()	Devuelve al programa *driver* los elementos del dataset en forma de array
	first()	Devuelve el primer elemento del dataset
	take(n)	Devuelve un array con los primeros *n* elementos del dataset
	foreach(func)	Ejecuta la funcion *func* para cada elemento del dataset

Tabla 13 - Métodos de Spark

Las variables de tipo *broadcast* pueden ser utilizadas para enviar una variable de solo lectura a todos los nodos sin necesidad de enviarla con una tarea. Podemos enviar a cada nodo una copia de un dataset grande que necesite como entrada de una forma muy eficiente.

```
broadcastVar = sc.broadcast([1, 2, 3])
```

Los *accumulator* son variables que se generan con una operación asociativa, pueden ser utilizados para generar contadores o sumadores. Los acumuladores que soporta Spark son de tipo numérico, aunque por programación se pueden generar otros tipos de acumuladores. Los acumuladores se modifican únicamente con las acciones, y Spark garantiza que su valor se actualiza solamente una vez para cada tarea, aunque ésta sea reiniciada.

8. Casos de Uso

Para el desarrollo de las pruebas de funcionamiento, se han utilizado diferentes programas de ejemplo proporcionados en la propia página de Hadoop y de Spark para testeo del sistema. Como aplicaciones principales, la aplicación Word Count, que realiza diferentes cálculos contando las palabras y líneas a partir de unos ficheros fuentes alojados en Hadoop. Además, se ha utilizado también un programa del Master de Big Data de la ETSINF de clasificación de valores por medio del algoritmo GMM - Gaussian Mixture Models que, a diferencia del WordCount, genera su propio conjunto de valores.

Tal y como he ido comentando a lo largo de este documento, los principales pasos para realizar un análisis de información en *Big Data* son los siguientes:

- Aceptación del trabajo por parte del clúster.
- Reparto de los datos, normalmente a través de una función de paralelización. Este reparto de datos es muy importante y en ocasiones, el tiempo de respuesta va a depender de ese reparto.
- Ejecución de tareas por parte de los workers.
- Recepción de resultados por parte del *driver*.

8.1. Caso de uso Contar Palabras

La aplicación de contar palabras está desarrollada en Python y utiliza las directivas de Spark para realizar los cálculos. El sistema toma como entrada un grupo de ficheros alojados en el HDFS de Hadoop con un tamaño conjunto superior a los 22 GB de información y que están repartidos por todos los nodos de nuestro clúster, para contar las palabras, líneas, etc. que contienen en su conjunto. Además, algunas de estas tareas las realizará utilizando los métodos *map* y *reduce* que proporciona Spark.

La parte que más interés tiene para nosotros de esta aplicación no es su resultado final, sino como cambian sus tiempos de ejecución en función de cómo lancemos la misma y sobre que partes del clúster lo hagamos.

Como podemos ver en el código de la aplicación, ésta lee una serie de ficheros en un directorio de nuestro *clúster* y realiza las siguientes operaciones sobre ellos:

1. Cuenta el número de líneas totales con la función *count* (que devuelve el número de elementos que existen en cada una de nuestras RDD, en nuestro caso, corresponde a una línea).
2. Cuenta el número de líneas con una función *map* (para asignar el valor de referencia 1 a la unidad básica) y luego una función *reduce* (donde realiza la suma de los diferentes valores).
3. Cuenta el número de líneas no vacías también con una función *map* (igual que la anterior, pero si la longitud de la línea es mayor que 0, le asigna el

valor mínimo 1, y en caso contrario le asigna el valor 0, es decir, no la cuenta) y luego una *reduce*.

4. Cuenta las palabras de nuestros ficheros con una función *map reduce*.
5. Cuenta las palabras con un acumulador y la función *Word_counter_1*
6. Cuenta las palabras con un acumulador y la función *Word_counter_2*

```
import sys
from operator import add
from pyspark import SparkContext

def function_word_counter_1( line ):
    global word_counter
    for w in line.split(): word_counter += 1

def function_word_counter_2( line ):
    global word_counter
    word_counter += len(line.split())

if __name__ == "__main__":
    """
        Usage: wordCount3
    """
    verbose=False
    input_dir="hdfs://eip6029.inf.upv.es:9000/user/usulocal/fusion/f001.txt"
    output_dir="hdfs://eip6029.inf.upv.es:9000/user/usulocal/output/wc3-1.d"

    sc = SparkContext( appName="wordCount3" )
    file_contents = sc.textFile( input_dir ) # For loading the contents of all the files in a directory

    file_contents.persist()

    lines_count_1 = file_contents.count()
    lines_count_2 = file_contents.map( lambda line : 1 ).reduce( lambda c1,c2 : c1+c2 )
    lines_count_3 = file_contents.map( lambda line : 1 if len(line.strip()) > 0 else 0 ).reduce( lambda c1,c2 : c1+c2 )

    word_counts = file_contents.flatMap( lambda line : line.split() ).map( lambda w: (w, 1) ).reduceByKey( lambda c1,c2 : c1+c2 )
    word_counts.persist()
    count_3 = word_counts.map( lambda pair: pair[1] ).reduce( lambda c1,c2: c1+c2 )
    word_counts.saveAsTextFile( output_dir )

    if verbose:
        for (word,count) in word_counts.collect():
            print( " %10d %s " % (count,word) )

    print( "Procesadas un total de %d lineas con count()"    % lines_count_1 )
    print( "Procesadas un total de %d lineas con map+reduce" % lines_count_2 )
    print( "Procesadas un total de %d lineas no vacias con map+reduce" % lines_count_3 )

    print( "Palabras contadas con map.reduce:     %d " % count_3 )

    word_counter = sc.accumulator(0)
    file_contents.foreach( function_word_counter_1 )
    print( "Palabras contadas con acumulador: %d " % word_counter.value )

    word_counter.value = 0
    file_contents.foreach( function_word_counter_2 )
    print( "Palabras contadas con acumulador: %d " % word_counter.value )

    word_counts.unpersist()
    file_contents.unpersist()

    sc.stop()
```

Imagen 37 - Código de aplicación Word Count

La aplicación la lanzamos con el siguiente script, donde modificamos los valores de memoria a utilizar y el número de cores disponibles, para poder tener diferentes comparativas:

```
#!/bin/bash

# Run a Python application on a Spark Standalone cluster


time spark-submit --master spark://eip6029.inf.upv.es:7077  --executor-memory 7G  --total-executor-cores 240 wordCount3-2.py 2>wc3.log
```

Imagen 38 - Script de ejecución de WordCount

Los resultados obtenidos en la función de conteo son los de la imagen 39 (a continuación). Como podemos observar, se trata de mucha cantidad de información, ya que el sistema ha

contado 175.246.600 líneas de texto y 3.864.235.685 de palabras, lo que nos da una idea de la envergadura de los ficheros.

```
usulocal@lxeip6001 ~/examples $
usulocal@lxeip6001 ~/examples $ ./spark-2-wc3-2.sh
Procesadas un total de 175246600 lineas con count()
Procesadas un total de 175246600 lineas con map+reduce
Procesadas un total de 173934560 lineas no vacias con map+reduce
Palabras contadas con map.reduce:     3864235685
Palabras contadas con acumulador: 3864235685
Palabras contadas con acumulador: 3864235685
```

Imagen 39 - Resultados Word Count

De las 6 tareas que tiene que realizar nuestro algoritmo, podemos observar (con tiempos parciales) que la más costosa de todas es la tarea número 1, el cálculo del número de líneas con la función *count()*. Le sigue (siempre en términos de tiempo de ejecución), la numero 4, conteo de palabras con *map reduce*. Luego, con tiempos bastante parecidos, aparecen las tareas 5 y 6, y por último, también con tiempos parecidos, las tareas 2 y 3.

La aplicación se ha lanzado 5 veces en cada uno de los diferentes modos para poder obtener unos tiempos medios representativos de la ejecución de la misma. Se han realizado variaciones en los modos de uso de memoria por nodo (2 GB o 6GB) y el número de cores utilizado. Esto correspondería con el número de equipos utilizado, ya que cada equipo tiene 4 cores, y se ha lanzado sobre 120 cores (30 equipos) y sobre 40 cores (10 equipos). Los resultados obtenidos han sido los siguientes (los tiempos están tomados en minutos):

Metodo	Memoria Nodo	Nodos ejecución	Tiempo Medio	Speed Up	Eficiencia
Standalone	6 GB	1	165	1,00	1,00
Yarn-cluster	6 GB	10	18,2	9,07	0,91
Yarn-cluster	6 GB	20	12,27	13,45	0,67
Yarn-cluster	6 GB	30	11,02	14,97	0,50
Standalone	6 GB	30	3,44	47,97	1,60
Standalone	2 GB	30	5	33,00	1,10
Standalone	6 GB	10	6,1	27,05	2,70
Standalone	2 GB	10	7,7	21,43	2,14

Tabla 14 - Tabla tiempos Word Count

Si definimos el *Speed Up* con la fórmula:

$$S = \frac{t(n)}{t(n')}$$

donde $t(n)$ es el tiempo base de ejecutar la aplicación en local en un equipo (primera línea de la tabla) y $t(n')$ es el tiempo del nuevo proceso en el clúster, observamos que dentro del clúster, obtendríamos un *Speed Up*[29] de entre 10 y 50, siempre en función del número de nodos que estuviéramos utilizando y de la cantidad de memoria de cada uno de los nodos que asignáramos.

[29] Speed up: Ganancia de velocidad que se consigue en una versión sobre otra.

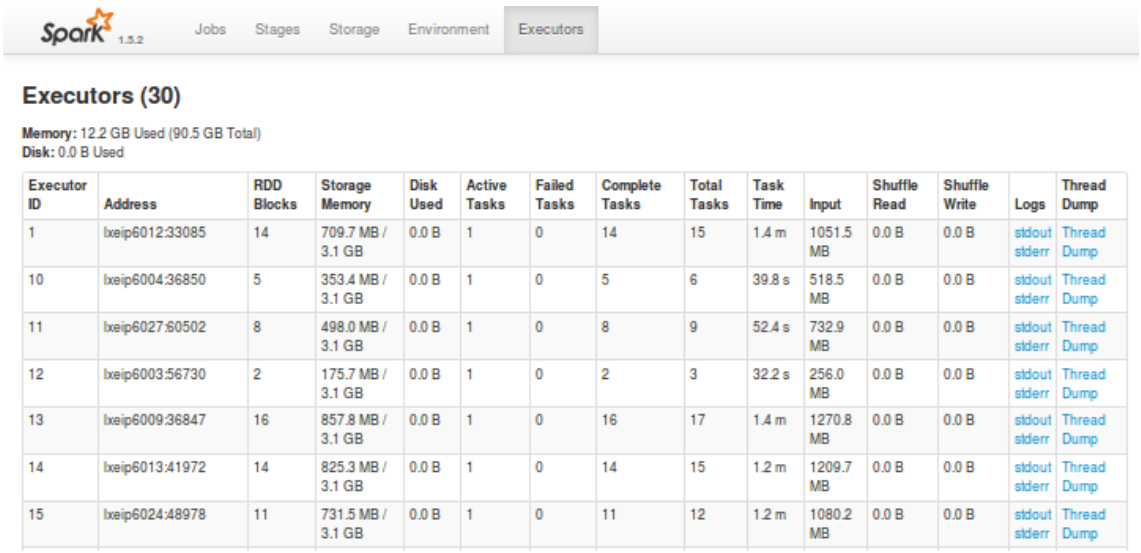
Podría parecer que si lanzamos una aplicación en modo *local* sobre un nodo y lanzamos la misma aplicación sobre los 30 nodos del clúster, esta debería ir 30 veces más rápido, pero esto no es cierto.

Hemos de tener en cuenta que, en nuestro caso, la versión *local* está fuertemente penalizada por la transmisión de la información. Nuestros 22 GB de información, no están físicamente en el nodo que está ejecutando la aplicación, sino en diferentes bloques de diferentes ficheros distribuidos a lo largo de nuestro clúster HDFS, por lo que deben viajar desde el resto de equipos del clúster a través de la LAN a nuestro nodo *local*, y eso incrementa mucho el tiempo de ejecución.

Sin embargo, definiendo la *Eficiencia*[30] con la siguiente formula:

$$E = \frac{S}{p}$$

Donde la S es el *speed up* obtenido y la p es el número de nodos que hemos utilizado en la ejecución, observamos que el lanzamiento más *eficiente* es el de 6GB y 10 nodos, es decir, es el que mejor aprovecha los recursos que tiene disponibles. Para poder explicarlo, de nuevo volvemos a los ficheros de datos y el tipo de almacenamiento que estamos utilizando.



Executor ID	Address	RDD Blocks	Storage Memory	Disk Used	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time	Input	Shuffle Read	Shuffle Write	Logs	Thread Dump
1	lxeip6012:33085	14	709.7 MB / 3.1 GB	0.0 B	1	0	14	15	1.4 m	1051.5 MB	0.0 B	0.0 B	stdout stderr	Thread Dump
10	lxeip6004:36850	5	353.4 MB / 3.1 GB	0.0 B	1	0	5	6	39.8 s	518.5 MB	0.0 B	0.0 B	stdout stderr	Thread Dump
11	lxeip6027:60502	8	498.0 MB / 3.1 GB	0.0 B	1	0	8	9	52.4 s	732.9 MB	0.0 B	0.0 B	stdout stderr	Thread Dump
12	lxeip6003:56730	2	175.7 MB / 3.1 GB	0.0 B	1	0	2	3	32.2 s	256.0 MB	0.0 B	0.0 B	stdout stderr	Thread Dump
13	lxeip6009:36847	16	857.8 MB / 3.1 GB	0.0 B	1	0	16	17	1.4 m	1270.8 MB	0.0 B	0.0 B	stdout stderr	Thread Dump
14	lxeip6013:41972	14	825.3 MB / 3.1 GB	0.0 B	1	0	14	15	1.2 m	1209.7 MB	0.0 B	0.0 B	stdout stderr	Thread Dump
15	lxeip6024:48978	11	731.5 MB / 3.1 GB	0.0 B	1	0	11	12	1.2 m	1080.2 MB	0.0 B	0.0 B	stdout stderr	Thread Dump

Imagen 40 - Trabajadores ejecutando tareas

Nuestros ficheros de datos están alojados en el clúster, pero no sabemos en cuantos y en que nodos están alojados, por lo que si ocupan espacio solo en unos pocos nodos, será más *eficiente* lanzar la aplicación solo sobre esos nodos para minimizar el tráfico de datos que tienen que circular por la red LAN. Si están alojados en unos pocos nodos, digamos los nodos del 1 al 10 y al lanzar la aplicación lo hiciéramos sobre los nodos 20 al 30, deberíamos mover toda la información de unos nodos a otros para realizar los cálculos.

[30] Eficiencia: Grado de aprovechamiento que la aplicación hace sobre los nuevos recursos del sistema.

También podemos llegar a la conclusión, a partir de estos datos, de que la ejecución de la aplicación sobre el clúster de Spark y con gestión de recursos por el propio sistema Spark, es de media unas 3 veces más rápido que con gestión de los recursos por el sistema Yarn proporcionado por Hadoop. Esto se ve fácilmente si comparamos las ejecuciones tanto en modo *yarn-cluster* como en modo *standalone* para 10 nodos y 6 GB, que en el segundo caso el tiempo es tres veces inferior.

Además, en cuanto a la propia gestión de los recursos del clúster, se ve que Spark es más eficiente en la gestión, ya que las cifras de eficiencia para los lanzamientos gestionados por Spark son siempre muy superiores a las gestionadas por Yarn, incluso, a medida que aumenta el número de nodos, la eficiencia de los gestionados por Yarn baja hasta un 0,5, lo que demuestra que utiliza peor los recursos a su disposición.

8.2. Caso de uso Gaussian Mixture Models

La aplicación GMM (Gaussian Mixture Models), está también desarrollada en Python y utiliza las funciones y métodos proporcionados por la API de Spark, aunque funciona de forma diferente a la aplicación de conteo de palabras. En primer lugar, no utiliza almacenamiento secundario de los equipos y utiliza únicamente datos en memoria RAM.

La aplicación, en primer lugar, genera unos conjuntos de datos de entrenamiento y de test, a partir de unas funciones definidas en una librería externa. Estos datos, son generados en memoria RAM del equipo *driver* y lo hace utilizando un parámetro que es el `num_classes` que luego modificaremos.

Una vez generados estos datos, llama a la función `sc.parallelize` con la que realiza el reparto de los datos a los diferentes nodos del clúster. A continuación, realiza una serie de bucles `while` para optimizar los cálculos y aproximar los resultados al índice de error determinado.

Una de las consecuencias de los repartos de datos y sistemas de ejecución de Spark explicados en el punto 7.1 se puede ver perfectamente en esta aplicación. Cuando generamos los datos de prueba con un `num_classes` pequeño (como aparece en la imagen, de 15) el volumen de datos generado no es muy grande. El sistema intenta generar entre 2 y 4 RDD´s para cada *executor*, pero al ser pocos datos genera solo 2 RDD, por lo que necesita solo 1 *executor*. Aunque luego el volumen de datos y cálculos a realizar va creciendo, el hecho de utilizar una asignación fija de recursos le impide utilizar más *executors*, por lo que se generan múltiples *jobs* que se ejecutan secuencialmente en un único nodo del *clúster*.

Para resolverlo podemos utilizar tres métodos. Por un lado, podemos incrementar el valor de la variable `num_classes` lo que generará conjuntos de datos mayores que generaran más RDD´s. Por otro lado, podemos forzar el número de RDD´s generados con el parámetro opcional de la instrucción `sc.parallelize(data, n)`, donde la `n` indica el número de particiones a realizar. Yo voy a utilizar el tercer método, que consiste en utilizar los dos sistemas anteriores.

```
import sys
import numpy

try:
    import cPickle as pickle
except:
    import pickle

from pyspark import SparkContext

from utils import generate_datasets
import utils

# --------------------------------------------------------------------------
def add_sample( gmm, sample ):
    global _b_gmm
    stable_gmm = _b_gmm.value
    gmm.accumulate_sample( sample, stable_gmm )
    return gmm
# --------------------------------------------------------------------------

# --------------------------------------------------------------------------
def combine_gmms( gmm1, gmm2 ):
    gmm1.add(gmm2)
    return gmm1
# --------------------------------------------------------------------------

if __name__ == "__main__":
    """
        Usage: GMM-02
    """
    sc = SparkContext( appName="GMM-02" )

    verbose=False

    log_file = open( "log/OUT", 'w' )

    num_classes=500
    X_train,Y_train,X_test,Y_test = generate_datasets.generate_multivariate_normals( num_classes, 7, 115, 25, 15.0, 12.0 )

    covar_type='diagonal'
    gmm = utils.GMM( n_components=1, dim=X_train.shape[1], covar=covar_type )
    gmm.initialize_from( X_train )

    X_train_p = sc.parallelize( X_train,60)

    X_train_p.persist()
```

```
    max_components = 100 if covar_type in ['diagonal'] else 50
    logL=0.0
    while gmm.n_components <= max_components:
        epsilon = 1.0e-5
        print ( "   N Componentes  %5d" % (gmm.n_components))
        iteration=1
        iterations_at_least=10
        relative_improvement = 1.0
        while iterations_at_least > 0:
            _b_gmm = sc.broadcast( gmm )
            _temp_gmm = X_train_p.aggregate( utils.GMM( gmm.n_components, gmm.dim, gmm.covar ), add_sample, combine_gmms )
            old_logL=logL
            logL = _temp_gmm.log_likelihood / len(X_train)

            #
            relative_improvement = abs( (logL-old_logL) / logL )
            gmm.update_parameters( _temp_gmm )
            print( "iteration %5d  logL = %e  delta_logL = %e\n" % (iteration, logL, relative_improvement) )
            log_file.write( "iteration %5d  logL = %e  delta_logL = %e\n" % (iteration, logL, relative_improvement) )
            log_file.flush()
            iteration=iteration+1
            if relative_improvement < epsilon: iterations_at_least = iterations_at_least-1
        #
        log_file.write( "n_components %5d  logL = %e \n" % (gmm.n_components, logL) )
        log_file.flush()
        gmm.save_to_text( 'data/gmm' )
        gmm.split()
    # --------------------------------------------------------------------------
    log_file.write( "task completed when %d components where exeeceded with %d\n" % (max_components, gmm.n_components) )
    log_file.flush()

    X_train_p.unpersist()

    sc.stop()

    log_file.close()
```

Imagen 41 - Código de aplicación GMM

Lanzamos varias ejecuciones de la aplicación utilizando diferentes parámetros para los valores mencionados y las medias de tiempos (tomados en minutos) son los de la tabla 15 a continuación. En este caso, al ir modificando el valor de la variable `num_classes` entre lanzamientos, no tiene mucho sentido comparar entre ellos, ya que los volúmenes de datos a manejar son diferentes.

Los cálculos, tanto del *speed up* como de la *eficiencia* han sido realizado siempre contra los lanzamientos que tienen el mismo valor para la variable `num_classes`, tomando como caso base para los cálculos el que tiene el menor número de *executors* asignado.

num_classes	Numero *executors*	*parallelize*	Tiempo Medio	Speed Up	Eficiencia
15	1	-	7,2	1,00	1,00
15	3	10	2,2	3,27	1,09
15	4	15	1,8	4,00	1,00
40	5	20	3,3	1,00	1,00
40	10	40	2,5	1,32	0,66
150	15	60	4,2	1,00	1,00
150	30	120	4,4	0,95	0,48
500	15	60	3,5	1,00	1,00
500	30	120	3,3	1,06	0,53
500	30	250	4,4	0,80	0,40

Tabla 15 - Comparativa resultados GMM

En este caso, es muchos más difícil evaluar los datos de *speed up* y *eficiencia*, ya que en alguna de las pruebas, generar muchos RDD´s (con el parámetro de *parallelize* alto) no supone ninguna ventaja sino todo lo contrario, el sistema necesita crear más *jobs* y más *tareas*, por lo que incrementa el número de procesos que, por su volumen de datos, no se justifican.

En este caso, sí que obtenemos una mejora sustancial en cuanto al tiempo medio de respuesta, aunque deberíamos incrementar mucho el volumen de los datos para utilizar adecuadamente todos los nodos de nuestro clúster.

Sin embargo, a pesar de obtener mejoras en los tiempos de respuesta, la eficiencia va siempre disminuyendo a medida que aumentan los nodos que ejecutan las aplicaciones.

9. Conclusiones

El Big Data está cobrando una gran relevancia en todos los sectores económicos de la sociedad, no solo en las grandes empresas tecnológicas que proporcionan servicios *en la nube* a sus millones de clientes (Google, Facebook, Twitter, Yahoo, Amazon, etc.) sino, cada vez más, a compañías de todo tipo (entidades financieras, aseguradoras, empresas de telecomunicaciones, administraciones públicas, investigación, etc.) y, poco a poco, a empresas de otro tipo de sectores y tamaños cada vez menores.

Por un lado, necesitamos almacenar todas estas cantidades de datos que somos capaces de generar con todos los dispositivos electrónicos de que disponemos actualmente, pero, sobre todo, si queremos que esos datos se conviertan en información y algún día formen parte del conocimiento, necesitamos procesar esos datos en un tiempo razonable y de una forma efectiva.

Google, Facebook, etc. obtienen grandes rendimientos del análisis de nuestra información, nuestros gustos, nuestras costumbres. Las entidades financieras necesitan saber cuáles son los patrones de comportamiento de los clientes para establecer sus sistemas de *scoring*[31]. Las aseguradoras estudiar los patrones para prevenir el fraude. Las autoridades sanitarias ver los comportamientos de determinadas enfermedades para intentar prevenirlas. Y así un largo etcétera de necesidades públicas, industriales, financieras, sanitarias y personales de análisis de información para predicción, prevención, toma de decisiones, etc.

Actualmente, existen muchos clúster implementados con Hadoop para usos muy diversos. Concretamente, Yahoo dispone de más de 40.000 nodos ejecutando Hadoop (4.500 de ellos en un único clúster) y ya existe algún despliegue con Spark de más de 8.000 nodos. Cuando hablamos de Big Data debemos pensar **"a lo grande"**. No tiene ningún sentido realizar un despliegue de Big Data para resolver problemas *pequeños* o para almacenar ficheros pequeños, que cualquier ordenador de sobremesa es capaz de resolver o almacenar, aunque sea con dificultad.

El verdadero sentido del Big Data es la solución de problemas (de cálculo o de almacenamiento) que son inabordables para los sistemas informáticos habituales. Claramente, pocos ordenadores son capaces de resolver en un tiempo razonable lo que pueden resolver 50, 500, 2.000 o 8.000 nodos de un clúster.

Hadoop y Spark han demostrado ser sistemas eficaces y estables a la vez que son sencillos de instalar, configurar y mantener. Al ser sistemas basados en software libre, el producto en sí no tiene coste, y podemos dedicar todos nuestros recursos y esfuerzos a la infraestructura hardware que lo soportará y a los recursos humanos que tendrán que gestionar y administrar nuestro clúster y, por supuesto, a nuestro proyecto de Big Data. Además, nuestra infraestructura de hardware no necesita de grandes y caros sistemas para poder funcionar, sino que podemos ejecutarlo sobre sistemas estándar.

Estas características convierten a Hadoop y Spark y toda la infraestructura hardware en meras herramientas (que, al fin y al cabo, es lo que son) de trabajo para poder trabajar y desarrollar plenamente nuestro verdadero objetivo, el análisis y almacenamiento de información. Esto pasa

[31] Scoring: Credit Rating. Calificación de crédito de una persona o entidad.

a convertir el *Big Data* no en un fin, sino en un medio para conseguir el verdadero fin que vendrá determinado por el tipo de organización en la que nos encontremos.

Estos sistemas trabajando en colaboración con otros tipos de middleware que no han sido analizados en este TFG como Spark SQL, Hive, Pig, Streaming y una innumerable cantidad de productos software hacen del Big Data una potentísima herramienta de futuro.

Para mí, personalmente, ha supuesto un gran descubrimiento el comprobar como un sistema de estas características que, actualmente utilizan muchas grandes compañías en clúster casi inimaginables, se puede replicar a pequeña escala de una forma razonablemente sencilla y rápida. Cómo el mismo sistema que gestiona 30 máquinas en un laboratorio de la ETSINF, puede gestionar 42.000 máquinas en uno o varios data center de Yahoo.

Ha sido una magnifica experiencia al adentrarme en un terreno desconocido para mí y poco documentado en general, donde la información es encontrada con cuentagotas y los problemas y errores tienen que ser investigados uno por uno sin mucha documentación específica para consultarlos.

Esto da sentido a algunas asignaturas estudiadas durante el **Grado en Ingeniería Informática** y a algunos de los conceptos vistos como escalabilidad, sistemas distribuidos, computación paralela, tolerancia a fallos, portabilidad, etc.

Respecto a trabajos futuros relacionados con el tema de este TFG, creo que el siguiente paso sería adentrarse en dos terrenos muy diferentes, por un lado, optimizar todas las diferentes opciones de configuración que nos ofrece nuestro clúster tanto para Hadoop, como para Yarn y Spark. Puede ser muy interesante observar cómo evoluciona el rendimiento modificando algunos parámetros. Por otro lado, la secuencia lógica seria adentrarnos en la programación de Spark y en sus API´s para poder conseguir sacar el máximo rendimiento y las mejores prestaciones para nuestras aplicaciones distribuidas.

Índice de tablas

Índice de imágenes

Bibliografía

1. **Hawking, Stephen.** *El Universo en una cáscara de nuez.* Barcelona : Editorial Critica, 2002. ISBN: 84-8432-293-9 .

2. **Asociación Profesional Española de Privacidad.** APEP. [En línea] 2015. [Citado el: 18 de Diciembre de 2015.] http://www.apep.es/nuevos-negocios-y-aplicaciones-big-data-y-la-evolucion-del-internet-de-las-cosas/.

3. **Computer Sciences Corporation.** CSC. [En línea] 2012. [Citado el: 18 de Diciembre de 2015.] http://assets1.csc.com/insights/downloads/CSC_Infographic_Big_Data.pdf.

4. **IBM.** [En línea] 2015. [Citado el: 20 de Diciembre de 2015.] http://www-03.ibm.com/press/us/en/pressrelease/46453.wss.

5. **SINTEF.** [En línea] 2013. [Citado el: 20 de Diciembre de 2015.] http://www.sciencedaily.com/releases/2013/05/130522085217.htm.

6. **Hide, Winston.** HARVARD, School of Public Health. [En línea] 2012. [Citado el: 20 de Diciembre de 2015.] http://www.hsph.harvard.edu/news/magazine/spr12-big-data-tb-health-costs/.

7. **Rogers, Shawn.** Information Management. [En línea] 2011. [Citado el: 23 de Diciembre de 2015.] http://www.information-management.com/issues/21_5/big-data-is-scaling-bi-and-analytics-10021093-1.html?zkPrintable=1&nopagination=1.

8. **Linux Mint.** Linux Mint Web. [En línea] 2015. [Citado el: 26 de Diciembre de 2015.] http://www.linuxmint.com/index.php.

9. **Apache Hadoop Project.** [En línea] 2015. [Citado el: 27 de Diciembre de 2015.] https://hadoop.apache.org/.

10. **HDFS Design.** [En línea] 2015. [Citado el: 27 de Diciembre de 2015.] http://hadoop.apache.org/docs/r2.6.2/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html.

11. **Hadoop MapReduce Tutorial.** [En línea] 2015. [Citado el: 28 de Diciembre de 2015.] http://hadoop.apache.org/docs/r2.6.2/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

12. **Jeffrey Dean, Sanjay Ghemawat.** *MapReduce: simplified data processing on large clusters.* s.l. : Communications of the ACM - 50th anniversary issue: 1958 - 2008, 2008.

13. **Tech Target Network.** [En línea] 2015. [Citado el: 18 de Enero de 2016.] http://searchdatamanagement.techtarget.com/essentialguide/Managing-Hadoop-projects-What-you-need-to-know-to-succeed.

14. **Apache Spark Project.** [En línea] 2015. [Citado el: 28 de Diciembre de 2015.] http://spark.apache.org/docs/latest/.

15. **Cluster Mode Overview.** Apache Spark Org. [En línea] 2015. [Citado el: 28 de Enero de 2016.] http://spark.apache.org/docs/latest/cluster-overview.html.

16. **Apache Hadoop Cluster Setup.** Cluster Setup. [En línea] 2015. [Citado el: 24 de Febrero de 2016.] http://hadoop.apache.org/docs/r2.6.4/hadoop-project-dist/hadoop-common/ClusterSetup.html.

17. **Apache Spark.** Spark Documentation. [En línea] 2015. [Citado el: 27 de Febrero de 2016.] http://spark.apache.org/docs/1.5.2/running-on-yarn.html.

18. **Apache Hadoop.** File System Shell. [En línea] [Citado el: 29 de Febrero de 2016.] http://hadoop.apache.org/docs/r2.6.2/hadoop-project-dist/hadoop-common/FileSystemShell.html.

19. **BBVA.** BBVA Open 4 U. [En línea] 2015. [Citado el: 3 de Marzo de 2016.] http://www.bbvaopen4u.com/es/actualidad/apache-spark-las-ventajas-de-usar-al-nuevo-rey-de-big-data.

20. **Apache Spark.** Submitting Applications. [En línea] 2015. [Citado el: 5 de Marzo de 2016.] http://spark.apache.org/docs/latest/submitting-applications.html.

21. **Riza, Sandy.** Cloudera Engineering Blog. [En línea] 2014. [Citado el: 16 de Marzo de 2016.] http://blog.cloudera.com/blog/2014/05/apache-spark-resource-management-and-yarn-app-models/.

22. **Apache Spark .** Spark Programming Guide. [En línea] 2015. [Citado el: 10 de Marzo de 2016.] http://spark.apache.org/docs/1.5.2/programming-guide.html.

23. **Lublinsky, Boris, Smith, Kevin T. y Yakubovich, Alexey.** *Hadoop Soluciones Big Data.* Madrid : Ediciones Anaya Multimedia, 2014. ISBN 978-84-415-3591-6.