UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

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: Ernesto Peralta Macías

Tutor: Jon Ander Gómez Adrian

2015-2016

Resumen

En este TFG describiremos el concepto de *big data* y las arquitectura y componentes de las herramientas *software* Apache *Hadoop* y Apache *Spark*. Instalaremos y configuraremos *Hadoop* y *Spark* en un *cluster* de 32 ordenadores en un áula física y en un *cluster* con 7 máquinas virtuales. Una vez instalado, ejecutaremos dos algoritmos en ellos.

Palabras clave: *Big data*, *Hadoop*, *Spark*, *MapReduce*, Virtualización, Sistemas Distribuidos.

Abstract

In this TFG we describe the concept of *big data* and the architecture and components of Apache *Hadoop* and Apache *Spark* software tools. We install and configure *Hadoop* and *Spark* in a *cluster* of 32 computers in a physical classroom and in a *cluster* with 7 virtual machines. Once installed, we run two algorithms on them.

Keywords : *Big data*, *Hadoop*, *Spark*, *MapReduce*, Virtualization, Distributed systems.

Tabla de contenidos

1. Introducción

Este TFG "Instalación y configuración de herramientas software para *Big data*" es el principio de una andadura personal en un nuevo campo en la informática llamado *big data* que consiste en el almacenamiento y tratamiento de datos a gran e*scala*.

Big data está de moda. Salen con frecuencia noticias en prensa y televisión alabando sus logros. Por ejemplo, "*El 'big data' da la victoria al Real Madrid en la Champions... ¡por los pelos!* "(27/05/2016- El confidencial). Y en efecto, así fue. Ganó al día siguiente en la tanda de penaltis, por los pelos. O por ejemplo las predicciones tan acertadas que llevan unos años haciendo la compañía *Farsite Forecast* en las 6 categorías principales.

Pero no solo nos sirve *big data* para realizar predicciones, también es muy útil para la toma de decisiones, por ejemplo "*El baloncesto español se abre al Big data, la revolución que viene de América*" (18/10/205- Sabemos digital) donde explica que la NBA tiene la recopilación, ordenación y sintetización de miles, millones de datos relacionados con el juego y los jugadores y obtienen estadísticas avanzadas y análisis numéricos con más de 4.5 cuatrillones ($4,5*10^{15}$) combinaciones estadísticas posible.

Es decir, los entrenadores toman decisiones en tiempo real ante lo que ve en esas estadísticas y el resultado es que los equipos han cambiado acertadamente sus estrategias de juego por ello.

En *big data* tenemos una doble vertiente, por un lado tenemos el soporte que nos da la posibilidad de almacenar masivamente datos, en nuestro caso *Hadoop*, y por otro la posibilidad de tratarlos, que en nuestro caso será *Spark*.

La finalidad del TFG será la instalación y configuración de ambas herramientas software y de esa forma crear un *cluster*[1] *Hadoop* + *Spark*, sobre el que ejecutaremos 2 aplicaciones.

Previamente, habremos explicado qué es *big data*, cómo ha surgido, cuáles son sus características y las numerosas tecnologías que la componen y en qué lugar están *Hadoop* y *Spark*. Una vez contextualizadas en el ecosistema *big data*, describiremos qué necesidades cubren y sobre todo cuáles son su arquitectura y componentes.

La instalación la haremos sobre un aula de la ETSINF, el aula Anita Borg, con 32 ordenadores, sobre los que previamente habremos realizados un conjunto de averiguaciones y acciones.

Adicionalmente, crearemos un *cluster* de 7 máquinas virtuales[2] en un ordenador personal para poder realizar pruebas de presión. De esa forma, si las pruebas inutilizaran el *cluster*, sería muy rápido levantarlo de cero otra vez y seguir haciendo pruebas.

Por último probaremos 2 algoritmos en nuestro *cluster*.

[1] conjuntos de ordenadores unidos entre sí por una red y que se comportan como si fuesen una única computadora.

[2] software que simula a una ordenador y puede ejecutar programas como si fuese una computadora real

1.1.-Motivación

Desde que Dª María José Ramírez Quintana, profesora del DSIC de la ETSINF de la UPV, que impartió clase de LTP (Lenguajes, Tecnologías y Paradigmas de Programación) nos explicó, en la última clase, la existencia de *MapReduce* y *Hadoop*, de sus expectativas y del interés creciente de las empresas en este tema, me puse a leer sobre ello y decidí enfocar mi carrera profesional en esta nueva etapa hacia ese campo. De hecho, en estos momentos, estoy cursando el Máster Big Data Analytics, que dirige Dº Jon Ander Gómez Adrián.

Grandes bancos, aseguradoras, compañías de electricidad y telefonía móvil ya han incorporado equipos de trabajo que hacen uso de la tecnología *Big data*. Las Pymes, por ahora reticentes, andan más a la expectativa y esperan a que haya una infraestructura más preparada, con menor coste y que dé beneficios más inmediatos; como se suele decir, que se democratice. No obstante, si quieren sobrevivir, acabarán uniéndose a esta filosofía que les permitirá conectar con el cliente de forma más directa y saber de primera mano sus necesidades.

Por tanto, se acerca una época de nuevas oportunidades laborales para la que hay que estar preparado con nuevos e ilusionantes desafíos profesionales por realizar.

1.2.-Objetivos

•Descripción de qué es *Big data* y contextualizar *Hadoop* y *Spark*.

•Entender las tecnologías de *Hadoop* y *Spark* con la descripción de sus arquitecturas y componentes.

•Saber instalar y configurar un *cluster Hadoop* + *Spark*.

•Conocer más opciones que las mínimas para la configuración de un *cluster Hadoop* + *Spark*.

•Instalar y configurar un *cluster Hadoop* + *Spark* en un aula física con 32 ordenadores.

•Instalar y configurar un *cluster* virtual *Hadoop* + *Spark* con 7 máquinas virtuales.

•Realización de una guía de instalación y configuración del *cluster*.

•Probar 2 algoritmos en los distintos *cluster*.

1.3.- Estructura de la memoria

En el capítulo 2 describiremos el marco donde nos movemos llamado *big data*, y contextualizaremos nuestras herramientas *Hadoop* y *Spark* dentro del ecosistema de tecnologías *big data* que hay en el mercado.

En el capítulo 3 conoceremos las características de las herramientas *Hadoop* y *Spark*, sus arquitecturas y componentes.

En el capítulo 4 veremos cuáles son los requisitos para poder instalar un *cluster Hadoop* y *Spark* para posteriormente explicar cómo se realiza su instalación y configuración.

En el capítulo 5 resumiremos en una guía el proceso de creación de un *cluster Hadoop* y *Spark* y en el capítulo 6 ejecutaremos 2 algoritmos en nuestro *cluster*.

1.4.- Notas bibliográficas

Para alcanzar los objetivos descritos he leído y estudiado documentación que en función de los temas han sido:

•Para *Hadoop* y *Spark*: los conceptos principales los he obtenido de las páginas oficial de ambos y los detalles de:

-*Hadoop*, en " *Hadoop: the definitive guide* " de Tom White y otros, editorial o'Reilly.

-*YARN*, en dos libros:

"*Learning Yarn*" de Akhil Arora y otros, editorial Packt Publishing.

"*Apache Hadoop™ YARN*" de Arun C. Murthy y otros, editorial o'Reilly.

-*Spark*, en: "*Learning Spark*" de H*Old*en Karau y otros, editorial o'Reilly.

•*big data*:

-apuntes de Dº Rubén Casado y Dº José Hernández-Orallo del *Master Big data* Analytics.

-artículos derivados de *datasciencecentral.com*.

•Conceptos generales: fundamentalmente *wikipedia.com* y *"Big data Glossary",* de Pete Warden, editorial o'Reilly y conceptos estudiados en más profundidad: distintos tutoriales sobre JRE, *MapReduce*, SSH.

2. Qué es *big data*.

> "*Everybody talks about it, nobody really knows how to do it, everyone thinks everyone else is doing it so everyone claims they are doing it*". Amir Kassaei (DDB) sobre *Big data* en mumbrella360, Junio 2014.

Hace 35 años, en 1981, IBM presentó el primer PC (*personal computer*), su modelo 5150, con un procesador de 4,77 MHz[3], una memoria RAM de 16 KB y con un disco duro externo de 10MB que inundó oficinas y hogares [1]. Y hace 26 años, en 1990, se crea el primer cliente *Web*, llamado WorldWideWeb (*WWW*), y el primer servidor *web*. En 2004 aparece el concepto de *WEB* 2.0[4]. En 2006, el número de ordenadores fue cercano a los 1.000 millones [2] e Internet alcanzó los 3.500 millones de usuarios en el 2016 [3].

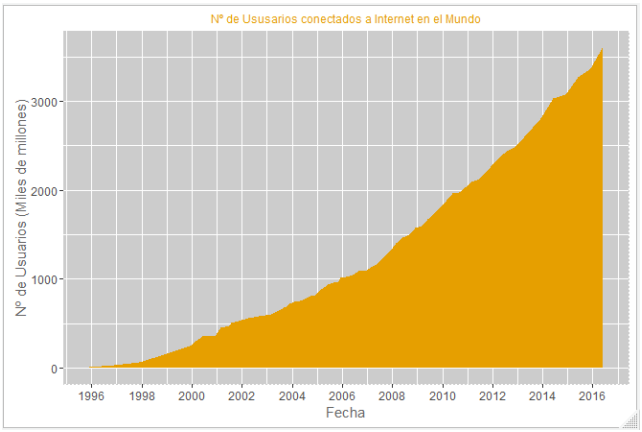


Gráfico 1. Crecimiento de usuarios conectados a Internet. Fuente datos:[2]

Adicionalmente han ido apareciendo dispositivos nuevos como teléfonos móviles, tabletas, relojes inteligentes, videoconsolas, reproductores digitales, cámaras de fotos y vídeo, tarjetas inteligentes, sensores de todo tipo, etc.

La sociedad, en todos sus sectores, ha asimilado la informática y sus dispositivos de una forma tan natural que en pocos años ha pasado a formar parte natural e indispensable de nuestra vida diaria.

De esta nueva era digital que tantos cambios ha producido y va a producir de forma revolucionaria en nuestro día a día, nos centraremos en una de sus consecuencias más evidentes: genera datos. Muchos datos. Muchísimos datos.

[3] Un **megahercio** es una unidad de medida de la frecuencia; equivale a 10^6 hercios
[4] o Web Social comprende aquellos sitios web que facilitan el compartir información, la interoperabilidad, el diseño centrado en el usuario y la colaboración en la WWW.

Datos, datos, datos

Está comenzando una nueva era. La era de los datos masivos. Los datos están creciendo tan rápido y tan vorazmente que no se pueden ignorar.

Para saber de cuánta cantidad de datos hablamos, recordemos las unidades de capacidad en la Tabla 1, porque las utilizaremos ahora en distintas comparaciones.

Por otro lado, para tener una idea intuitiva de la capacidad de 1 EB almacenar 36.000 años de vídeo de calidad HD [4] y, por tanto, 1 ZB almacenaría 36.8 millones de años.

Unidades de espacio			
Sistema Binario		Sistema Decimal	
Unidad	Nº Bytes	Nombre Unidad	Nº Bytes
1 B	1B		1
1 KB	2^{10} B	Kilo	$\approx 10^{3}$
1 MB	2^{20} B	Mega	$\approx 10^{6}$
1 GB	2^{30} B	Giga	$\approx 10^{9}$
1 TB	2^{40} B	Tera	$\approx 10^{12}$
1 PT	2^{50} B	Peta	$\approx 10^{15}$
1 EB	2^{60} B	Exa	$\approx 10^{18}$
1 ZB	2^{70} B	Zetta	$\approx 10^{21}$
1 YB	2^{80} B	Yotta	$\approx 10^{24}$

Tabla 1. Unidades de capacidad.

Veamos algunas estadísticas para hacernos una composición de lugar [5]:

• Han sido creado más datos en los dos últimos años que en la historia previa de la raza humana.

• Se prevé que en el año 2020, cada persona en el planeta, generará cerca de 1.7 MB de información nueva, cada segundo.

•Para entonces, nuestro universo digital de datos acumulados habrá crecido de 4.4 ZB de hoy día a 44 ZB.

•En 2020, se prevén sobre 6.100 millones de usuarios globales de teléfonos móviles.

• En los próximos cinco años habrán sobre 50.000 millones de dispositivos inteligentes conectados en el mundo desarrollados para almacenar, analizar y compartir datos.

Como curiosidad, en domo.com, han obtenido unas estadísticas basada en la cantidad de eventos que se generan en determinadas webs cada minuto [6].

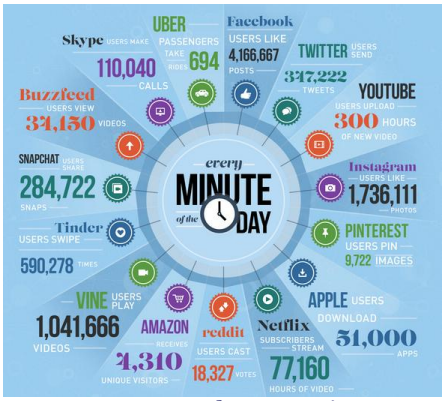


Imagen 1. Eventos generados por minuto. Fuente: [6].

¿Qué hacemos con los datos? ¿Big data?

Muchas empresas todavía están luchando para entender de lo que trata *Big data* y si les es necesario. Mucho de esto tiene que ver con que [7]:

•es una tecnología nueva que requiere inversión en formación e infraestructura,

•la gran complejidad del análisis de datos y sus condiciones de privacidad y seguridad,

•la escasez de los casos de éxito donde referenciarse, debido que proporcionan un valor diferencial que las empresas que lo realizan no están lógicamente dispuestas a compartir.

En *mumbrella360,* conferencias anuales celebradas en Australia, diseñadas para el negocio de los medios de comunicación, el marketing y el entretenimiento, se habló en Junio del 2014 de las posibilidades en torno a los datos, pero también hubo alguna discusión en torno al bombo publicitario del concepto *big data*.

Amir Kassaei, global chief creative officer de DDB, en su intervención, no intentó rebajar las posibilidades que ofrecían los datos, pero al igual que otros muchos, cree que *Big data* es otra palabra de moda y la industria no esta todavía preparada para aprovecharla de forma efectiva. Su opinión es que hay que dar un paso atrás para entender mejor la forma de aprovechar los datos para crear significado y hacer marcas más relevantes para los consumidores. Amir, en su contextualización, comparaba *big data* con el sexo adolescente: "*Todo el mundo habla de ello, nadie sabe cómo hacerlo, todo el mundo piensa que todos los demás lo hacen, por lo que todo el mundo dice que lo está haciendo*"[8].

Y esta situación en que se escucha el escopetazo de salida, y todos los corredores en vez de salir corriendo, están mirándose los unos a los otros, es así porque menos del 0.5% del total de datos es alguna vez analizada y utilizada [5].

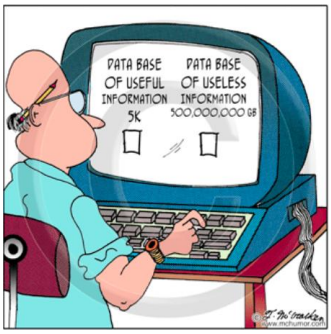


Imagen 2. Fuente: McHumor.com by T.McCracken

Lo que es evidente es que la tecnología permite que los clientes estén emitiendo permanentemente información a través de sus dispositivos y que esos datos, al alcance de las empresas, pueden ser aprovechados desde el punto de vista del marketing o de la segmentación de clientes.

En el mundo de la economía las pautas las marcaban las empresas y ahora las marcarán las personas. Quien sepa escucharlas y ofrecerles lo que quieren serán los que sobrevivan [9].

Las empresas tendrán que cambiar su centro de gravedad, digitalizarse e hiperconectarse, para transformar su relación con el cliente y diseñar nuevos servicios.

Big data

En general, un problema se puede catalogar como *Big data* si involucra un conjunto de datos suficientemente grande como para que no pueda ser procesado en una sola máquina.

Aunque no debemos perder la perspectiva, nuestro objetivo es obtener conocimiento a partir de los datos, que nos permitan tomar decisiones, incluso en tiempo real.

Muchas empresas llevan muchos años gestionando importantes volúmenes de datos con DataWarehouses[5] y obteniendo conocimiento de ellos con herramientas analíticas. No es nada nuevo. ¿Cuál es entonces la diferencia entre las aplicaciones, estas herramientas y los nuevos conceptos de *Big data*? ¡ Las 'Vs' del *Big data !*

La 'Vs' de *Big data*[10]

Características que nos permite definir a nuestro problema como big data.

⇨*Big data 1.0: Las 3 V's (focalizado en la capacidad)*

•*Volumen*: tamaño del conjunto de datos.

•*Velocidad*: Latencia en el procesado de los datos para alta interactividad.

•*Variedad*: Diversidad de fuentes, formatos , calidad , estructuras , etc.

⇨*Big data 2.0: The 4-5 V's (focalizado en la cualidad)*

•*Veracidad*: la calidad del dato, su confiabilidad.

•*Variabilidad*: datos cuyo significado está en constante cambio.

⇨*Big data 3.0: Más V's. (focalizado en la aplicabilidad)*

•*Valor*: importancia del dato para el negocio.

•*Volatibilidad*: cuánto tiempo el dato es válido.

•*Versatibilidad*: como de útil es el dato en diferentes escenarios.

•*Visualización*: imprescindible para explicar resultados complejos de forma sencilla.

Nosotros nos centraremos en 4 v's: Volumen, Velocidad, Variedad y Valor.

Retos y soluciones[11]

"Big data son activos de información de alto volumen, alta velocidad y/o alta variedad que precisa de nuevas formas de procesamiento para permitir la toma de decisiones mejorada, descubrimiento de conocimiento y optimización de procesos."- Gartner IT Glossary.

En los problemas de *Big data* surgen los retos y de ellos las soluciones.

[5] Almacén de datos, que no debe usarse con datos de uso actual, integradol y variable en el tiempo, diseñada para favorecer el análisis y la divulgación eficiente de datos que ayuda a la toma de decisiones en la entidad en la que se usa.

•*Batch processing*: forma eficiente de procesar grandes volúmenes de datos

•*Streaming*[6] *processing*: paradigma de programación que requiere una entrada continua de datos, un proceso y una salida de datos que debe ser procesado en un pequeño período de tiempo, en tiempo real.

•*NoSql* (Not Only SQL): Familia de soluciones de bases de datos "no relacionales, distribuidas y open-source" para tratar datos estructurados, no estructurados, ficheros, imágenes, sonidos, vídeos, etc

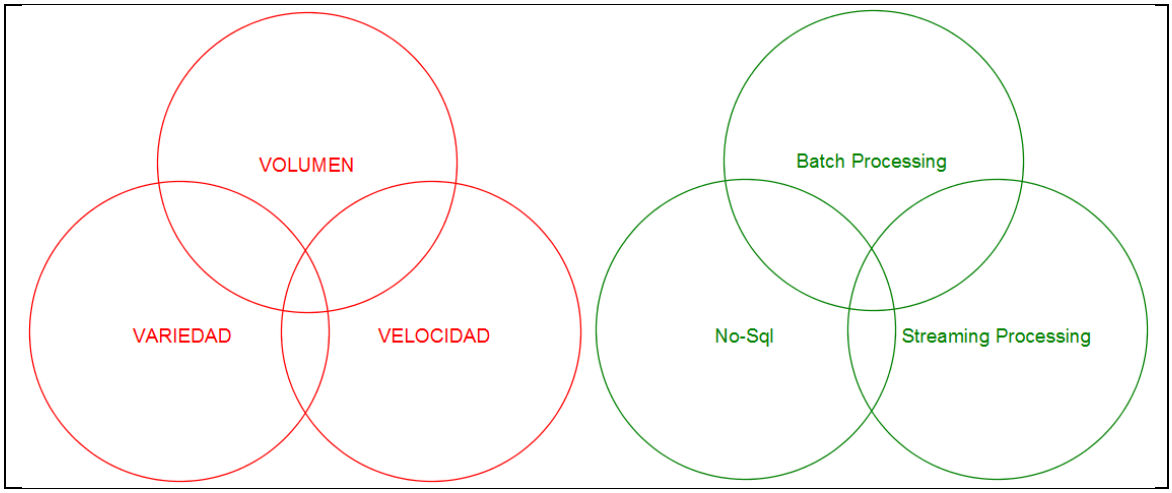


Gráfico 2. Retos y soluciones a los problemas de Big data. Fuente: [11].

Ante los retos que nos presenta Big data, la ***ingeniería del software*** nos permite ofrecer soluciones para abordar la cantidad de datos, su variedad y/o el procesamiento en tiempos asumibles; y la ***data science***[7] nos permitirá aplicar modelos, ecuaciones, algoritmos, así como evaluación e interpretación de resultados para obtener el conocimiento de los datos.

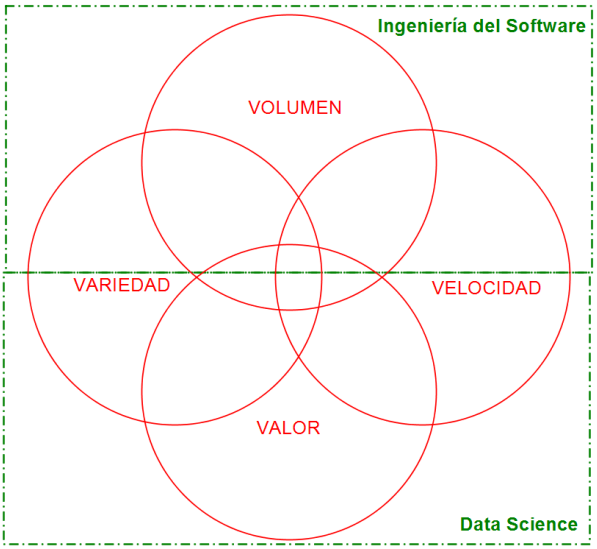


Gráfico 3. Quién soluciona qué. Fuente [11].

[6] corriente continua que fluye sin interrupción.
[7] ciencia de los datos

Tecnologías de *big data* [11]

Por un lado tenemos el ***flujo de procesamiento*** desde que localizamos los datos hasta que obtenemos un resultado.

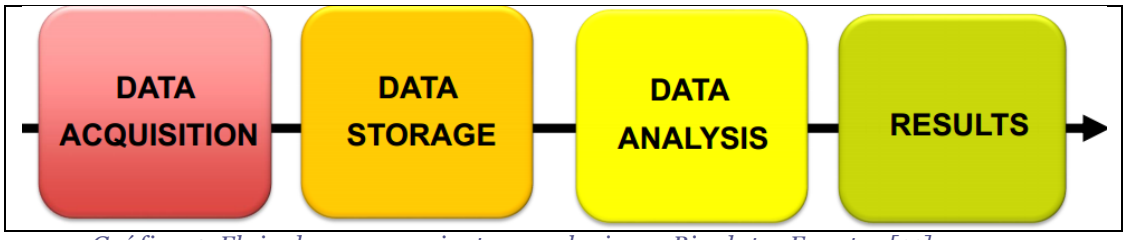


Gráfico 4. Flujo de procesamiento en soluciones Big data. Fuente: [11].

•*data acquisition:* Son nuestros ficheros de entrada, *datasets*, bases de datos, etc

•*data storage:* lo traemos a nuestro *cluster*.

•*data analisis:* hacemos los tratamientos adecuados.

•*results*: mostramos los resultados.

Por otro lado tenemos nuestros ***paradigmas de procesamiento***.

•*batch processing:* nos permite aplicar e*scala*bilidad a datos estáticos con mucho volumen .

•*streaming processing:* Adecuado para datos en continuo con resultados en tiempo real que nos da velocidad de procesamiento.

•*hybrid computation:* unifica las anteriores y con las arquitecturas Lambda[8] y Kappa[9] nos permite trabajar con volumen y velocidad.

La siguiente Tabla nos permite ayudar a buscar la tecnología adecuada a usar.

Flujo/ Paradigmas	data acquisition	data storage	data analisis
batch processing	Comandos *HDFS* Sqoop Flume Scribe	*HDFS* HBase	*MapReduce* Hive Pig Cascading *Spark* *Spark*SQL
streaming processing	Flume	Kafka Kestrel RabbitMQ	Flume Storm Trident S4 *Spark* Streaming Samza
hybrid computation			Summingbird Lamdoop Flink

Tabla 2. Tecnologías a utilizar en función de flujo y los paradigmas de procesamiento.

[8] Arquitectura de procesamiento de datos diseñado para manejar cantidades masivas de datos aprovechando las ventajas del batch processing y streaming processing.
[9] Simplificación de la arquitectura pero sin la capa de processing. Todos los datos son alimentados por el sistema de streaming más rápidamente.

Tendencias en la actualidad: 2016

•*Hadoop* y *Spark* [12]:

Basado en los resultados de la encuesta que la empresa *Syncsort* realiza sobre cómo las empresas van a aprovechar el potencial de *Big data* analytics, hay tres áreas donde se focalizarán:

1.- Apache *Spark*, se concreta como una opción a desplegar en el 70% de los encuestados.

2.- Las compañías se están mudando de plataformas *warehouse* en servidores caros a servidores multi-*cores*[10] basado en *Hadoop* que incrementa su agilidad IT[11] y de negocio, su eficiencia operacional, *reduce* costes y hace los datos visibles a toda la organización; aceptando los desafíos que implica un cambio tan importante.

3.-Aprovechar las ventajas de *Hadoop* para utilizar datos de social media y de IOT, para aplicarlos en análisis predictivo y una mejor visualización del negocio.

•Streaming processing:

Está por dilucidar qué tecnología se impondrá en real-time processing.

•Computación híbrida:

Campo nuevo en el que hace falta madurar las tecnologías y en el que habrá novedades sobre los modelos de computación híbrida.

•data lake:

Se afianza este concepto como un paso más en la estrategia de analítica de una compañía que hace referencia a un gran almacén de todos los datos de la compañía, estructurados y sin estructurar, sin ningún tipo de preprocesamiento y sin ningún tipo de esquema; para ser analizados posteriormente, cuando se necesite.

[10] componente que tiene varias unidades independientes de proceso, que leen y ejecutan instrucciones de programas.
[11] Tecnologías de la Información: Aplicación digital para almacenar, obtener, transmitir y manipular datos.

3. Tecnologías: *Hadoop* y *Spark*

3.1 Apache Hadoop

El proyecto Apache™ *Hadoop*® desarrolla software de código abierto para computación distribuida, e*scala*ble y confiable.

La librería de software Apache *Hadoop* es un *framework*[12] que permite el procesamiento distribuido de grandes conjuntos de datos a través de *clusters* de computadores, construidos con *commodity hardware* [13] , usando modelos de programación simples.

El proyecto incluye los siguiente módulos[13]:

•***Hadoop Common:*** Utilidades comunes a todos los módulos.

•***Hadoop Distributed File System (HDFS™):*** Sistema de ficheros distribuido.

•***Hadoop MapReduce***: Sistema para el procesamiento de datos en paralelo.

•***Hadoop YARN:*** *Framework* para planificación de trabajos y gestión de recursos.

Veámoslo con un poco más de detalle:

3.1.1 Hadoop Common

Hadoop Core renombrado como *Hadoop* Common se compone de un conjunto de componentes e interfaces para sistemas de ficheros distribuidos y aspectos de Entrada/Salida, como, por ejemplo[14]:

•*Serialization*: proceso de convertir los objetos estructurados en un flujo de bytes para la comunicación entre procesos, como la transmisión a través de una red o para escribir en almacenamiento persistente. La deserialización es el proceso contrario.

•*Java RPC*[14]: protocolo que utiliza un nodo para ejecutar código en otro remoto utilizando la serialización como forma de enviar el mensaje.

•*Persistent data structures*: estructuras de datos permanentes utilizadas por el *NameNode*[15] desde el proceso de formateo para crear un sistema de ficheros vacío.

3.1.2 Hadoop Distributed File System (HDFS™)

Un sistema de ficheros es un componente del sistema operativo encargado de asignar espacio a los ficheros en uno o varios dispositvos de memoria para que sean almacenados y accedidos para su uso.

Cuando un conjuto de datos crece más que la capacidad de almacenamiento de un ordenador, tenemos que particionarlos en varios ordenadores.

---

[12] Marco conceptual y tecnológico con módulos concretos de software, que puede servir para el desarrollo de software
[13] Ordenadores comunes, nada sofisticados.
[14] (Remote Procedure Call) es una técnica para la comunicación entre procesos en una o más computadoras conectadas a una red.
[15] *Nodo que hace de master en el sistema de fichero hadoop, HDFS.*

Los sistemas de ficheros que gestionan el almacenamiento a través de una red de ordenadores se llaman ***Sistemas de ficheros distribuidos***.

Hadoop viene con un sistema de ficheros distribido llamado **HDFS**, que es su buque insignia, pero actualmente *Hadoop* tiene una abstracción de sistemas de ficheros de uso general para integrarse con otros sistemas de almacenamiento, como puede ser el sistema de ficheros local o S3 de Amazon.[14]

Hadoop Distributed File System (**HDFS**) es un sistema de ficheros sobre el que se ejecutan aplicaciones *Hadoop*, que por el hecho de ser distribuido se enfrenta a una serie de desafíos que en la forma de superarlos, lo define:

•*Fallo de hardware*:

Reto: Al haber muchos componentes de hardware, alta probabilidad de fallos.

Estrategia: rápida detección de fallos y recuperación automática de estos.

•*Acceso a los datos en Streaming:*
Reto: Por el tipo de aplicación, más de procesamiento en batch que realizan grandes lecturas secuenciales en streaming, que de uso interactivo con los usuarios, el énfasis está en la alta productividad del acceso a datos más que la baja latencia.

Estrategia: accesos a los datos con un rendimiento constante y elevado.

•*Grandes conjuntos de datos:*
Reto: Almacenar y procesar ficheros de terabytes(1 TB=2^{10} GB) o petabytes (1PB=2^{20} GB).

Estrategia: Difusión de los datos a través de gran cantidad de máquinas.

•*Modelo de coherencia simple:*
Supuesto: Una aplicación típica de *Hadoop*, *MapReduce* o *Web* Crawler, una vez que la fuente de los datos ha sido copiada, o generada, en *HDFS*, se realizan uno o varios análisis sobre ellos o gran parte de ellos.

Estrategia: Seguir un patrón de procesamiento "escrito una vez, leído muchas veces" simplifica el modelo de coherencia de los datos y permite un alto rendimiento en el acceso a ellos.

•*"Mover el cálculo es más barato que mover los datos":*
Reto: Hacer el cómputo más eficiente.

Estrategia: *HDFS* proporciona interface a las aplicaciones para moverse más cerca de donde están los datos, minimizando la congestión de red e incrementando la productividad total del sistema.

•*"Portabilidad a través de Hardware heterogéneo y plataforma software ":*
Reto: Facilitar la adopción de *HDFS* como plataforma para un gran conjunto de aplicaciones por su facilidad de despliegue en ellos.

Estrategia: Portabilidad tanto de software como de hardware.

Arquitectura HDFS: NameNode y DataNode

Los ficheros se dividen en bloques de tamaño fijo, *blocks*, que se almacenan en máquinas del *cluster* elegidas al azar. Esta división permite almacenar ficheros de mayor tamaño que un disco duro pueda almacenar.

Ya que distintas máquinas almacenan distintos bloques del fichero, cada uno de ellos se replica, por defecto 3 veces, en máquinas distintas, de forma que si una máquina cae tenemos el respaldo del bloque.

Veamos la arquitectura que sigue un patrón del tipo maestro/worker:

⇨un ***NameNode*** (*master*) que:

• Determina la asignación de bloques a los *DataNodes* y sabe donde están localizados todos los bloques de un fichero.

• Gestiona el repositorio de todos los metadatos[16] *HDFS* (nombre de ficheros, permisos, y la localización de cada bloque de cada fichero).

-Gestiona el espacio de nombres del sistema de ficheros. Ejecuta operaciones como abrir, cerrar y cambiar el nombre de archivos y directorios.

-Regula el acceso a los archivos de los clientes.

⇨un conjunto de ***DataNodes*** (*workers*), uno por cada nodo del *cluster*, que:

-gestionan físicamente el almacenamiento de los archivos mediante los bloques.

-sirven las peticiones de lectura y escritura del *NameNode* o de los clientes.

-crean, eliminan y replican bloques según instrucciones del *NameNode*.

- informa periódicamente al *NameNode* de la lista de bloques que están almacenando.

Ejemplo de dos ficheros A y B, de tamaño en bloques 2 y 3, almacenados en un sistema de ficheros distribuido *HDFS* con factor de replicación 2:

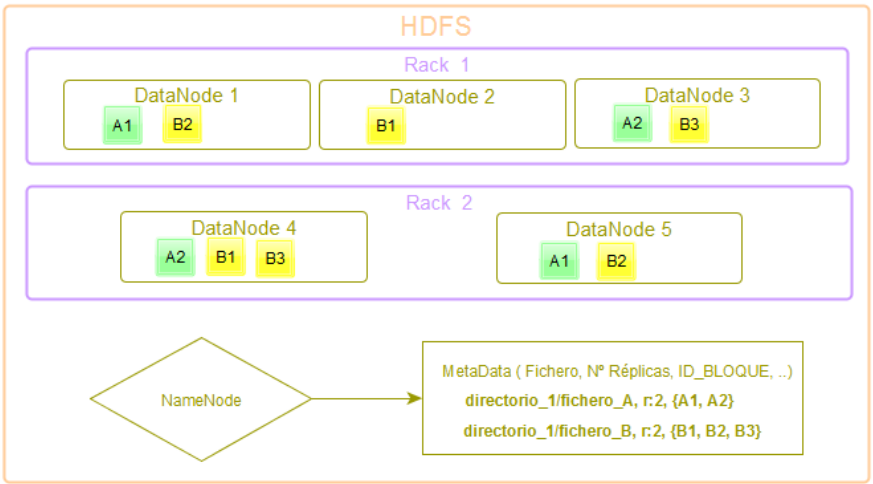


Ilustración 1. Ejemplo de 2 ficheros almacenados en HDFS con factor de repicación 2.

[16] Grupo de datos que describen el contenido informativo de un objeto al que se denomina recurso

Alta productividad de acceso a datos.

Utiliza tamaños de bloque de, por defecto, 64MB, pero pueden ser de 128MB ó 256MB, que contrasta con los 4K u 8k de otros sistemas de ficheros, y esto le permite al *NameNode*:

•menos almacenamiento de metadatos por fichero, ya que la lista de bloques es menor.

•metadatos almacenados en su memoria, permitiéndole un acceso muy rápido a ellos.

•cargar con más agilidad ficheros de principio a fin para una aplicación.

•gestión de grandes cantidades de datos de forma secuencial colocado sobre el disco, de forma que se consigue una rápida transmisión al leer los datos.

•junto con la estructuras de bloques en distintas máquinas, la ejecución de programación paralela del *MapReduce*.

Persitencia de los metadatos.

El *NameNode* guarda en memoria principal una imagen del espacio de nombres del sistema de ficheros entero y del mapa de todos los bloques de cada fichero.

Para mantener la persistencia, el *NameNode* utiliza un proceso llamado ***checkpoint*** y dos ficheros que guarda en su sistema de fichero local:

•*editLog*: registro de transacciones de cada cambio que ocurre en los metadatos.

•*fsimage*: Se almacena el espacio de nombres del sistema de ficheros entero, el mapeo de los bloques de archivos y las propiedades.

Checkpoint es un proceso que se activa cuando *NameNode* arranca y lee *fsimage* y *editLog* de disco, aplica todas las transacciones del *editLog* a la representación en memoria de *fsimage* y:

-guarda esta nueva versión de *fsimage* en disco.

-trunca a cero EditLog.

El *DataNode* no tiene conocimiento acerca de los archivos *HDFS* y almacena cada bloque de datos *HDFS* en un archivo separado en su sistema de archivos local.

Cuando un *DataNode* se pone en marcha, explora a través de su sistema de archivos local y genera una lista de todos los bloques de datos *HDFS* que corresponden a cada uno de estos archivos locales y envía este informe, llamado ***Blockreport***, al *NameNode*.

El ***NameNode secundario*** fusiona *fsimage* y *EditLog* periódicamente y mantiene este dentro de unos límites razonables de tamaño. Por lo general se ejecuta en una máquina diferente a la del *NameNode* primario.

Robustez

El objetivo principal de *HDFS* es almacenar datos de forma fiable.

Heartbeats[17]

Cada *DataNode* envía un h*eartbeats* al *NameNode* periódicamente.
En el momento que un *NameNode* detecta la ausencia de h*eartbeats* de un *DataNode*, lo marca como muerto. Esto puede hacer que el factor de replicación de algunos bloques caigan por debajo de su valor especificado y el *NameNode* inicia la replicación.

La necesidad de "*re-replicación*" puede surgir debido a que un *DataNode* puede no estar disponible momentáneamente, o una réplica se puede dañar, o un disco duro en un *DataNode* puede fallar o el factor de replicación de un archivo puede ser aumentado.

Rebalanceo del *cluster*

Por la misma dinámica del funcionamiento del sistema es posible que el espacio libre de un *DataNode* caiga por debajo de un cierto umbral. La arquitectura *HDFS* es compatible con esquemas de reequilibrio de datos y puede mover automáticamente los datos de un *DataNode* a otro si es necesario.

Existen varias políticas para asegurarse la integridad de los datos y el correcto balanceo del *cluster* como mantener una de las réplicas de los bloques en un *rack*[18] distinto al que se está escribiendo, o en el mismo *rack* pero distinto nodo o intentar mantener un tamaño en unos umbrales parecidos de espacio en el disco entre los nodos.

Integridad de los datos

Cuando un cliente crea un archivo de *HDFS*, a cada bloque del archivo se le calcula una suma de comprobación que se almacena en un archivo oculto en el mismo espacio de nombres *HDFS*.

Cuando un cliente recupera el contenido del archivo se verifica que los datos que recibe de cada *DataNode* coincide con la suma de control almacenada en el archivo de control asociado. Si no, entonces el cliente puede optar por recuperar ese bloque de otro *DataNode* que tiene una réplica de ese bloque.

Fallos de disco de los metadatos

NameNode puede ser configurado para soportar el mantenimiento de múltiples copias de *fsimage* y *editLog* de forma sincrónica, que, aunque pueda degradar la tasa de transacciones de espacio de nombres por segundo que puede soportar un *NameNode*, se asume como aceptable.

La máquina *NameNode* es un único punto de error para un *cluster* de *HDFS*. Si la máquina *NameNode* falla, la intervención manual es necesaria.

[17] servicio que permite tener conocimiento de la presencia (o desaparición) de los procesos en otras maquinas e intercambiar fácilmente mensajes entre ellos.
[18] o armarios, son soporte metálico destinado a alojar equipamiento electrónico, informático y de comunicaciones

Accesibilidad

HDFS puede ser accedidos de diferentes formas:

FS Shell

Un cliente tiene acceso al sistema de ficheros HDFS a través de una interfaz similar a la de un sistema operativo POSIX, que tiene sus propias utilidades para la gestión de archivos, muy similares o ya conocidas, como ls, cp, mv, etc.

FS Shell permite al usuario organizar los datos en la forma de ficheros y directorios y proporciona una interface de línea de comando de la que mostramos algunos ejemplos:

Unidades de tiempo	descripción
Anexar uno o varios ficheros en local al destino	*HDFS* dfs -appendToFile <localsrc> <dst>
Obtener un fichero de *HDFS*	*HDFS* dfs -get <src> <localdst>
Lista un fichero o directorio	*HDFS* dfs -ls [-R] <args>
Takes path uri's as argument and creates directories.	*HDFS* dfs -mkdir <paths>
Copia uno o varios ficheros en local al destino	*HDFS* dfs -put <localsrc> ... <dst>
Borra los ficheros especificados	*HDFS* dfs -rm URI [URI ...]

Tabla 3. Ejemplo comandos FS Shell

DFSAdmin

Intérprete de comandos para labores administrativas. Por ejemplo:

Unidades de tiempo	descripción
Pone el *cluster* en safe-mode	*HDFS* dfsadmin -safemode enter
Lista información básica y estadísticas de sistema de ficheros.	*HDFS* dfsadmin -report
Refresca *DataNodes* que se conectan al *NameNode*	*HDFS* dfsadmin -refreshNodes

Tabla 4. Ejemplo comandos DFSAdmin.

También hay comandos para trabajar directamente con el *NameNode* ("*HDFS NameNode*"), los *DataNode* ("*HDFS DataNode*") y el Secondary *NameNode* ("*HDFS* secondary*NameNode*").

API[19] *de Java*

Proporciona un *API*, *FileSystem Java API,* para que las aplicaciones trabajen directamente con *HDFS*.

Browser Interface

Una vez el *cluster* está en ejecución, podemos acceder a *HDFS* a través de su servicio *NameNode* por la *web* http://N*ameNode_host:50070 y acceder a información general del cluster HDFS, al sistema de ficheros y sus bloques o a los logs*.

[19] (*Application Programming Interface*), es el conjunto de métodos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

3.1.3 Hadoop MapReduce

MapReduce es un f*ramework* para el procesamiento paralelo de grandes volúmenes de datos en *cluster*s de forma distribuida, inspirada en la clásica estrategia de "Divide y Vencerás".

Las tareas de *MapReduce* están programadas para ejecutarse en los mismos nodos físicos en los que residen los datos, gracias al uso de *HDFS*, reduciendo el tráfico de red y manteniendo la Entrada/Salida en el disco local o en el mismo *rack*, siendo el principio básico mover los datos calculados en lugar de mover los datos a través de la red del *cluster* [15].

Arquitectura MapReduce

Veamos las fases por las que pasan los datos hasta ofrecer un resultado en el modelo de programación *MapReduce,* tomando como referencia visual la ilustración 2.

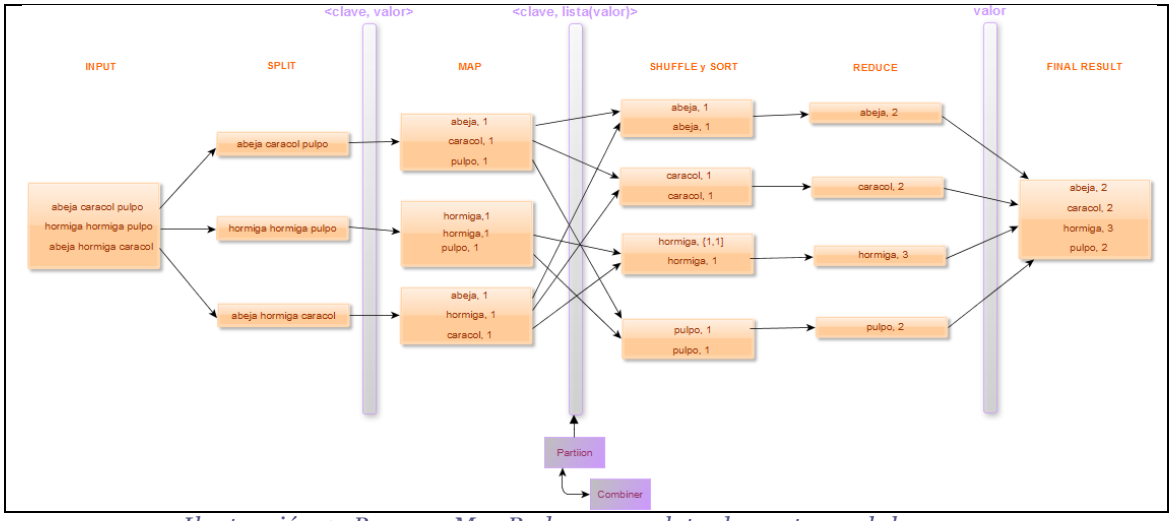


Ilustración 2. Proceso MapReduce completo de contar palabras.

•***Input***:

Si los datos residen en *HDFS*, cada nodo cargaría los bloques con los datos que tuvieren en su sistemas de ficheros *HDFS* local. Es típico de estos fichero ser de tamaños de gigabytes o más.

La clase *InputFormat* se encargaría de seleccionar ficheros u objetos que deberían ser usados en el input.

En este caso cada línea del fichero sería un registro y una salida, por ejemplo, sería: "hormiga hormiga pulpo".

•***Split***:

Un Split obtiene una unidad de trabajo, par clave/valor, que comprende una sola tarea *map* en el programa *MapReduce*.

En nuestro ejemplo, recibiría como entrada "hormiga hormiga pulpo", y la dividiría en pares <clave,valor>, es decir, <'hormiga',1><'hormiga',1><'pulpo',1>.

•***Map***:

Procesa los pares <clave,valor> y produce un conjunto de pares intermedio <clave, lista(valor)> .

En nuestro ejemplo, a partir de la entrada <'hormiga',1> <'hormiga',1> <'pulpo',1> produciría la salida: <'hormiga', [1,1]><'caracol', [1]>.

El número de tareas *Maps* depende del tamaño total de las entradas, que está en función del número de bloques del fichero. El nivel adecuado de paralelismo para tareas *Map* parece ser de 10 a 100 tareas *maps* por nodo.

•***Shuffle y Sort***:

Los resultados intermedios (pares) son agrupados y ordenados (por clave) <clave, lista(valores)>.

En nuestro caso, todas las claves, 'hormiga', por ejemplo, se agruparían: <'hormiga', [1,1]> <'hormiga', [1]>

•***Reduce***:

Los pares ordenados por clave son procesados por otro conjunto de tareas simples "*reduce*" para producir el resultado.

En nuestro caso, todas las claves 'hormiga' se reducirían: <'hormiga', 3>

El número adecuado de tareas reduce parece ser:

[0.95 ó 1.75 * (<*número de nodos* > * <número máximo *containers* por nodo >)

Con 0.95 todos las tareas *reduce* pueden lanzarse inmediatamente y empezar a transferir salidas de las tareas *map* que vayan finalizando.

Con 1.75 los nodos más ràpidos terminarán sus primeras rondas de tareas *reduce* y lanzarán una segunda oleada de tareas *reduce* mejorando el equilibrio de la carga [17].

•Analizemos de cerca las tareas de *combiner* y *partitioner*.

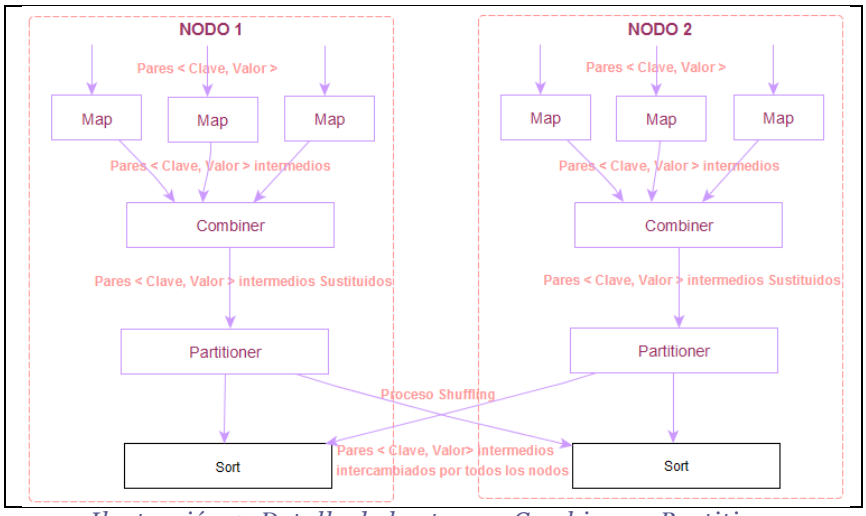


Ilustración 3. Detalle de las tareas Combiner y Partitioner.

•***combiner***:

Tarea opcional utilizada tras el mapper y antes del *reduce* que opera con datos generados en una sola máquina y cuya función es igual que una tarea "mini-*reduce*" que logra reducir la cantidad de datos que posteriormente será transferido.

Por ejemplo, si le llegaran 3 veces el par <"abeja",1>, <"abeja",[1,1,1]> la tarea *combiner* lo condensaría a <"abeja", [3]>.

•***partitioner***:

Una partición es un subconjunto donde los pares con la misma clave son asignado a un nodo *reduce* en particular.

Los pares en estas particiones son la entrada a las tareas *reduce*.

Cada tarea *map-combiner*, desde su nodo origen, enviará, en este proceso shuffling, cada par <clave, lista(valor)> al nodo en cuya partición estarán todos los de su misma clave para ser "reducidos" juntos.

¿y cómo sabe a qué partición enviarlo? A partir del valor de la clave se calcula con una función *hash*[20].

Hadoop MapReduce

MapReduce v1 - MRv1

En la estructura básica de Apache *Hadoop* 1.0 hay dos componentes, *HDFS* y *MapReduce*, que forman la base de casi toda la funcionalidad de *Hadoop*.

[20] funciones que comprimen la entrada a una salida de menor longitud, de forma que el valor de una salida solo puede obtenerse ante el mismo valor de la entrada.

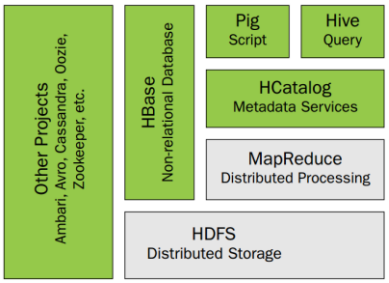


Ilustración 4. Ecosistema Hadoop 1.0. Fuente: [18].

Todos los otros componentes están construidos alrededor de estos servicios y deben usar *MapReduce* para ejecutar trabajos *Hadoop*.

El componente *MapReduce* se servía de dos servicios para la ejecución de sus aplicaciones: JobTracker y TaskTracker.

•*JobTracker*: servicio *master* responsable de la gestión de recursos, planificación de tareas y gestión.

•*TaskTracker*: servicio responsable de la ejecución de tareas *Map-Reduce* en cada nodo.

MapReduce v2 - MRv2

En la versión MRv2 aparece un cambio estructural al incorporar la arquitectura de *Hadoop* un componente nuevo, llamado ***Hadoop Yarn***, dentro del cual podremos ejecutar tanto aplicaciones *MapReduce* como otros tipos de aplicaciones distribuidas.

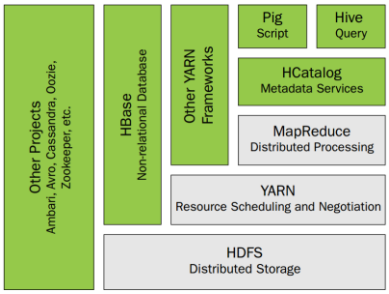


Ilustración 5. Ecosistema Hadoop 2.0. Fuente: [18].

MRv2 mantiene sus características que la identifican a nivel de usuario como las fases del proceso, pero los servicios sobre los que se ejecuta han cambiado por completo, por tanto ***Hadoop MapReduce*** se convierte en un sistema, *basado en YARN,* para el procesamiento de grandes conjuntos de datos en paralelo.

MRv2 mantiene la compatibilidad de *API* a la hora de programar aplicaciones de manera que las aplicaciones de MRv1 son funcionales.

3.1.4 Hadoop Yarn

Hadoop YARN (*Yet Another Resource Negotiator***)** es un *framework* para planificación de trabajos y recursos del *cluster*, introducido en la versión 2.0 de *Hadoop*, que proporciona más e*scala*bilidad y eficiencia.

YARN da la posibilidad de ejecutar también trabajos no *MapReduce* dentro de *Hadoop*, dando soporte para prácticamente cualquier aplicación distribuida.

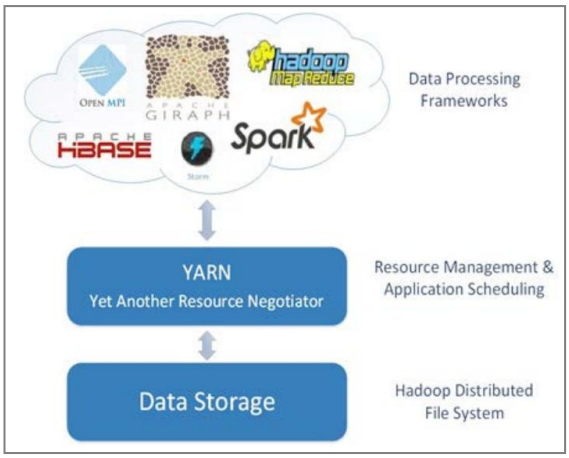


Ilustración 6. YARN permite la ejecución de otros framewors. Fuente: [18].

Arquitectura Hadoop YARN

Cuando configuramos *Hadoop YARN* en un *cluster* multi-nodo es recomendable tener separados físicamente los nodos para los servicios ***NameNode*** y ***ResourceManager***.

A medida que el número de nodos *workers* aumentan, los requerimientos de memoria, procesador y red del nodo *master* se incrementan. La siguiente ilustración muestra una vista "lógica" de alto nivel.

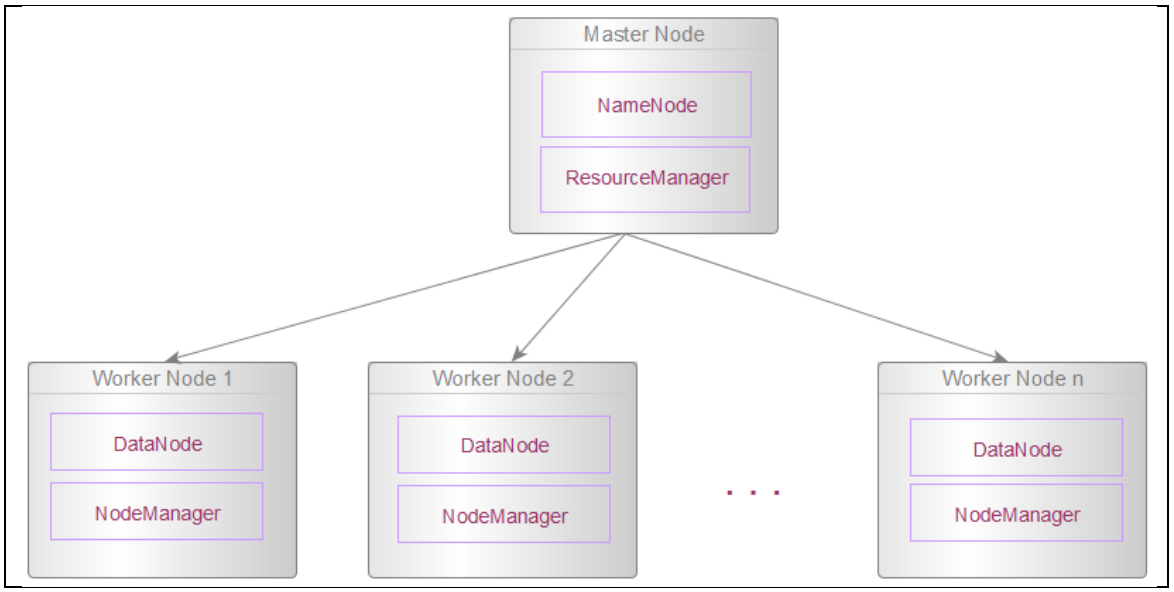


Ilustración 7. Vista de alto nivel de la arquitectura HDFS-YARN.

Los servicios con los que cuenta *YARN* en su arquitectura son[15]:

•***ResourceManager***

Servicio que gestiona la planificación de los recursos de cómputo a las aplicaciones optimizándolo en términos de memoria, *core*s de *CPU*, equidad y acuerdos a nivel de servicios (SLA's).

Para permitir diferentes políticas cuenta con *pluggable schedulers,* tales como *capacity* y *fair* que permiten configurar la asignación de recursos de forma personalizada y que veremos en la propiedad *yarn.resourcemanager.scheduler.class* en el apartado 4.3.2.3 *YARN*-site.xml con más detalle.

Se ejecuta un servicio por *cluster* y tiene 2 componentes principales:

-*Scheduler*: Asigna recursos para aplicaciones lanzadas al *cluster* aplicando límites de capacidad y colas. No hace nada más.

-*ApplicationsManager (AsM):* Servicio worker que ejecuta un servicio por nodo y envía al *ResourceManager* tanto señales *heartbeat* como informaciones de su estado y gestiona:

1.-*la applicationMasters (explicada en esta misma página)* a través del *cluster*, responsabilizándose de la creación de esta y de proporcionarles recursos para empezar, monitozrizar su progreso y reiniciar en caso de fallo de la aplicación.

2.- la ejecución de *containers* en función de su capacidad.

•***NodeManager***

Servicio worker que es responsable de la ejecución de *containers* en función de la capacidad del nodo, limitada por su memoria RAM y sus *core*s de *CPU*.

Se ejecuta un servicio por nodo y envía al *ResourceManager* tanto señales heartbeat como informaciones de su estado.

•***ApplicationMaster***

Servicio que gestiona cada instancia de una aplicación ejecutada en *YARN*. Es un servicio por cada aplicación y negocia los recursos con *ResourceManager scheduler* y trabaja con el *NodeManager* para ejecutar y monitorizar las tareas.

El *ResourceManager* asigna *containers* a la *ApplicattionMaster* y estos *containers* son usados para ejecutar los procesos específicos de la aplicación.

•***Container***

Conjunto de recursos en términos de memoria RAM, *core*s de *CPU*, memoria de disco asignado a un nodo en particular.

Para entenderlos mejor veamos cómo interactúan los componentes a través de la ejecución de una aplicación, con el soporte de la ilustración 8 [19]:

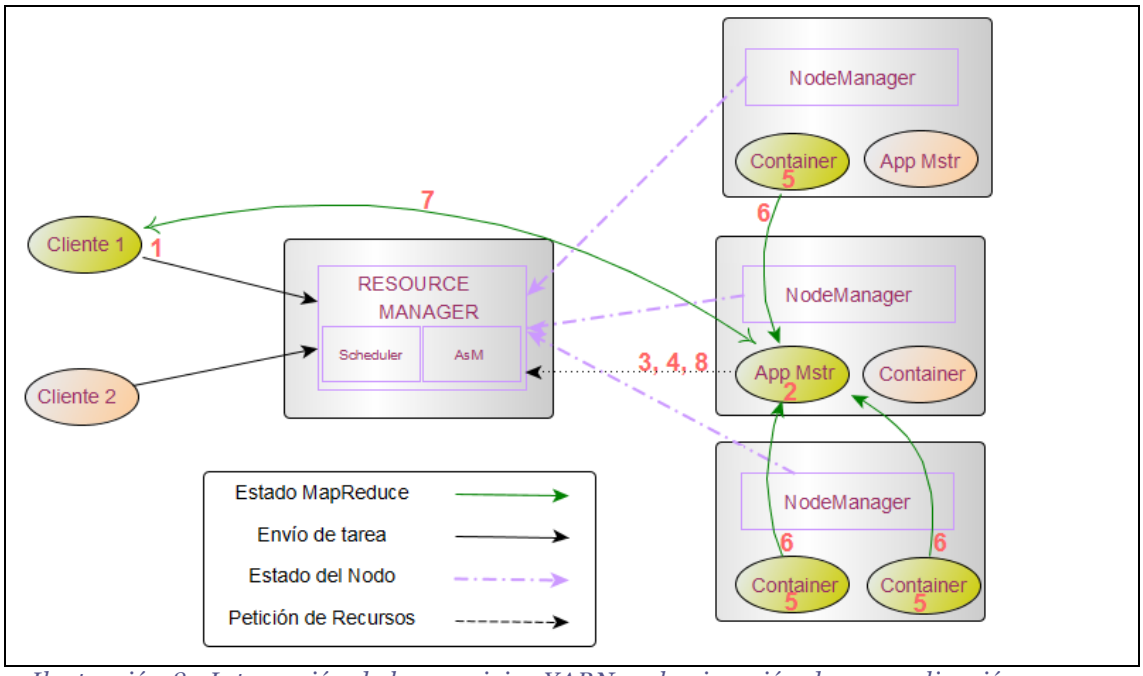


Ilustración 8. Interacción de los servicios YARN en la ejecución de una aplicación.

0.- El servicio *ResourceManager* se ejecuta en el nodo maestro del *cluster*.

1.- Un cliente *YARN* presenta una aplicación al ***ResourceManager***. El cliente también define una ***ApplicationMaster*** y un comando para lanzar la *ApplicationMaster* sobre el nodo.

2.- El servicio ***ApplicationManager*** validará y aceptará la petición de la aplicación del cliente.

El servicio ***scheduler*** asignará un *container* a ***ApplicationMaster*** en un nodo.

El servicio ***NodeManager*** en ese nodo usará el comando para lanzar el servicio ***ApplicationMaster***.

3.- La ***ApplicationMaster***, al arrancar, se registra en el ***ResourceManager.*** El registro permitirá al programa cliente preguntar al ***ResourceManager*** por detalles, lo cual le permitirá comunicar directamente con su propia ***ApplicationMaster***.

4.- La ***ApplicationMaster*** solicita recursos de ***ResourceManager***.

5.- El ***ResourceManager*** asignará los recursos como ***containers*** en un conjunto de nodos.

La ***ApplicationMaster*** conectará al servicio ***NodeManager*** y solicitará a ***NodeManager*** que lancen los ***containers***.

6.- La ***ApplicationMaster*** gestiona la ejecución de los ***containers*** y notificará al ***ResourceManager*** una vez la ejecución de la aplicación ha terminado.

El servicio ***NodeManager*** se ejecuta en cada worker del *cluster YARN*. Este es responsable de la ejecución de los ***containers*** de la aplicación. Los recursos especificados para un ***container*** son tomados de los recursos del ***NodeManager***.

Cada ***NodeManager*** periodicamente revisa en el ***ResourceManager*** para el conjunto de recursos disponibles.

El servicio ***ResourceManager scheduler*** usa la matriz de recursos para asignar nuevos ***containers*** a ***ApplicationMaster*** o para empezar la ejecución de una nueva aplicación.

7.- Durante la ejecución el cliente contacta directamente con ***ApplicationMaster*** para saber su estado, progreso, etc.

8.- Cuando la aplicación ha terminado, la ***ApplicationMaster*** libera los recursos a través del ***ResourceManager*** y termina.

3.2 Apache Spark

Apache *Spark* es un *cluster* de computación diseñado para ser rápido, fácilmente accesible y de propósito general.

La rapidez la consigue por su capacidad de ejecutar en memoria de manera distribuida, la facilidad de uso la obtiene por las *API*'s nativas de *Java*, *Scala* y Python y, por otro lado, extiende el popular modelo *MapReduce* soportando un conjunto de herramientas de alto nivel, que nos da la habilidad de construir aplicaciones combinando diferentes modelos de programación de forma natural, como [20]:

•***Spark SQL***: paquete para realizar consultas SQL sobre datos estructurados.
•***MLlib:*** librería con funcionalidad común de Machine learning
•***GraphX***: librería para manipular grafos y realizar computaciones *graph-parallel*.
• ***Spark Streaming***: componente para el procesado de flujo de datos en tiempo real.

Como capa base tiene ***Spark Core*** que contiene la funcionalidad básica de *Spark*, que incluye componentes para la planificación de tareas, gestión de memoria, recuperación de fallos, interacción con los sistemas de almacenamiento, el *API* que define los *RDD's* (*Resilient Distribuited Datasets*), y de los que hablaremos posteriormente, etc .

También *Spark* está integrado estrechamente con otras herramientas de *Big data*, por ejemplo *Spark* puede ejecutarse en un *cluster Hadoop* y acceder a fuentes de datos (*data sources*) *Hadoop* incluyendo Cassandra[11].

Todo ello, lo mostramos en la ilustración 9.

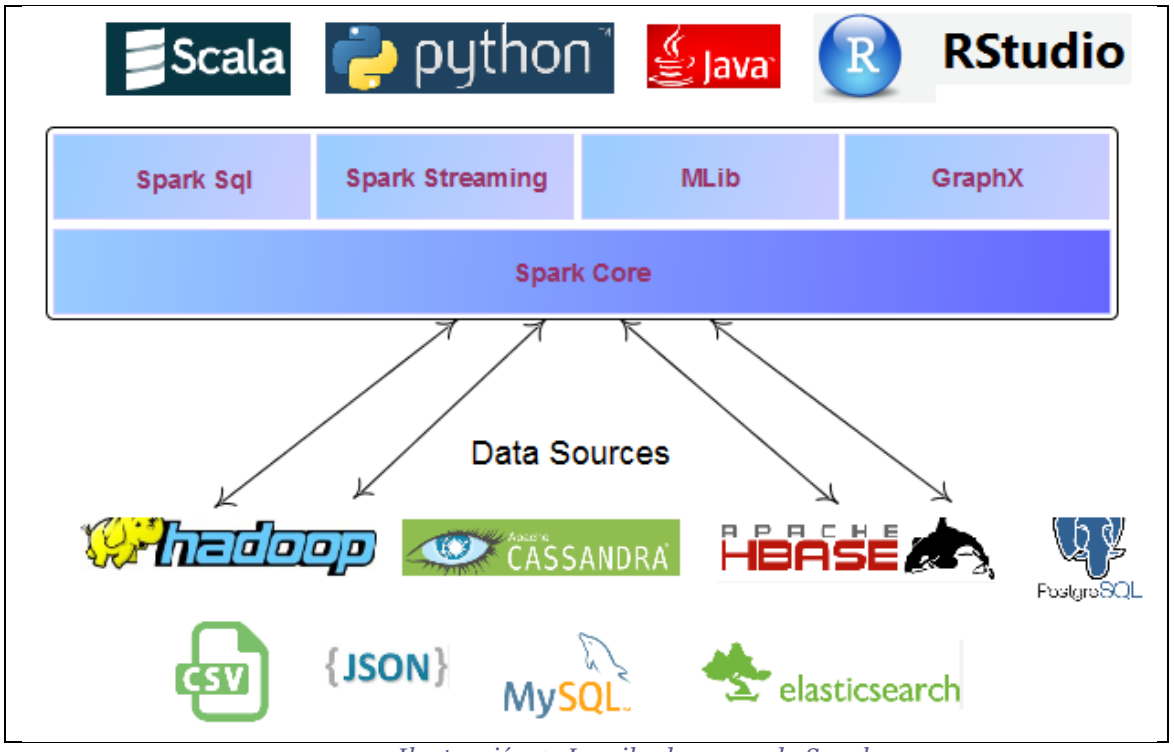


Ilustración 9. La pila de capas de Spark.

Arquitectura Spark

Veamos la arquitectura de *Spark* en modo distribuido cuyo patrón es *master/workers* con un coordinador central, llamado ***driver,*** y muchos ***workers*** distribuidos.

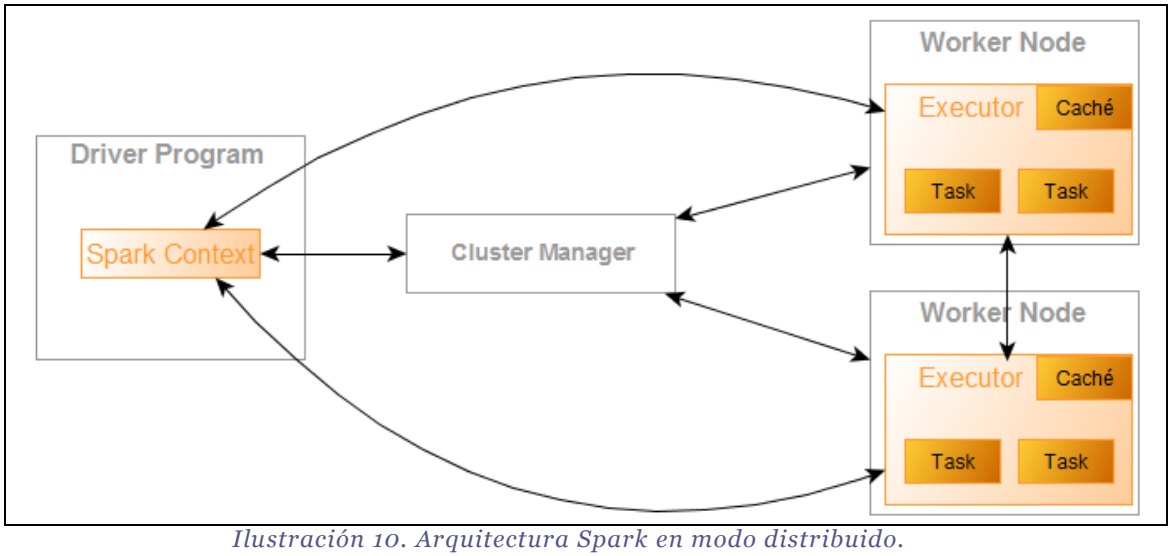


Ilustración 10. Arquitectura Spark en modo distribuido.

Las aplicaciones *Spark* ejecutan conjuntos independientes de procesos en el *cluster*, coordinados por el objeto ***SparkContext*** en tu programa principal ***driver program***.

SparkContext utiliza un servicio externo llamado ***cluster manager***, que puede elegir varios tipos de gestores de *cluster* (el propio de *Spark*, *standalone*, *Mesos*

o *YARN*), los cuales asignan recursos a través de las aplicaciones y posibilitan su conexión y ejecución en el *cluster*.

Una vez conectado al *cluster*, la ***aplicación Spark*** adquiere ***executors*** en los nodos del *cluster* a quienes se les envía el código de la aplicación y ***SparkContext*** envía tareas para que ejecuten.

Resumamos los conceptos visto e introduzcamos unos nuevos para adentrarnos en los tipos de despliegues o gestores del *cluster*:

Application: Programa de usuario construido en *Spark*.

Application jar: Un compirmido jar[21] de la aplicación *Spark*.

Driver program: Un proceso ejecutando la aplicación del usuario que contiene el objeto *SparkContext*.

Cluster manager: Un servicio externo para adquirir recursos del *cluster*.

Deploy mode: Distingue dónde el proceso *driver program* se ejecuta: *cluster* mode o client mode.

Worker node: Cualquier nodo que pueda ejecutar código de aplicación en el *cluster*.

Executor: Proceso lanzado por la aplicación en un *worker node*, que ejecuta tareas y guarda datos en memoria RAM o disco. Cada aplicación tiene sus propios *executors*.

Task: Unidad de trabajo que será enviada a un *executor*.

Job: Cómputo paralelo que consta de múltiples tareas que se generan por una acción de *Spark*, por ejemplo, collect().

Stage: Pequeños conjuntos de tareas en los que se divide un trabasjo que dependen unas de otras, similar a las etapas de *map* y *reduce* en *MapReduce*.

Tipos de gestores de cluster

Spark puede ejecutarse en una variedad de modos o entornos. Más allá del ***modo local*** que es un modo no distribuido, donde se ejecuta en una sola máquina, puede ejecutarse, de forma distribuida, en tres tipos de *cluster* managers: ***Hadoop YARN***, **Apache Mesos** y el ***Standalone***. Veamoslo brevemente:

[21] (por sus siglas en inglés, **J**ava **AR**chive) es un tipo de archivo que permite ejecutar aplicaciones escritas en el Java.

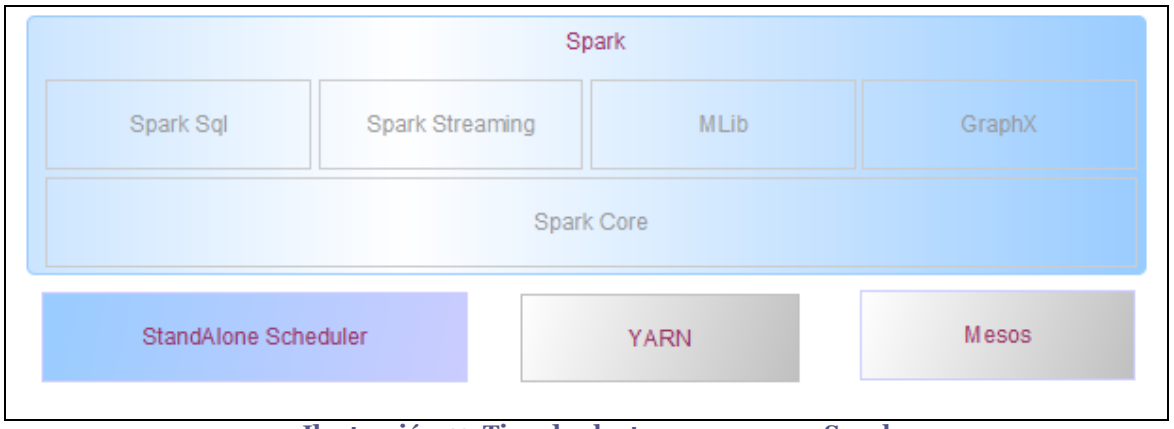


Ilustración 11. Tipo de cluster manager en Spark.

•*Standalone*:

El gestor de *cluster* Standalone de *Spark* ofrece una forma sencilla de ejecutar aplicaciones en un *cluster* con paralelismos de datos implícito, tolerancia a fallos y una política básica de planificación que permite limitar el uso de cada aplicación de forma que múltiples aplicaciones pueden ejecutarse concurrentemente.

Consiste en un *master* y múltiples *workers*, cada uno con una cantidad de memoria y número de *core*s configurados. Cada aplicación puedes solicitar la memoria a usar por sus *executor*s o el número total de *core*s a utilizar.

Soporta dos modos de despliegue donde el programa driver de tu aplicación se ejecuta:

⇨*Modo cliente* (por defecto)

El driver se ejecuta en la máquina donde se ejecuta el *spark-submit*.

Esto significa que si lo lanzas desde tu ordenador, puedes ver la salida de tu *driver program*, pero este requiere que tu máquina tenga una rápida conectividad a los *workers* y permanezca disponible durante toda la duración de la aplicación, es decir, no puedes apagarla.

⇨Modo *cluster*,

El driver es lanzado dentro de *cluster* Standalone, como otro proceso en uno de los *workernode*s, y este conecta de nuevo a petición de los *executor*s.

En este modo *Spark*-submit[22] es "fire-and-forget" y tú puedes apagar tu portátil mientras la aplicación se está ejecutandose. Para acceder al log de la aplicación, hay que acceder a la *Web UI* del gestor del *cluster*.

•*Apache Mesos*

Es un gestor de *cluster* de propósito general, que ejecuta tanto carga de trabajo analítica, como servicio de larga ejecución, como aplicaciones *web* o almacenamientos de clave-valor, que proporciona un aislamiento eficaz de los recursos y la compartición dinámica de recursos a través de aplicaciones o f*rameworks*.

Mesos ofrece dos modos de compartir recursos:

⇨*modo "grano fino"*

*L*os *executors* escal*a*n arriba y abajo el número de *CPU*s que demandan a *Mesos* mientras ejecutan las tareas y de esa forma una máquina ejecuta múltiples *executors* compartiendo *CPU* dinámicamente.

Este modo es conveniente cuando múltiples usuarios ejecutan cargas de trabajo interactiva como *shells*. Tener en cuenta que se añade cierta latencia en la situación de esperar cierta cantidad de tiempo para que se le asignen *CPU* de *core*s y aplicaciones de baja latencia, como *Spark* Streaming, pueden sufrir en su rendimiento.

⇨*modo "grano-grueso"*

[22] script al que le damos distintas opciones de elección de recursos del cluster para la ejecución de la aplicación en este.

Spark asigna un fijo número de *CPU*s a cada *executor* y nunca se liberan hasta que la aplicación termine.

•*Hadoop Yarn*

Es un gestor de *cluster* que permite mediante un concepto de cola, limitar el uso de recurso a varios conjuntos de aplicaciones, y es típicamente instalado sobre los mismos nodos que el sistema de ficheros de *Hadoop* (*HDFS*).

La documentación de *Spark* usa los términos *driver* y *executor* cuando describen los procesos que ejecutan cada aplicación *Spark*. Los términos *master* y *worker* son usados para describir las partes centralizadas y distribuidas de la gestión de un *cluster*.

Hadoop YARN ejecuta un servicio *master* (llamado *Resource Manager*) y varios servicios *worker* (llamado *Node Managers*). *Spark* ejecuta ambos, *drivers* y *executors* sobre los nodos *workers* de *YARN*.

Lanzamiento de la aplicaciónes al cluster con spark-submit

Una vez que tenemos el código de la aplicación preparado, este puede ser lanzado usando el script ***bin/spark-submit***, que se encarga de establecer, por ejemplo, el *cluster* manager o el modo de despliegue o distintas opciones para elegir qué recursos del *cluster* obtener para la ejecución de la aplicación.

Es importante tener en cuenta la flexibilidad que tenemos al poder asignar a distintas aplicaciones distintos recursos del *cluster* mediante varias opciones.

Veamos algunas de estas opciones:

```
./bin/Spark-submit \
  --class <main-class>
  --master <master-url> \
  --deploy-mode <deploy-mode> \
  --conf <key>=<value> \
  ... # other options
  <application-jar> \
  [application-arguments]
```

--class: La clase "main" de tu aplicación. Válido para *Java* o *Scala*.
--master: La URL del *cluster manager*. Por ejemplo, *spark://23.195.26.187:7077*

URL	descripción
local	Ejecuta localmente *Spark* con un hilo de trabajo (es decir, sin paralelismo).
local[K]	Ejecuta localmente *Spark* con K hilos de trabajo (idealmente, un hilo por *core* de la máquina).
local[*]	Ejecuta localmente *Spark* con tantos hilos de trabajos como *cores* tenga la máquina.
Spark://HOST:PORT	Conecta al *master* del *cluster* standalone de *Spark*.
mesos://HOST:PORT	Conecta al *cluster* Mesos.
YARN-client	Conecta al *cluster YARN* en modo cliente.
YARN-cluster	Conecta al *cluster YARN* en modo *cluster*.

Tabla 5. Lista de distintos formatos de una URL master pasada al cluster.

--deploy-mode: Como ya vimos, *cluster* o *client*.
--conf: Distintas propiedades de configuración de *Spark* en formato "clave = valor".
application-jar: Path a tu jar empaquetado incluyendo aplicación y dependencias.

application-arguments: Argumentos pasados a la aplicación *Spark*.
--name: nombre descriptivo de tu aplicación que será mostrado en la *Web UI*.
--files: lista de ficheros a ser situados en el directorio de trabajo de tu aplicación.
--py-files: lista de ficheros a ser añadidos al PYTHONPATH de tu aplicación.

Un ejemplo de lanzamiento de una aplicación al *cluster* sería crear un fichero .sh, por ejemplo myApp.sh, con el siguiente contenido:

```
../bin/Spark-submit
   --name "My app"
   --master local[4]
   --conf Spark.shuffle.spill=false
   --conf "Spark.executor.extraJavaOptions=-XX:+PrintGCDetails -
XX:+PrintGCTimeStamps" myApp.jar
```

Desde el shell[23] ejecutaríamos el fichero:

```
$ ./myApp.sh
```

Una vez lanzada la aplicación al *cluster Spark* podemos ver en la aplicación *web* UI, en http://<driver>:4040, accediento a las distintas pestañas.

[23] o intérprete de comandos es un programa para acceder a los servicios del sistema operativo.

4. Infraestructura Big data

En este capítulo describiremos paso a paso cómo, a partir de un conjunto de ordenadores conectados a una red de área local mediante un *switch* y la electrónica de red necesaria (cables de *ethernet*, tarjetas de red en cada equipo), instalaremos el software *Hadoop* y *Spark*, construyendo así un *cluster* para, posteriormente, configurarlo y ejecutar en él una sencilla aplicación con el paradigma *MapReduce*.

Finalmente elaboraremos una guía resumen del proceso de instalación, configuración y ejecución de un sencillo algoritmo.

4.1 Preparación del cluster

Describiremos en este apartado con qué computadores y de qué características contamos para construir el *cluster* y qué preparativos son los necesarios para que se instale el software con éxito.

4.1.1 Aula Anita Borg: recursos informáticos

El aula tiene 32 computadores conectados a un *switch* que nos habilita la red de área local y la salida a internet.

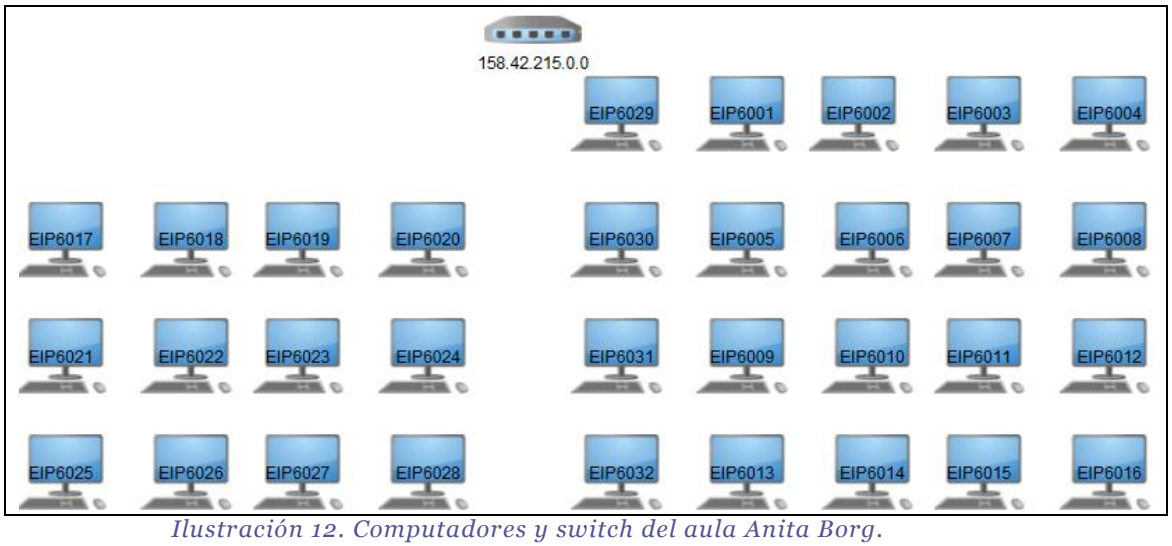


Ilustración 12. Computadores y switch del aula Anita Borg.

Las características de cada computador son:

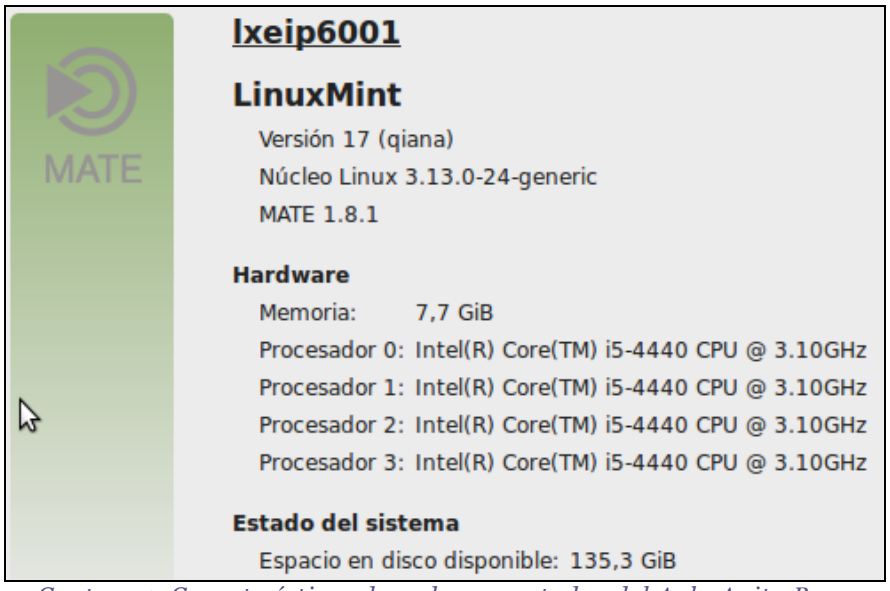
•**Sistema Operativo**: El equipo dispone de tres particiones: dos del sistema operativo *Windows* y una del sistema operativo *Linux*. Nosotros trabajaremos con la partición de *linux*, que tiene instalado el sistema operativo: *Linux Mint, versión 17 (qiana)*

•**Disco duro:** En nuestra partición de *linux* disponemos de 188.2 MB de capacidad.

•**Memoria RAM:** 8GB de capacidad.

•**Procesador:** Intel® *Core*™ i5-4440 *Processor* de 4 núcleos de hasta 3.30 GHz, 64 MB de caché y 64 bits de conjunto de instrucciones.

•**Tarjeta de red:** *ethernet* de 100 MBits de ancho de banda.



Captura 1. Características de cada computador del Aula Anita Borg.

4.1.2 Topología de red

Si tenemos la necesidad de unir diferentes ordenadores para que puedan intercambiar datos entre sí, estamos hablando de instalar una red de ordenadores.

A las redes las podemos clasificar de varias formas en función de la característica que consideremos; por ejemplo, en función de su alcance (en nuestro caso, en el aula Anita Borg, sería una Red de área local) o en función de su tipo de conexión (en nuestro caso sería guiado con cable coaxial) o en función de la topología de la red, que será donde nos extenderemos un poco más para nuestras tareas de configuración de *Hadoop*.

La topología de red [21] hace referencia a la forma de la red, a la disposición física o lógica de los ordenadores que determina la configuración de las conexiones entre los nodos. Se reconocen ocho tipos básicos de topologías: bus, estrella, árbol, mixta, anillo, doble anillo, malla y totalmente conexa.

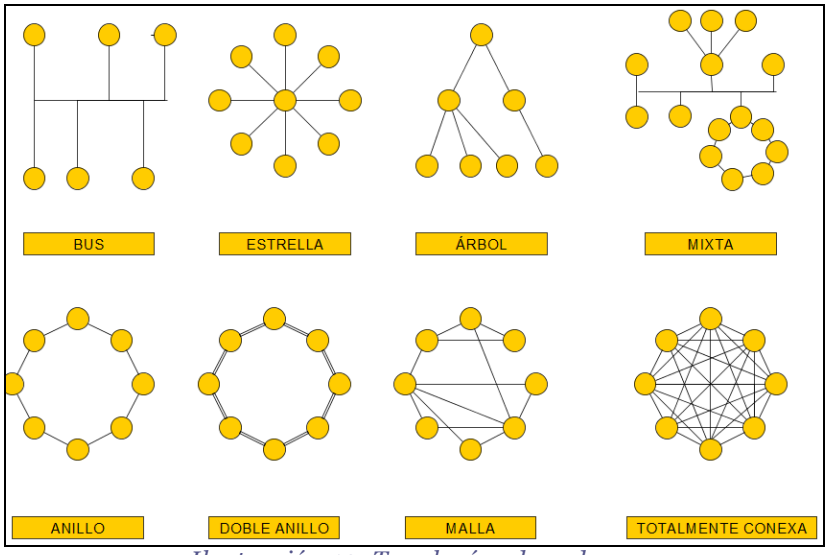


Ilustración 13. Topologías de red.

De los ocho tipos básicos de topologías, las que más nos encontramos en los *cluster Hadoop* y *Spark* son de la tipología en árbol y estrella.

La topología en árbol es una variante de la topología en estrella.

En una topología en estrella, todos y cada uno de los nodos de la red se conectan a un *switch*. Los datos en estas redes fluyen del emisor hasta el *switch*, el cual realiza todas las funciones de la red, además de actuar como amplificador de los datos.

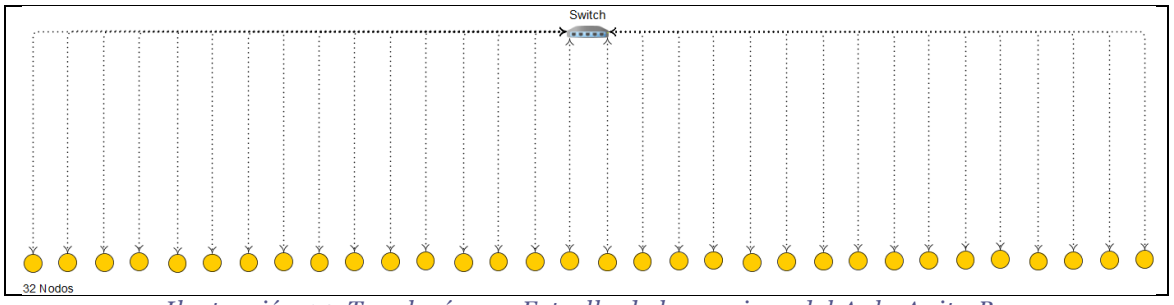


Ilustración 14. Topologías en Estrella de los equipos del Aula Anita Borg.

En una topología en árbol, la mayoría de los dispositivos se conectan a un *switch* secundario en un *rack,* que a su vez, se conecta al *switch* central. Es decir nos permite gestionar niveles, necesarios para organizar, por ejemplo, computadoras de una oficina en un primer piso conectadas a las del segundo piso o a un piso de otro edificio, como mostramos en la Ilustración 15.

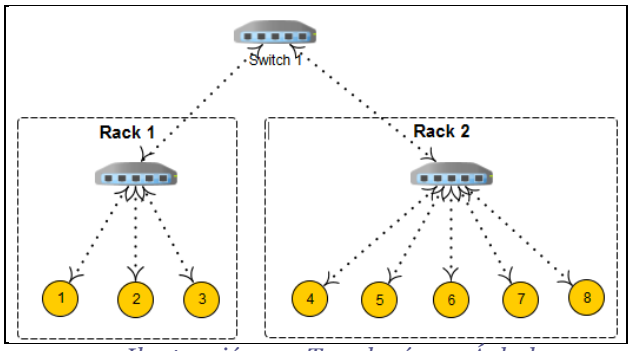


Ilustración 15. Topología en Árbol.

Para obtener el máximo rendimiento de *Hadoop* es importante configurar *Hadoop* para que conozca la topología de la red.

El *cluster* del aula Anita Borg se ejecuta sobre una topología de estrella, que es la configuración por defecto en *Hadoop* y no hay nada por configurar por parte del administrador del Sistema.

Y aunque para nuestros propósitos de configuración ya tenemos los suficientes conocimientos en este tema, hemos preferido adentrarnos un paso más en este importante tema con unas pocas pinceladas que completarán el conocimiento sobre la configuración de la topología de red en *Hadoop*.

Si se ejecutara nuestro *cluster* sobre una topología de árbol sería necesario mapear los nodos a **su** correspondiente *rack*. Al hacer esto, *Hadoop*, a la hora de tomar determinadas decisiones, preferirá hacer transferencia dentro de nodos de un mismo *rack* pues el ancho de banda disponible es mayor. Por ejemplo, el

namenode usa la localización de red cuando determina dónde situar las réplicas de los bloques y el ***jobtracker*** las usa para determinar dónde está la réplica más próxima como una entrada a un mapa de tareas que es planificado para ejecutarse sobre un ***tasktracker.***

Veremos una explicación a las propiedades de *Hadoop* que gestionan la topología de red en el apartado 4.3.2.1 de *core*-site.xml.

4.1.3 JRE. Java Runtime Environment.

La mayor parte de *Hadoop* está escrito *Java*, por lo cual es indispensable que en cada ordenador esté instalado JRE (*Java Runtime Environment*) para funcionar.

En el aula Anita Borg, ya hay una versión de JRE instalada en todos los ordenador con la versión 1.8.0_25. Para saber si hay una versión instalada, ejecutar el comando:

```
$ Java -version
```

Si no estuviera instalado JRE en nuestro sistema, lo más recomendable es acceder a la página oficial de *Java* que le guiará en el proceso de instalación y le ofrecerá el software necesario a su equipo: hardware y sistema operativo [22].

Para saber qué path de *Java* ponemos en la variable de entorno $*JAVA_HOME*, podemos encontrar un didáctico tutorial en [25]. Una vez que encuentre el path de *Java*, omita los directorios finales /bin/*java*, ya que en los ficheros .sh se les añade.

Dado que JRE es un pilar importante dentro del *cluster* lo más razonable es entenderlo y estudiarlo porque nos allanará el camino para los procesos de instalación, configuración y adaptación de las aplicaciones diseñadas por nosotros en el *cluster*.

Introducción

¿Por qué necesitamos JRE para ejecutar aplicaciones *JAVA*? Supongamos que un cliente nos encarga una aplicación y nosotros, en un entorno de desarrollo en *Windows*, le diseñamos el código fuente y la compilamos para que genere el fichero ejecutable. Le damos el fichero ejecutable al cliente que lo prueba en su sistema operativo *Windows* y funciona con éxito. Pero a los días nos llama y nos dice que la aplicación no funciona en un ordenador con *Linux*.

¿Qué ha pasado? Nuestra aplicación no es portable a otros sistemas. El código fuente que hemos diseñado lo hemos compilado para que genere un lenguaje máquina entendible para el sistema operativo *Windows* pero, ese fichero ejecutable, otro sistema operativo no lo entiende.

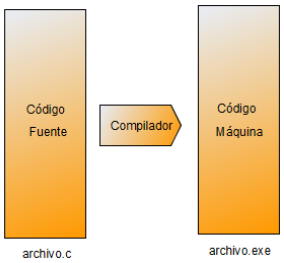


Ilustración 16. Proceso habitual de compilación.

¿Qué hace JRE? El compilador de *JAVA* compila el código fuente a un código máquina intermedio, llamado *Byte-code*, que el JRE interpreta según el sistema operativo en el que esté, pasándolo al código máquina que este entienda.

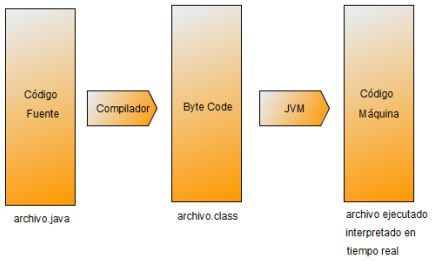


Ilustración 17. Proceso de compilación en Java.

Para cada dispositivo, por tanto, debe haber una JRE específica, ya sea un teléfono móvil, un PC con *Windows* XP o un microondas. De forma que JRE conoce el conjunto de instrucciones de la plataforma destino, y traduce un código escrito en lenguaje *Java* (común para todas) al código nativo que es capaz de entender el hardware de la plataforma.

JRE. Java Runtime Environment

JRE es un conjunto de utilidades que conforman un entorno en tiempo de ejecución que actúa como intermediario entre el sistema operativo y *Java*, de modo que cualquier aplicación *Java* pueda funcionar en cualquier dispositivo y sistema operativo que disponga de él.

Ilustración 18. Esquema conceptual de JRE [23].

JRE, está compuesto por:

• *JVM* (*Java Virtual Machine*): Máquina virtual de *Java*.

•Un conjunto de librerías o *APIs* que contienen código *JAVA* compilado.

JVM - Java Virtual Machine

Componente ejecutable que se encarga de interpretar y ejecutar cada una de las líneas de código "compilado" en *Java* (*bytecode*). Ya en una plataforma en particular puede ser compilado a código máquina para ser ejecutado con mayor rapidez que hacerlo interpretado, que va instrucción por instrucción.

Cada aplicación *Java* ejecutada, en un proceso diferente, se iniciará en una instancia diferente de la máquina virtual de *Java*.

Runtime data areas

El área de datos en tiempo de ejecución son áreas diseñadas para almacenar en memoria datos usados por el *JVM* para sus trabajos internos. La Ilustración 19 muestra una vista de conjunto de las diferentes áreas, unas únicas, como por ejemplo *Heap*, y otras, tantas áreas como hilos en ejecución, como *pcRegister* o *stack*.

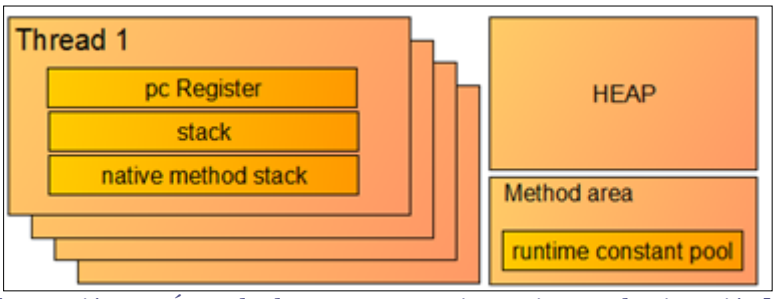


Ilustración 19. Área de datos en memoria en tiempo de ejecución[24].

Heap

Es un área de memoria compartida por todos los hilos de *JVM* en ejecución, creada en el arranque de la máquina virtual. Todas las instancias de clases y *array*, asignadas con el operador new, se asignan al *Heap*.

Esta zona debe ser gestionada por el *garbage collector* para borrar las instancias que no se usan más y liberar memoria. En *Hadoop* y *Spark* se pueden configurar distintas estrategias para limpiar la memoria.

El *Heap* puede ser dinámicamente expandida o contraída y puede tener un tamaño mínimo y máximo con los parámetros -Xms=512m -Xmx=1024m, con estos valores, como ejemplo. Si se excede el tamaño máximo la *JVM* lanza una excepción del tipo *OutOfMemoryError*.

Method area

Área compartida por todas los hilos de la *JVM*. Los datos en él permanecen en memoria tanto tiempo como la *classloader* que lo cargó esté viva y se almacena en un espacio separado de memoria nativa llamada *Metaspace*, cuyo máximo disponible es el total del sistema de memoria disponible.

Runtime constant pool

Se incrementa con cada clase/interface cargada. Es como una Tabla de símbolos para un lenguaje de programación convencional. Cuando una clase, método,

campo o constante es referido, la *JVM* busca la dirección actual en la memoria usando este depósito.

The pc Register (por hilo)

Cada hilo tiene su propio regristro pc (contador de programa) que contiene la dirección de la instrucción *JVM* que está siendo ejecutada actualmente.

Stacks (por hilo)

Este área almacena múltiples *frames*. Un *frame* es una estructura que contiene múltiples datos que representan el estado del hilo en el método actual que es creado cuando el método es invocado y destruido cuando el método es completado.

Native method stack (por hilo)

Es una pila para código nativo escrito en otro lenguaje de programación y llamado a través de (*Java Native Interface*).

Otros aspectos

En el apartado posterior *4.4.4.3 Ajuste del Garbage Collection(GC)* volveremos a la *JVM* para ver su modelo de memoria y cómo gestiona en él sus áreas de datos en ejecución, conceptos necesarios para entender la *"recolección de basura"* de datos en memoria (*Garbage Collection*) y su ajuste.

4.1.4 Creación de un usuario Hadoop en cada equipo.

Es una buena práctica crear una cuenta de usuario dedicado a *Hadoop* para separar la instalación de *Hadoop* de otros servicios que se ejecutan en la misma máquina.

Para cada máquina del aula Anita Borg crearemos un mismo usuario llamado *usulocal* y le pondremos una contraseña:

```
$ sudo useradd -m usulocal
$ sudo passwd usulocal
```

4.1.5 Generar par de claves RSA para autenticar por SSH.

Para instalar y configurar el *cluster* tendremos que acceder a todos los ordenadores del mismo.

Comando SSH

Para no ir físicamente de ordenador en ordenador, utilizaremos el **comando SSH** y desde uno de los ordenadores del *cluster*, en nuestro caso EIP6029, accederemos al resto de ordenadores para realizar dichas tareas.

RSA

Por otro lado, cuando accedamos mediante SSH a cada uno de los ordenadores nos solicitarán, como es lógico, una contraseña para iniciar la sesión y puede resultar bastante tedioso con tantos equipos a los que acceder. ¿Cómo evitar poner la contraseña cada vez que intentemos acceder a un equipo de forma remota sin comprometer la seguridad? ¡ Usando la autenticación basada en claves **RSA**!

RSA es un sistema criptográfico de clave pública válido tanto para cifrar como para firmar digitalmente, donde cada usuario posee dos claves de cifrado: una pública y otra privada.

Cuando se quiere enviar un mensaje cifrado, el emisor busca la clave pública del receptor, cifra su mensaje con esa clave, y una vez que el mensaje cifrado llega al receptor, este se ocupa de descifrarlo, solo él puede, usando su clave privada.

Proceso de autenticación basada en clave RSA por SSH

•El equipo *A* se conecta por ssh al equipo *B* y se negocia una sesión encriptada.
•*A* envia su clave pública, id_rsa.pub, a *B*.
• *B* comprueba que la clave pública está en su authorized_keys.
•Una vez la clave está comprobada, *B* envía a *A* un desafío: un número aleatorio encriptado con la clave pública id_rsa.pub de *A*.
•Si *A* puede devolver el número aleatorio sin cifrar, quiere decir que tiene la clave privada (porque no existe otra forma de desencriptar el número del desafío).
• *B* da su autorización para el inicio de la sesión SSH.

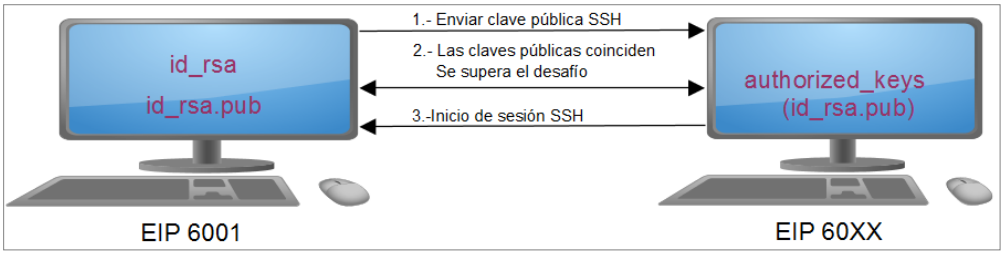


Ilustración 20. Esquema de autenticación basada en clave RSA por SSH.

Aplicación en el cluster del Aula Anita Borg

1º.-En el equipo EIP6029 creamos el par de claves publica/privada RSA.

```
$ cd
$ mkdir -p .ssh
$ cd .ssh
$ ssh-keygen -t dsa -P '' -f id_dsa  #ojo: "son dos comillas simples sin espacio
```

```
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:
13:e1:11:5e:12:86:2c:96:fa:01:a4:02:31:67:de:29 usulocal@localhost.localdomain
The key's randomart image is:
+--[ DSA 1024]----+
|+o+  o .Bo.      |
|.*..+.o+ =       |
|o E+o.  +        |
|. ...    .       |
|   . .  S        |
|    .    .       |
|                 |
|                 |
|                 |
+-----------------+
```

```
$ ls -l
```

```
total 8
-rw-------. 1 usulocal usulocal 668 ago 10 20:20 id_dsa
-rw-r--r--. 1 usulocal usulocal 620 ago 10 20:20 id_dsa.pub
```

2º.- Para cada ordenador, realizamos dos pasos:

a.-Añadir el fichero *id_dsa.pub* al fichero ~/.ssh/authorized_keys.

```
# Desde el ordenador EIP6029
$ cd
```

```
$ cd .ssh
$ ssh usulocal@<ip_worker-node>  mkdir -p .ssh
$ cat id_dsa.pub | ssh usulocal@<ip_worker-node> "cat - >>.ssh/authorized_keys"
```

b.-Modificar los permisos para poder acceder sin contraseña.

```
# Desde EIP6029 accedemos al ordenador destino y modificamos permisos
$ ssh usulocal@<ip_worker-node>  #sustituir <ip_worker-node>  por una IP
# Solicita contraseña y se le facilita
$ cd ~/.ssh
$ chmod go-rwx .
$ chmod go-rw authorized_keys
```

3º.-Y ya podemos acceder de forma remota desde EIP6029 al resto de equipos sin introducir contraseña alguna:

```
ssh usulocal@<ip_worker-node> #sustituir <ip_worker-node>  por una IP
```

4.1.6 Diseño del mapa del cluster.

Típicamente una de las máquina en el *cluster* se designa como ***NameNode*** y ***Spark Master*** y otra máquina como ***ResourceManager***, exclusivamente. Esto son los *master*s. El resto de las máquinas en el *cluster* actúan como ***DataNode***, ***NodeManager*** y ***Spark Slave***. Estos son los *slaves* (esclavos) o *workernodes* (nodos trabajadores).

El mapa del *cluster* en el aula Anita Borg será el siguiente:

Ordenador 6029: *HDFS NameNode* + *Spark Master*

Ordenador 6001: *YARN ResourceManager*+ JobHistoryServer+ProxyServer

Resto Ordenadores: *HDFS DataNode*+*YARN NodeManager*+*Spark* Slave

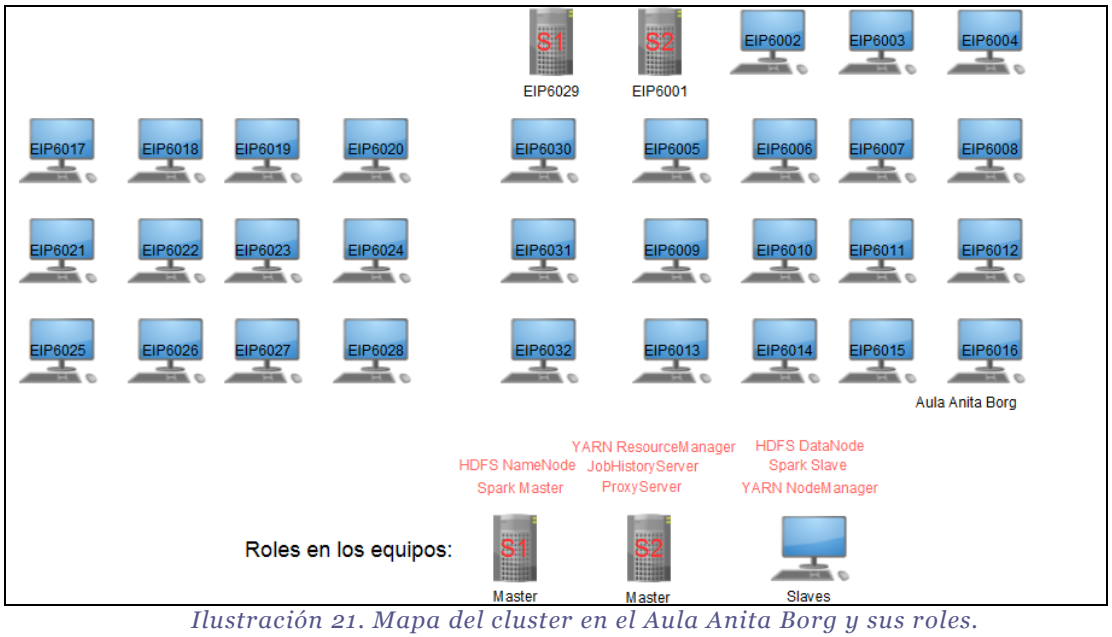


Ilustración 21. Mapa del cluster en el Aula Anita Borg y sus roles.

4.1.7 Versiones software de Hadoop y Spark a instalar.

Instalaremos *Hadoop* y *Spark*, en sus versiones binarias más recientes pero estables. La versión de *Spark* debe ser la correspondiente a *Hadoop*.

Teniendo en cuenta esas consideraciones, las versiones son:

•Apache *Hadoop* version 2.6.2

•Apache *Spark* version 1.5.2

4.2 Instalación Apache Hadoop y Apache Spark

La instalación de Apache *Hadoop* versión 2.6.2 y Apache *Spark* versión 1.5.2 en los 32 equipos del aula Anita Borg, a efectos prácticos, lo realizaremos en 3 fases:

•Instalación en equipo EIP6029: Obtener y descomprimir los ficheros de instalación de *Hadoop* y *Spark* en el equipo EIP6029.

•Configurar el equipo EIP6029: Ajustar los distintos ficheros de configuración.

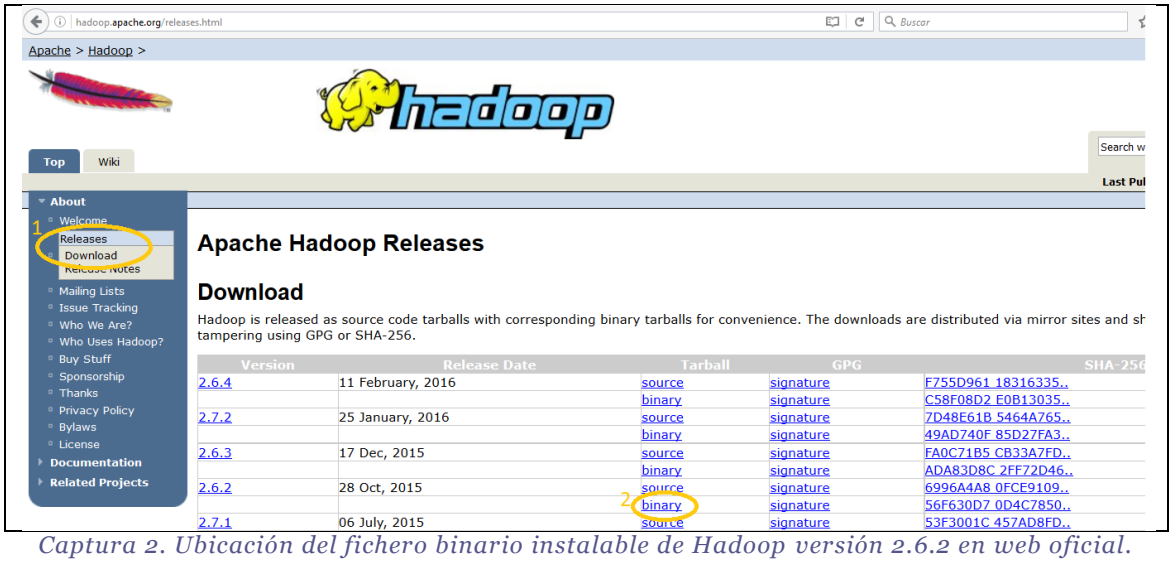
•Desplegar en el resto de equipo: Copiar los directorios de *Hadoop* y *Spark* del equipo EIP6029 en los otros 31 equipos, finalizando la instalación en sí.

En este apartado veremos cómo obtener los ficheros instalables y los descomprimeros en el equipo EIP6029.

4.2.1 Obtener ruta de los ficheros instalables

Apache Hadoop

Entramos en la página oficial de *Hadoop* http://*Hadoop*.apache.org/ y dentro de la opción de menú *Releases*, pulsamos *Download*.



Captura 2. Ubicación del fichero binario instalable de Hadoop versión 2.6.2 en web oficial.

De la versión ***2.6.2*** pulsamos en ***binary***, y se accedemos a *The Apache Software Foundation*.

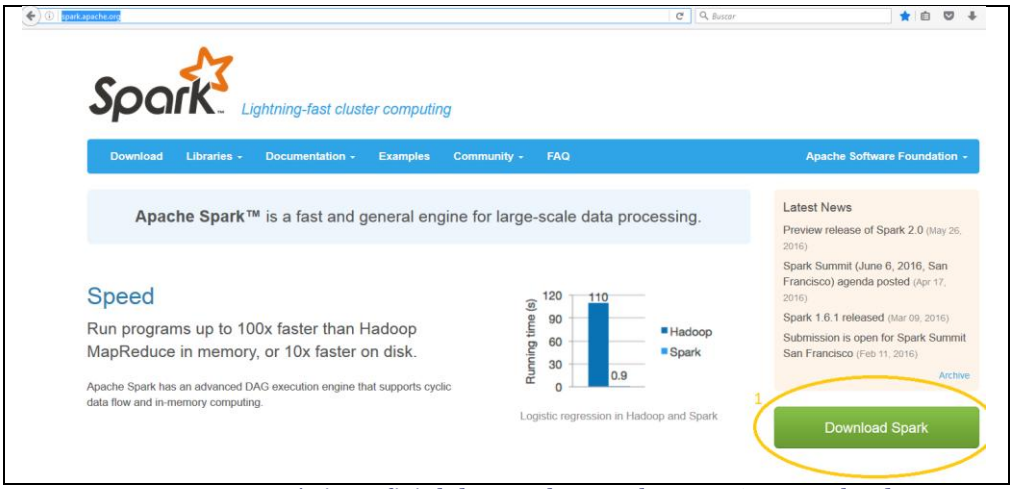
Captura 3. The Apache Software Foundation: Hadoop.

Nos apuntamos uno de los enlaces al fichero ***hadoop-2.6.2.tarz.gz*** para usar posteriormente: http://apache.rediris.es/Hadoop/common/Hadoop-2.6.2/Hadoop-2.6.2.tar.gz

Apache Spark

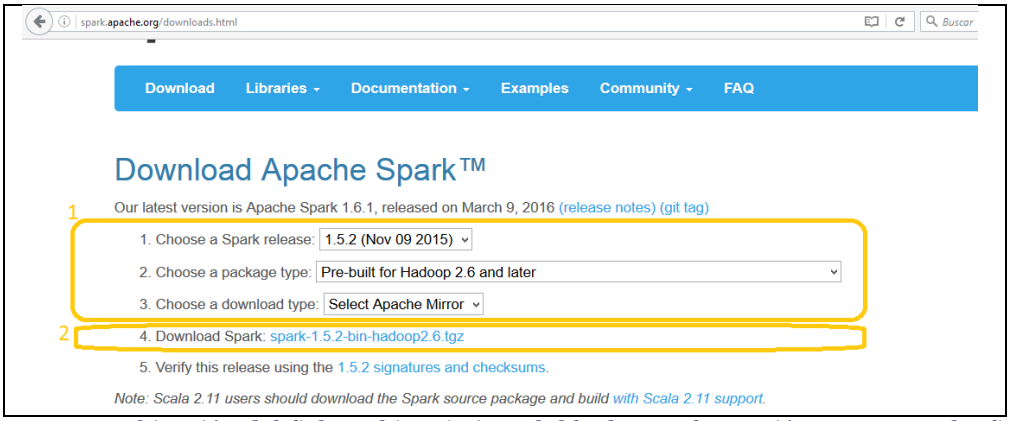
Entramos en la página oficial de Apache *Spark* http://*Spark*.apache.org/

y pulsamos en ***download***.



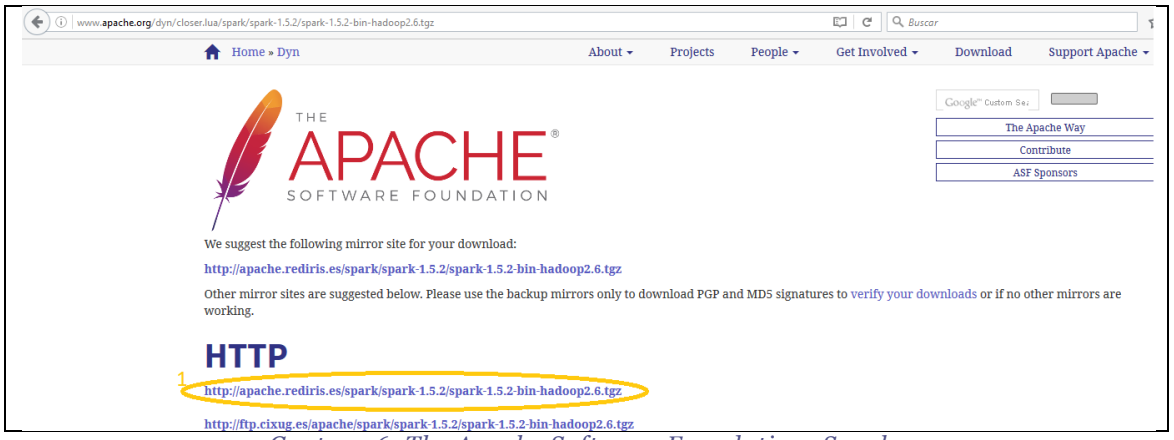
Captura 4. Página oficial de Apache Spark. Pestaña Download.

Seleccionamos la versión de despliegue ***1.5.2*** de Apache *Spark* y como tipo de paquete ***Pre-built for Hadoop 2.6 and later***. Pulsamos en el nombre del fichero ***spark-1.5.2-bin-hadoop2.6.tgz*** y select *Apache Mirror*.



Captura 5. Ubicación del fichero binario instalable de Spark versión 1.5.2 en web oficial.

Pulsamos sobre el nombre del fichero ***spark-1.5.2-bin-hadoop2.6.tgz*** y se accede a la página *de The Apache Software Foundation.*



Captura 6. The Apache Software Foundation: Spark.

Nos apuntamos uno de los enlaces al fichero ***spark-1.5.2-bin-hadoop2.6.tgz*** para usar posteriormente: http://apache.rediris.es/*Spark*/*Spark*-1.5.2/*Spark*-1.5.2-bin-*Hadoop*2.6.tgz

4.2.2 Instalación de Apache Hadoop y Apache Spark

Desplegamos los instalables de *Hadoop* y *Spark* en el equipo E6029 del Aula Anita Borg.

```
# Nos situamos en ordenador 6029
# Descargamos instalables de Hadoop y Spark
$ cd
$ mkdir -p cluster/opt
$ cd cluster/opt
$ wget http://apache.rediris.es/Hadoop/common/Hadoop-2.6.2/Hadoop-2.6.2.tar.gz
$ wget http://apache.rediris.es/Spark/Spark-1.5.2/Spark-1.5.2-bin-Hadoop2.6.tgz
# Desplegamos, descomprimiendo, Hadoop en el directorio /cluster/opt
$ tar zxvf  Hadoop-2.6.2.tar.gz
# Creamos un enlace dinámico para trabajar con /cluster/opt/Hadoop
$ rm Hadoop
$ ln -s Hadoop-2.6.2 Hadoop
# Desplegamos, descomprimiendo,  Spark en el directorio /cluster/opt
$ cd ~/cluster/opt
$ tar zxvf Spark-1.5.2-bin-Hadoop2.6.tgz
$ rm Spark
$ ln -s Spark-1.5.2-bin-Hadoop2.6 Spark
```

4.3 Configuración Apache Hadoop

Pasamos al corazón del trabajo. Dado que ambos sistemas, *Hadoop* y *Spark*, tienen cientos de parámetros a configurar, elegiremos, como primera aproximación en el aprendizaje, los que creemos nos darán más margen de maniobra para poder enfrentarnos en el futuro a distintos tipos de algoritmos y, por otro lado, que podamos acoplar a nuestro *cluster* con éxito los dos algoritmos planteados en el TFG.

En el anexo mostramos distintos ficheros de configuración con su número total de parámetros y la descripción de algunos de ellos para que el lector se haga una idea de la dimensión de diversidad de parámetros que estamos manejando.

Para configurar el *cluster Hadoop* plantearemos una serie de pasos a seguir aunque previamente necesitaremos configurar :

• el entorno en el cual los servicios de *Hadoop* se ejecutan.

• los parámetros de configuración de los servicios de *Hadoop*.

• los mensajes de información que muestra nuestra aplicación en ejecución en el *cluster*.

• el fichero *slaves* rellenándolo con los *worker nodes* del *cluster*.

4.3.1 Ajustando el entorno de los servicios Hadoop

Los administradores deben usar los scripts *yarn-env.sh*, *hadoop-env.sh* y *mapred-env.sh* para configurar el entorno de los procesos de los servicios de *Hadoop*. Todos estos scripts se encuentran en la ruta: *~/cluster/opt/hadoop/etc/hadoop/*

4.3.1.1 yarn-env.sh

Script que ejecuta *YARN* de inicio donde se especifican variables de entorno como el tamaño del *Heap*, el directorio de logs o el directorio de ficheros pid.

Veamos algunas variables de entorno a ser configuradas:

*YARN*_LOG_DIR: Directorio donde los ficheros de logs son almacenados.

YARN_MASTER: Path del host donde está el código *Hadoop* para ser usado por rsync.

JAVA_HOME: Determinamos la localización de la implementación de JRE a usar. Para obtener su valor, repasar el apartado *4.1.3 JRE. Java Runtime Environment.*

Podríamos darle el path de *Java* por la variable de entorno del Shell; pero si lo hacemos con este parámetro, nos aseguramos que el *cluster* entero tiene la misma versión de *Java*.

*YARN_HEAP*SIZE: Máxima cantidad del tamaños del *Heap*, *heap*size, de la máquina virtual a utilizar. Son usados para configurar el tamaño del *Heap* para el servicio siendo por defecto 1000 MB; no obstante, si se quisiera configurar los valores de forma separada para cada servicio, se podría usar las siguientes variables:

Servicio	Variable de entorno	Ubicación
ResourceManager	*YARN_RESOURCEMANAGER_HEAP*SIZE	*YARN*-env.sh
NodeManager	*YARN_NODEMANAGER_HEAP*SIZE	*YARN*-env.sh
WebAppProxy	*YARN*_PROXYSERVER_*HEAP*SIZE	*YARN*-env.sh

Tabla 6.Variable de entorno Yarn_Heapsize para los distintos servicios de Yarn.

*YARN*_servicio x_OPTS: Variables de entorno específicas para cada servicio y en ella se pueden configurar todos los *flags* de la *JVM*.

Demonio	Variable de entorno	Ubicación
ResourceManager	*YARN_RESOURCEMANAGER*_OPTS	*YARN*-env.sh
NodeManager	*YARN_NODEMANAGER*_OPTS	*YARN*-env.sh
WebAppProxy	*YARN*_PROXYSERVER_OPTS	*YARN*-env.sh

Tabla 7. Variable de entorno OPTS de servicios de Yarn para ajustar JVM.

Configuración en cluster del Aula Anita Borg

En nuestro *cluster*, al fichero YARN-env.sh recién instalado, le hemos añadido las siguientes líneas:

```
export YARN_MASTER="eip6001.inf.upv.es:/HOME/usulocal/cluster/opt/Hadoop/"
export JAVA_HOME="/opt/jdk1.8.0_25"
```

Por tanto, el fichero *YARN*-env.sh tendría el siguiente aspecto:

```
export YARN_MASTER="eip6001.inf.upv.es:/home/usulocal/cluster/opt/hadoop/"

# User for YARN daemons
export HADOOP_YARN_USER=${HADOOP_YARN_USER:-yarn}

# resolve links - $0 may be a softlink
export YARN_CONF_DIR="${YARN_CONF_DIR:-$HADOOP_YARN_HOME/conf}"

# some Java parameters
export JAVA_HOME="/opt/jdk1.8.0_25"
if [ "$JAVA_HOME" != "" ]; then
  #echo "run java in $JAVA_HOME"
  JAVA_HOME=$JAVA_HOME
fi
if [ "$JAVA_HOME" = "" ]; then
  echo "Error: JAVA HOME is not set."
  exit 1
fi
JAVA=$JAVA_HOME/bin/java
JAVA_HEAP_MAX=-Xmx1000m

# check envvars which might override default args
if [ "$YARN_HEAPSIZE" != "" ]; then
  JAVA_HEAP_MAX="-Xmx""$YARN_HEAPSIZE""m"
fi

# default log directory & file
if [ "$YARN_LOG_DIR" = "" ]; then
  YARN_LOG_DIR="$HADOOP_YARN_HOME/logs"
fi
if [ "$YARN_LOGFILE" = "" ]; then
  YARN_LOGFILE='yarn.log'
fi

# default policy file for service-level authorization
if [ "$YARN_POLICYFILE" = "" ]; then
  YARN_POLICYFILE="hadoop-policy.xml"
fi

# restore ordinary behaviour
unset IFS

YARN_OPTS="$YARN_OPTS -Dhadoop.log.dir=$YARN_LOG_DIR"
YARN_OPTS="$YARN_OPTS -Dyarn.log.dir=$YARN_LOG_DIR"
YARN_OPTS="$YARN_OPTS -Dhadoop.log.file=$YARN_LOGFILE"
YARN_OPTS="$YARN_OPTS -Dyarn.log.file=$YARN_LOGFILE"
YARN_OPTS="$YARN_OPTS -Dyarn.home.dir=$YARN_COMMON_HOME"
YARN_OPTS="$YARN_OPTS -Dyarn.id.str=$YARN_IDENT_STRING"
YARN_OPTS="$YARN_OPTS -Dhadoop.root.logger=${YARN_ROOT_LOGGER:-INFO,console}"
YARN_OPTS="$YARN_OPTS -Dyarn.root.logger=${YARN_ROOT_LOGGER:-INFO,console}"
if [ "x$JAVA_LIBRARY_PATH" != "x" ]; then
  YARN_OPTS="$YARN_OPTS -Djava.library.path=$JAVA_LIBRARY_PATH"
fi
YARN_OPTS="$YARN_OPTS -Dyarn.policy.file=$YARN_POLICYFILE"
```

Captura 7. yarn-env.sh.

4.3.1.2 hadoop-env.sh

Script que especifica variables de entorno que afectan a la ejecución de los servicios *Hadoop*. Veamos cómo configurar algunas de ellas:

HADOOP_HOME: Directorio donde se ha instalado *Hadoop*.

JAVA_HOME: Determinamos la localización de la implementación de JRE a usar, que vimos cómo obtenerla en el apartado *4.1.3 JRE. Java Runtime Environment*.

*HADOOP_HEAP*SIZE: Entero pasado a la *JVM* como el argmento de memoria *Heap* máxima (Xmx). Por defecto toma el valor 1000 MB. De esta forma asigna el tamaño de *Heap* de la *JVM* de todos los servidores de proyectos *Hadoop*, tales como *HDFS*, *YARN* y *MAPREDUCE*.

Todos los procesos *Hadoop* se ejecutan sobre la máquina virtual de *Java* (*JVM*). El número de máquinas virtuales está en función del modo de despliegue del *cluster*. Los modos son:

• *local* (o standalone): No hay servicios y todo se ejecuta en una sola *JVM*.

•*Pseudo-distribuido*: Cada servicio se ejecuta sobre su propia *JVM* en un host único.

•*Distribuido*: Cada servicio se ejecuta sobre su propia *JVM* a través de un *cluster* de host.

*HADOOP_*<Servicio x>*_OPTS*: Variables de entorno específicas para cada servicio y en ella se pueden configurar todos los *flag*s de la *JVM*:

Servicio	Variable de entorno	Ubicación
NameNode	*HADOOP_NAMENODE_*OPTS	*Hadoop*-env.sh
DataNode	*HADOOP_DATANODE_*OPTS	*Hadoop*-env.sh
Secondary *NameNode*	*HADOOP_*SECONDARY*NAMENODE_*OPTS	*Hadoop*-env.sh

Tabla 8. Variable de entorno OPTS de servicios de HDFS para ajustar JVM.

*HADOOP_NAMENODE_*OPTS: Variables de entorno específico para el servicio *NameNode* y en él se puede configurar cada *flag*s de la *JVM*.

Un *NameNode* puede consumir mucha memoria ya que mantiene una referencia a cada bloque de cada fichero en memoria.

Se puede incrementar esta memoria sin cambiar la memoria asignada a los otros servicios de *Hadoop*, por lo tanto *HADOOP_NAMENODE_OPTS* sobreescribe el valor Xmx *HADOOP_HEAPSIZE* para el *NameNode*.

Dos *flag*s que pueden resultar útiles:

XX:+HeapDumpOnOutOfMemoryError: Al ocurrir un error de memoria, se ejecuta un dump del *Heap*. Habría que usarlo con la siguiente propiedad.

-XX:HeapDumpPath para la localización del fichero dump. Por ejemplo:

```
-XX:+HeapDumpOnOutOfMemoryError -
XX:HeapDumpPath=./etc/Heapdump.hprof
```

*HADOOP_*SECONDARY*NAMENODE_*OPTS: Si cambiamos la asignación de memoria al *NameNode*, no debemos olvidar hacerlo con el secondary*NameNode*, para tener configurado el nodo de reserva de la misma forma.

```
HADOOP_SECONDARYNAMENODE_OPTS=$HADOOP_NAMENODE_OPTS
```

*HADOOP_*LOG_DIR: Establece la localización del sistema de ficheros log.

Por defecto los ficheros log producidos por *Hadoop* son almacenados *HADOOP_*INSTALL/logs.

Es conveniente cambiarlo y guardarlos fuera de los directorios de una instalacióon *Hadoop* porque podrían perderse despues de una actualización de *Hadoop*. Un directorio común podría ser /var/log/*Hadoop*.

```
export HADOOP_LOG_DIR=/var/log/Hadoop
```

Cada servicio *Hadoop* ejecutado en una máquina produce dos ficheros de log:

• .log: escrito vía log4j es al que primero se recurre cuando se diagnostica un problema, ya que la mayoría de los mensajes de logs de las aplicaciones se escriben aquí. •.out: es la combinación de la salida estándar y el error estándar.

*HADOOP*_IDENT_STRING: Cambiar la estructura de nombres de los ficheros log.

El nombre de los ficheros Log son una combinación de: nombre del usuario que ejecuta el servicio, nombre del servicio y *hostname* de la máquina. Esta estructura de nombres los hace únicos y nos permite archivar todos los logs del *cluster* en un único directorio. Con esta opcion podríamos cambiar esta estructura de nombres según nuestra conveniencia.

*HADOOP*_PID_DIR y *HADOOP*_SECURE_DN_PID_DIR: Se configuran para que apunten a directorios que pueden ser solo escritos por los usuarios que lanzan los servicios de *Hadoop*, ya que puede haber riesgo potencial de un ataque del tipo *symlink*[24].

Los scripts de control nos permiten ejecutar comandos sobre *workers* remotos desde el *master* usando SSH, por lo que puede ser útil configurarlo con estas dos variables y con *HADOOP*_SSH_OPTS.

*HADOOP*_SSH_OPTS: Permite poner opciones extras al comando ssh.

Veamos un par de opciones, no obstante en las páginas del manual de ssh y ssh_config hay más [26]:
•*connection timeout* (tiempo de expiración de la conexión): para que el script de control no se estanque esperando a que el nodo le responda. Nos puede ser útil reducir este tiempo, pero no nos conviene bajarlo demasiado porque podemos dar por muerto a un nodo que sencillamente está muy ocupado.

•*StrictHostKeyChecking:* determina si deberías ignorar errores cuando conectas a la máquina remota y es útil para controlar el inicio de sesión a máquinas cuya clave de host no es conocido o ha cambiado. Puesto a 'No' añade automáticamente la nueva clave de host al fichero de hosts conocidos preguntado al usuario para confirmar que ha verificado la clave pública. No es muy adecuado para entornos de grandes *cluster*.

*HADOOP*_INSTALL: Apunta al directorio creado después de la extracción de *Hadoop*.

HADOOP_MASTER: Path del host donde está el código *Hadoop* para ser utilizado por rsync.

Los scripts de control de Hadoop pueden distribuir ficheros de configuración a todos los nodos usando rsync. No está activado por defecto.

Al definir HADOOP_MASTER, los servicios de los *workers* sincronizarán los ficheros de configuración en HADOOP_MASTER al directorio HADOOP_INSTALL del nodo local cuando quiera que los servicios del cluster arranquen.

[24] Problema de seguridad que haciendo uso de enlaces simbólicos puede llegar a crear o sobreescribir ficheros y directorios de forma remota.

*HADOOP*_SLAVE_SLEEP: Cuando arranca un *cluster* grande con rsyncing activado, los nodos *workers* pueden abrumar al nodo *master* con peticiones rsync ya que los *workers* arrancan casi al mismo tiempo. Para evitar esto, ponemos esta variable a un pequeño número de segundos, por ejemplo 0.1.

Configuración en cluster del Aula Anita Borg

En nuestro *cluster*, al fichero *Hadoop*-env.sh recién instalado, le hemos añadido las siguiente línea:

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

Por tanto, el fichero *hadoop.env.sh* tendría el siguiente aspecto:

```
# The java implementation to use.
#export JAVA_HOME=${JAVA_HOME}
export JAVA_HOME="/opt/jdk1.8.0_25"

export HADOOP_CONF_DIR=${HADOOP_CONF_DIR:-"/etc/hadoop"}

# Extra Java CLASSPATH elements.  Automatically insert capacity-scheduler.
for f in $HADOOP_HOME/contrib/capacity-scheduler/*.jar; do
  if [ "$HADOOP_CLASSPATH" ]; then
    export HADOOP_CLASSPATH=$HADOOP_CLASSPATH:$f
  else
    export HADOOP_CLASSPATH=$f
  fi
done

# Extra Java runtime options.  Empty by default.
export HADOOP_OPTS="$HADOOP_OPTS -Djava.net.preferIPv4Stack=true"

# Command specific options appended to HADOOP_OPTS when specified
export HADOOP_NAMENODE_OPTS="-Dhadoop.security.logger=${HADOOP_SECURITY_LOGGER:-INFO,RFAS}
      -Dhdfs.audit.logger=${HDFS_AUDIT_LOGGER:-INFO,NullAppender} $HADOOP_NAMENODE_OPTS"
export HADOOP_DATANODE_OPTS="-Dhadoop.security.logger=ERROR,RFAS $HADOOP_DATANODE_OPTS"

export HADOOP_SECONDARYNAMENODE_OPTS="-Dhadoop.security.logger=${HADOOP_SECURITY_LOGGER:
      -INFO,RFAS} -Dhdfs.audit.logger=${HDFS_AUDIT_LOGGER:-INFO,NullAppender}
$HADOOP_SECONDARYNAMENODE_OPTS"

export HADOOP_NFS3_OPTS="$HADOOP_NFS3_OPTS"
export HADOOP_PORTMAP_OPTS="-Xmx512m $HADOOP_PORTMAP_OPTS"

# The following applies to multiple commands (fs, dfs, fsck, distcp etc)
export HADOOP_CLIENT_OPTS="-Xmx512m $HADOOP_CLIENT_OPTS"
# using non-privileged ports.
export HADOOP_SECURE_DN_USER=${HADOOP_SECURE_DN_USER}
# Where log files are stored in the secure data environment.
export HADOOP_SECURE_DN_LOG_DIR=${HADOOP_LOG_DIR}/${HADOOP_HDFS_USER}
export HADOOP_PID_DIR=${HADOOP_PID_DIR}
export HADOOP_SECURE_DN_PID_DIR=${HADOOP_PID_DIR}

# A string representing this instance of hadoop. $USER by default.
export HADOOP_IDENT_STRING=$USER
```

Captura 8.hadoop-env.sh.

4.3.1.3 mapred-env.sh

Para el servicio *Map Reduce* Job History Server tenemos estas dos variables de entornos para ajustar los valores de la *JVM*.

Servicio	Variable de entorno	Ubicación
Map Reduce Job History Server	HADOOP_JOB_HISTORYSERVER_OPTS	mapred-env.sh
Map Reduce Job History	HADOOP_JOB_HISTORYSERVER_HEAPSIZE	mapred-

Server		env.sh

Captura 9. Variable de entorno de MapReduce Job History Server para ajustar JVM.

Configuración en cluster del Aula Anita Borg

En nuestro *cluster*, al fichero mapred-env.sh recién instalado, le hemos añadido las siguiente línea:

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

Por tanto, el fichero tendría el siguiente aspecto:

```
# export JAVA_HOME=/home/y/libexec/jdk1.6.0/
export JAVA_HOME="/opt/jdk1.8.0_25"

export HADOOP_JOB_HISTORYSERVER_HEAPSIZE=1000

export HADOOP_MAPRED_ROOT_LOGGER=INFO,RFA
```

Captura 10. mapred-env.sh.

4.3.2 Ajustando los servicios Hadoop

Los servicios de *Hadoop* son:

- *NameNode/DataNode.*
- *ResourceManager/NodeManager.*
- Secondary *NameNode.*
- WebAppProxy.
- *Map Reduce* Job History Server.

Cada componente en *Hadoop* es configurado usando dos ficheros XML.

Ficheros de configuración	Componente
core-site.xml / core-default.xml	Core
HDFS-site.xml / HDFS-default.xml	HDFS
mapred-site.xml / mapred-default.xml	MapReduce
YARN-site.xml / YARN-default.xml	YARN

Tabla 9. Ficheros de configuración de los componentes de Hadoop.

La configuración de los componentes se conforman con dos importantes grupos de ficheros debido a que *Hadoop* separa las propiedades por defecto del sistema, de las que personalizemos en nuestro *cluster*, en ficheros de configuración:

⇨ ***de solo-lectura***

Ficheros que recogen las propiedades del sistema y sus valores por defecto:
core-default.xml,
hdfs-default.xml,
yarn-default.xml
mapred-default.xml

⇨***de configuración específica***

Ficheros ubicados en *~/cluster/opt/hadoop/etc/hadoop/*, que aceptan los valores que queremos cambiar de las propiedades de *-default.xml y que sobreescribirá el valor dado por defecto a las propiedades de nuestro *cluster*.

core-site.xml,
hdfs-site.xml,

yarn-site.xml
mapred-site.xml

En conclusión, las propiedades especificadas en los ficheros *-site.xml sobreescriben a los que estén en *-default.xml y el resto serán atribuidos desde *-default.xml.

Las propiedades están configuradas en formato XML usando las etiquetas:
•*propiedad*: atributo con el nombre de la propiedad a configurar
•*valor*: valor que le damos a la propiedad
•*descripción*: breve descripción de la propiedad.
•*final*: atributo que marca la propiedad como final. Sus valores posibles son: {true, false}. Si un administrador no quiere que un cliente actualice el valor de un parámetro entonces definirá la propiedad como final.

Por ejemplo, configuremos la propiedad YARN.acl.enable:

```
<property>
<name>YARN.acl.enable</name>
<value>true</value>
<final>true</final>
</property>
```

En estos ficheros *.xml, los valores que hacen referencia al tamaño, por defecto, aparecen en bytes. Para configurar los valores podemos especificar el tamaño proporcionando al valor una unidad: k(kilo), m(mega), g(giga), t(tera), p(peta), e(exa) o, sencillamente, proporcionándole el tamaño completo en bytes. Por ejemplo, 134217728, o bien, 128m.

4.3.2.1 core-site.xml

En *core-site.xml* ajustamos la configuración del *core* de *Hadoop* tales como ajustes de Entrada/Salida que son comunes a *HDFS* y *MapReduce*.

Veamos algunas propiedades interesantes:

fs.defaultFS: URI al sistema de ficheros *HDFS* que designa una máquina como un *NameNode* que resulta imprescindible para ejecutar *HDFS* cuyo:

host: es el nombre de host del *NameNode* o dirección *IP*, y el
puerto: es el puerto por el que el *NameNode* escucha los *RPCs*. Por defecto es 8020.

El Sistema de ficheros por defecto es usado para resolver rutas relativas, lo cual es útil ya que se puede evitar escribir la dirección del *NameNode* y poner, por ejemplo, /a/b, que realmente se resuelve como *HDFS*://*NameNode*/a/b, evitando el conocimiento de la dirección particular de un *NameNode*.

io.files.buffer.size: Tamaño que determina cuántos datos son almacenados en memoria intermedia durante las operaciones de lectura y escritura.

Hadoop usa, como opciones comunes, un tamaño de buffer, de:
•4 KB (4096 bytes),
•64 KB (65536 bytes) or
•128 KB (131072 bytes).

fs.trash.interval: Número de minutos entre cada *trash checkpoints (chequeo de basura)*. Por defecto este valor está a 0 (borra los ficheros sin almacenarlos en el directorio /trash).

Si trash está habilitado, cada usuario tiene su propio directorio *.Trash* en su directorio *HOME*. Cuando un fichero es borrado por un usuario o aplicación, éste no es automáticamente borrado de *HDFS*, sino que es movido al directorio */trash* y mientras permanezca allí podrá ser rápidamente restaurado.

Un fichero permanece en /trash por una cantidad de tiempo configurable, que se concreta en esta propiedad. Después de pasado este tiempo el Name Node borra el fichero del espacio de nombres del *HDFS*, lo cual libera todos los bloques asociados a él.

Como acabamos de ver *HDFS* borrará automáticamente los ficheros en el directorio trash después de un período mínimo, pero otros sistemas de ficheros no lo harán, de forma que tendremos que hacerlo periódicamente, usando desde el shell del sistema de ficheros el siguiente comando:

```
$ hadoop fs -expunge
```

Conocimiento de la topología de Red por parte de Hadoop [14]

Los componentes de *HDFS* y *YARN* son conscientes del *rack*. Pero, ¿Cómo? El ***NameNode*** y el ***ResourceManager*** obtienen información del *rack* de los nodos *worker* en el *cluster* invocando una *API* configurado por el administrador del Sistema. El *API* resuelve el nombre DNS (tambíen dirección *IP*) a un identificador de *rack*.

La configuración de *Hadoop* debe especificar un mapa entre la dirección del nodo y la localización de la red. El mapa es descrito por una interface *Java*, *DNSToSwitchMapping*, cuya signatura es:

```
public interface DNSToSwitchMapping {
    public List<String> resolve(List<String> names);
}
```

donde el parámetro *names* es una lista de direcciones IP y el valor de retorno es una *List* de la localización de red correspondiente.

net.topology.node.*switch*.mapping.impl: Esta propiedad de configuración define una implementación de la interface *DNSToSwitchMapping* que el ***namenode*** y el ***jobtracker*** usan para resolver la localización de red de los nodos worker.

org.apache.hadoop.net.ScriptBasedMapping, es su valor por defecto, y eso implica que para resolver la localización de red invoca a un script que lo puede encontrar usando la propiedad *net.topology.script.file.name*. En caso de que no sea especificada esta localización del script, el comportamiento por defecto es mapear todos los nodos a una localización de red única, llamada */default-rack*.

net.topology.script.file.name: Nombre del script que debería ser invocado para resolver el nombre del DNS al nombre de topología de red.

Configuración en cluster del Aula Anita Borg

En nuestro *cluster*, al fichero *core-site.xml* recién instalado, le hemos añadido las siguientes líneas:

```
<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>
```

Por tanto, el fichero tendría el siguiente aspecto:

```
<?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>
```

Captura 11. core-site.xml.

En el anexo 1, se puede acceder a parte del contenido del fichero *core*-default.xml para que el lector se haga una idea de su contenido.

4.3.2.2 hdfs-site.xml

Ficheros donde se ajustan propiedades para los servicios *HDFS*: *NameNode*, secondary *NameNode*, y *DataNodes*.

Agruparemos, por cuestiones de claridad, las propiedades en dos grupos de configuración. Para el *NameNode y el DataNode*.

Configuración para el namenode

dfs.*NameNode*.name.dir: Especifica una lista de directorios donde el *NameNode* almacena, de forma persistente, los metadatos del sistema de ficheros (*edits* y *fsimage*).

Una copia de cada fichero de metadatos es almacenado en cada directorio por redundancia. Es común configurarlo para que los metadatos del *NameNode* sean escritos a uno o dos discos locales y un disco remoto, guardándonos de un fallo en el disco local y, por tanto, fallo del *NameNode* entero.

El *NameNode* secundario coge solo periodos del checkpoint del *NameNode*, por eso no proporciona una copia de seguridad actualizada del *NameNode*.

dfs.*NameNode*.checkpoint.dir: Ubicación donde el *NameNode* secundario almacena sus *checkpoints* del sistema de ficheros.

Esta propiedad especifica la lista de directorios donde los *checkpoints* son guardados de forma redundante.

Consejo: teniendo en cuenta que los directorios de almacén para *HDFS* están por defecto en directorios temporales es importante configurar esta propiedad y que los directorios temporales no se pierdan al vaciar el sistema.

dfs.web*HDFS*.enabled: Permite Web*HDFS* (REST *API*) en *NameNode*s y *DataNodes*.

Hadoop proporciona librerías nativas para acceder al *HDFS*. Sin embargo, los usuarios prefieren usar *HDFS* de forma remota con el cliente pesado de las librerías nativas. Por ejemplo, algunas aplicaciones necesitan cargar datos en o desde el *cluster*, o externamente interactuar con los datos del *HDFS*.

Web*HDFS* aborda estas cuestiones proporcionando una *API* REST HTTP completamente funcional para acceder *HDFS* [14].

dfs.permissions.enabled: Modelo de permisos de *HDFS* para los archivos y directorios que se parece mucho a POSIX. Si "True" la comprobación de permisos en *HDFS* está habilitada. Si "False", desactivada.

Hay tres tipos de permisos: lectura(r), escritura(w) y ejecución(x) con significados personalizados para ficheros y directorios.

Cada fichero y directorio tiene un propietario, un grupo y un modo. El modo se compone de los permisos para el usuario que es propietario, los permisos para los usuarios que son miembro del grupo y, los permisos para los usuarios que no son ni el propietario ni los miembros del grupo.

La identidad de un cliente es determinado por el nombre de usuario y el grupo del proceso que se está ejecutando.

Vale la pena tener los permisos habilitados para evitar modificaciones accidentales o borrado de partes sustanciales del sistema de ficheros o por usuarios o por herramientas o programas automáticos.

Cuando se habilita la comprobación de permisos, los permisos del propietario se comprueban de la siguiente forma: si el nombre del usuario coincide con el propietario y de los permisos de grupo se comprueba si el usuario es miembro del grupo; de lo contrario se comprueban los demás permisos.

Existe el concepto de super usuario, que es la identidad del proceso *NameNode* para el cual la comprobación de permisos no se realizan para él.

dfs.*NameNode*.handler.count: Número de hilos del servidor *NameNode* para mantener RPCs de un gran número de *DataNodes*.

dfs.*NameNode*.hosts: Nombre de un fichero con una lista de *DataNodes* que están autorizados para conectar al *NameNode*. La ruta completa del fichero debe ser especificada. Si el valor es vacío, todos los hosts están permitidos.

dfs.*NameNode*.hosts.exclude: Nombre de un fichero con una lista de *DataNodes* que no están autorizados para conectar al *NameNode*. La ruta completa del fichero debe ser especificada. Si el valor es vacío, todos los hosts están permitidos.

dfs.blocksize: El tamaño de bloque de *HDFS* es de 64 MB por defecto, pero muchos *cluster* usan 128 MB (134.217.728 Bytes) o incluso 256 MB (268.435.456 bytes)

Configuración para el datanode

dfs.*DataNode*.data.dir: Determina en qué parte del sistema de archivos local un *DataNode* DFS debería almacenar sus bloques. Si hay una lista de directorio separados por comas, los datos se almacenarán en todos ellos, típicamente en distintos dispositivos. Los directorios que no existan serán ignorados.

Consejo: Para un máximo rendimiento, se debería montar los discos de almacenamiento con la opción *noatime*. Esto nos asegura que la información del último acceso no se escribe en los ficheros leídos, lo cual da una ganancia de rendimiento.

dfs.*DataNode*.balance.bandwidthPerSec: Especifica la máxima cantidad de ancho de banda que cada *DataNode* puede utilizar con el propósito del balanceo en términos de Bytes por segundo.

Con el tiempo, la distribución de los bloques en los *DataNodes* puede llegar a no estar balanceado. Que haya nodos con más carga y otros con menos afecta al rendimiento de *MapReduce* y produce una mayor presión a los nodos más utilizados.

El programa balanceador es un servicio de *Hadoop* que redestribuye los bloques moviéndolos de los *DataNodes* más utilizados a los menos.

Para lanzar el balanceador en el *cluster*, ejecutamos el comando start-balancer.sh en el shell de *linux*.

```
$ start-balancer.sh
```

El argumento *-threshOld* especifica el porcentaje umbral que define qué significa para el *cluster* estar balanceado.

El balanceador se ejecuta en segundo plano sin afectar indebidamente al *cluster* ni interferir con otros clientes que usan el *cluster*. Este limita el ancho de banda que usa para copiar los bloques de un nodo a otro con un modesto 1 MB/s, valor por defecto que puede ser ajustado cambiando el valor de esta propiedad.

Configuración en cluster del Aula Anita Borg

En nuestro *cluster*, al fichero *hdfs-site.xml* recién instalado, le hemos añadido las siguientes líneas:

```
<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.blocksize</name>
                <value>128m</value>
        </property>
        <property>
                <name>dfs.webHDFS.enabled</name>
                <value>true</value>
        </property>
        <!-- disable permissions; only for development, of course -->
        <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>
```

Por tanto, el fichero tendría el siguiente aspecto:

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

<!-- Put site-specific property overrides in this file. -->
<configuration>
    <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>
    <!-- desactivado permisos. Sólo para desarrollo. -->
    <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 3. hdfs-site.xml.

En el anexo 2, se puede acceder a parte del contenido del fichero *hdfs-default.xml* para que el lector se haga una idea de su contenido.

4.3.2.3 yarn-site.xml

Fichero que contiene información relacionada con los servicios y propiedades de *YARN*: Resource Manager, Node Manager and History Server.

Agruparemos, por cuestiones de claridad, las propiedades del fichero *YARN*-site.xml en cuatro grupos de configuración. Para el:

- *ResourceManager* y *NodeManager*
- *ResourceManager*
- *NodeManager*
- Historyserver

Configuraciones para ResourceManager y NodeManager

YARN.acl.enable: Habilita las ACL's. Por defecto, está desactivado.

Las listas de control de acceso (*ACL -Access Control Lists*) son listas de usuarios y grupos de usuarios que tienen permiso de acceso o modificación, a una operación específica.

Para poder añadir autorización a objetos en el *cluster*, *YARN* proporciona cuatro tipos de ACLs: *Administration, Service Level, Queue* y *Application*.

YARN.admin.acl: Especifica la ACL que indica quien puede ser administrador del *cluster YARN*. Por defecto tiene el valor * que significa que cualquiera tiene permiso.

Centrándonos en la lista de *administration ACL,* un administrador hace referencia a un usuario que puede realizar cualquier operación en el *cluster YARN*; y solo él puede ejecutar, por ejemplo, el comando *rmadmin,* el cual lanza un cliente *ResourceManager* desde la línea de comandos. Este es usado para refrescar la política de control de acceso, planificación de colas y nodos registrados con *ResourceManager*.

YARN.NodeManager.remote-app-log-dir: Ubicación por defecto en /tmp/logs.

YARN.log-aggregation-enable: Habilita o deshabilita la agregación del log.

•Con esta propiedad hablitada:

-se recoge el registro log de cada *container*, y cuando la aplicación haya finalizado, lo mueve al directorio configurado en la propiedad *yarn.nodemanager.remote-app-log-dir*.

-y las siguientes propiedades son activadas:

yarn.nodemanager.remote-app-log-dir
yarn.nodemanager.remote-app-log-dir-suffix
yarn.log-aggregation.retain-seconds
yarn.log-aggregation.retain-check-interval-seconds
yarn.log.server.url

•Y con la propiedad deshabilitada, cada *NodeManager* guardará el log de forma local y no los agregará, activando las siguientes propiedades.

yarn.nodemanager.log.retain-seconds
yarn.nodemanager.log.deletion-threads-count

Dependiendo del directorio agregado, hay que chequear sus permisos. Por ejemplo, si queremos que sean depositados en /*YARN*/logs:

```
$ su - HDFS -c "Hadoop fs -mkdir -p /YARN/logs"
$ su - HDFS -c "Hadoop fs -chown -R YARN:Hadoop /YARN/logs"
$ su - HDFS -c "Hadoop fs -chmod -R g+rw /YARN/logs
```

Debemos tener en cuenta la siguiente relación de directorios por si se quisiera cambiar el valor por defecto.

Propietario. Usuario:Grupo	Propiedad Configuración	Sistema de Ficheros	Permisos
YARN:Hadoop	$YARN_LOG_DIR	Local	drwxrwxr-x
	YARN.NodeManager.local-dirs		drwxr-xr-x
	YARN.NodeManager.logl-dirs		drwxr-xr-x
	YARN.NodeManager.remote-app-log-dir	HDFS	drwxrwxrwxt

Tabla 10. propiedades Yarn de configuración de directorios y sus permisos.

Configuración en cluster del Aula Anita Borg

En nuestro *cluster*, al fichero *YARN*-site.xml, en el bloque de propiedades correspondientes a los servicios *ResourceManager* y *NodeManager*, le hemos añadido las siguientes líneas:

```
<configuration>
        <!-- For ResourceManager and NodeManager -->
        <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>
```

Por tanto, parte del fichero tendría el siguiente aspecto:

```
<!-- Site specific YARN configuration properties -->

    <!-- For ResourceManager and NodeManager -->
    <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>
```

Imagen 4. yarn-site.xml: propiedades específicas para ResourceManager y NodeManager.

Configuraciones para ResourceManager

Lista resumida de propiedades con los puertos por defecto configurados para los servicios de *YARN* de las que comentaremos algunas.

Servicio		Propiedad	Puerto (por defecto)
ResourceManager	RPC Communication	YARN.ResourceManager.adress	8032
	Web UI	YARN.ResourceManager.webapp.address	8088
	Scheduler	YARN.ResourceManager.scheduler.address	8030
	Resource tracker	YARN.ResourceManager.resource-tracker.address	8031
	Admin	YARN.ResourceManager.admin.address	8033
NodeManager	RPC Communication	YARN.NodeManager.address	8041
	Localizer	YARN.NodeManager.localizer.address	8040
	Web UI	YARN.NodeManager.webapp.address	8042
TimeLineServer	RPC Communication	YARN.timeline-service. address	10200
	Web UI	YARN.timeline-service. webapp.address	8188
	HTTPS	YARN.timeline-service. webapp.https.address	8190

Tabla 11. Resumen de puertos por defecto para los servicios YARN [20].

Tener en cuenta que si configuraramos cualquier propiedad de la Tabla 11, proporcionándole un valor, sobreescribimos con este nuevo valor a las propiedades *yarn.resourcemanager.hostname, yarn.nodemanager.hostname o yarn.timeline-service.hostname.*

YARN.ResourceManager.hostname: Host del *ResourceManager*

Un administrador del *cluster* necesita configurar esta propiedad para definir el *hostname* o IP del nodo *ResourceManager*.

El valor por defecto es 0.0.0.0.

YARN.ResourceManager.address: *ResourceManager* host:port

En el servidor especifica el puerto del gestor de aplicaciones en el *Resource Manager* y en el cliente es usado para conectar con el *Resource Manager*.

Si le damos valor, como hemos comentado, sobreescribe a la propiedad *yarn.resourcemanager.hostname*. Su valor por defecto es: *${yarn.resourcemanager.hostname}:8032*.

YARN.ResourceManager.scheduler.address: *ResourceManager* host:port

Usado por la aplicación *Master*, por ejemplo *MapReduce*, para obtener recursos del *ResourceManager*.

Valor por defecto: *${yarn.resourcemanager.hostname}:8030*.

YARN.ResourceManager.resource-tracker.address *ResourceManager* host:port

Node Manager la usa para comunicarse con *Resource Manager*.

Valor por defecto: *${yarn.resourcemanager.hostname}:8031*.

YARN.ResourceManager.admin.address *ResourceManager* host:port

Dirección de la interfaz administrativa del *ResourceManager*, utilizado por clientes administrativos, como por ejemplo, *rmadmin* para comunicar con *Resource Manager*.

Valor por defecto: *${yarn.resourcemanager.hostname}:8033*.

YARN.ResourceManager.webapp.address *ResourceManager* host:port

El *API* del *ResourceManager YARN* permite al usuario o administrador obtener métricas, lista de *NodeManagers*, información de planificación, aplicaciones asociadas, etc.

Valor por defecto: *${yarn.resourcemanager.hostname}:8088*.

YARN.ResourceManager.scheduler.class: Clase utilizada en planificación de recursos.

La planificación *YARN* consiste en la asignación de recursos del *cluster* a aplicaciones que se están ejecutando en el *cluster*.

La planificación es uno de los componentes principales del servicio *ResourceManager* y es el responsable de la asignación de recursos basado en los requerimientos de recursos de los *container*.

Los procesadores y la RAM son dos recursos críticos para la ejecución de cada *container*. Con un número creciente de *container* concurrentes con sus requerimientos de recursos satisfechas, la gestión de recursos en el *cluster* puede llegar a ser crítica para la ejecución exitosa de las aplicaciones.

YARN define dos tipos de planificadores predefinidos: *capacity* y *fair schedulers*), que dividen los recursos:

•*Capacity scheduler:* como un porcentaje del total de recursos del *cluster y* permite:
-la ejecución de múltiples aplicaciones compartiendo recursos del *cluster*.
-multi-tenencia y capacidad garantizada.
-Uso colas *CSQueue*[25] para la definición de colas.

•*Fair scheduler:* como una proporción justa de recursos a todas las aplicaciones que se están ejecuntando en el *cluster*.

Cuando una sola aplicación es enviada al *cluster*, todos los recursos están disponibles para esta. Cuando otra aplicación es enviada al *cluster*, una parte de los recursos es dado a la segunda aplicación.

YARN.scheduler.minimum-allocation-mb Límite mínimo, en MB, de memoria asignada a cada *container* solicitado al *Resource Manager*.

Un *container* solicitado más pequeño sería asignado a un contenedor con este tamaño.

El valor por defecto es 1024 MB.

YARN.scheduler.maximum-allocation-mb Límite máximo, en MB, de memoria asignada a cada *container* proporcionado al *Resource Manager. V*

El valor por defecto es 8192 MB

YARN.ResourceManager.nodes.include-path: Lista *NodeManagers* permitidos.

YARN.ResourceManager.nodes.exclude-path: Lista de *NodeManagers* excluídos.

[25] interface que representa una estrucutra de colas para un nodo en un árbol de jerarquía de colas para *CapacityScheduler*.

Dentro de los componentes del *ResourceManager* que interactuan con los *NodeManagers* está el *gestor de lista de nodos,* una colección en la memoria del *ResourceManager* de nodos válidos y excluídos.

El gestor de lista de nodos es el responsable de leer el fichero de configuración de los *hosts* especificados en las propiedades de configuración *yarn.resourcemanager.nodes.include-path* y *yarn.resourcemanager.nodes.exclude-path* que alimentan la lista inicial de nodos.

También guarda el seguimiento de los nodos que son explicitamente decomisionados por los administradores.

Configuración en cluster del Aula Anita Borg

En nuestro *cluster*, al fichero *YARN-site.xml*, en el bloque de propiedades correspondientes al servicio *ResourceManager*, le hemos añadido las siguientes líneas:

```
<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>
```

Por tanto, parte del fichero tendría el siguiente aspecto:

```
<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>
```

Imagen 5, yarn-site.xml: propiedades específicas para ResourceManager.

Configuraciones para NodeManager

YARN.NodeManager.resource.memory-mb: Cantidad de memoria física (RAM), en MB, en el nodo de cómputo para los *containers*.

Es importante que este valor no sea el total de RAM del nodo, ya que los servicios de *Hadoop* también requieren memoria *RAM*. El valor por defecto es 8192 MB.

YARN.NodeManager.vmem-pmem-ratio: Máximo ratio por el cual el uso de la memoria virtual de tareas puede exceder la memoria física.

Calculamos la cantidad de memoria virtual que le es permitida a cada *container*, con la fórmula: *containerMemoryRequest*vmem-pmem-ratio*.

El valor por defecto es 2.1

YARN.NodeManager.disk-health-checker.min-healthy-disks: Mínima fracción del número de discos que deben estar en buen estado de funcionamiento para que *nodemanager* lance nuevos contenedores.

Se corresponde al espacio de las propiedades *yarn-nodemanager.local-dirs* y *yarn.nodemanager.log-dirs*.

Si hay menos número de directorios disponibles, entonces los *containers* nuevos no se lanzarán en este nodo.

YARN.NodeManager.local-dirs: Lista de directorios separada por comas del sistema de ficheros local donde datos intermedios son guardados temporalmente.

En esta propiedad queda definido el raíz de cada sistema de ficheros local de cada *container*. Tomando como base este directorio, el *NodeManager* creará subdirectorios donde cada *container* tendrá su propio sistema de ficheros.

Esta estructura será borrada una vez que el *container* pare; no obstante puede ser configurado para labores de supervisón o depuración de fallos a través de la propiedad *yarn.nodemanager.delete.debug-delay-sec,* que veremos posteriormente.

Valor por defecto: *${hadoop.tmp.dir}/nm-local-dir*

YARN.NodeManager.log-dirs Lista de directorios separada por comas del sistema de ficheros local donde los logs de los contenedores son guardados temporalmente.

Valor por defecto: ${*YARN*.log.dir}/userlogs

La *ApplicationMaster* así como los *containers* se ejecutan en el *cluster* y generan logs de aplicación, cuyo estudio nos permite analizarlas y depurar fallos. Estos logs son generados bajo el directorio por defecto de esta propiedad que podemos configurar.

A una aplicación que se ejecuta sobre *YARN* se le proporciona un *ID* único: {appid}. Los *ID's* de los *containers* son derivados del *ID* de la aplicación añadiendo el número de *container* a éste: {$*containerId*}.

Los logs para una aplicación son generados en el directorio con un *ID* de aplicación: *{yarn.nodemanager.log-dirs}/application_${appid}*.

Este directorio contiene directorios de los contenedores cuyo nombre es su *ID* de *container*: /*container*_{$containerId} .

Cada *container* genera tres tipos de ficheros. Contienen:
•***stderr*:** errores que ocurren durante la ejecución de un *container* particular.
•syslog**:** tres tipos de mensajes de log: INFO, WARN ERROR y FATAL.
• stdout**:** mensajes impresos del flujo de ejecución del *container*.

YARN.NodeManager.delete.debug-delay-sec: Número de segundos después de que una aplicación termine y el servicio de borrado del *nodemanager* borre la raiz de directorios de trabajo y del log de las aplicaciones *YARN*.

Para diagnosticar problemas de la aplicación *YARN*, un valor de 600 s (10 minutos) le podría permitir al administrador del sistema o usuario examinar estos directorios y en función de sus necesidades ajustarlo posteriormente.

YARN.NodeManager.log.retain-seconds: Tiempo en segundos para retener el log de usuario de los *nodemanagers* solo aplicable si la agregación del log está deshabilitado. Por defecto, su valor es 10800.

YARN.log-aggregation.retain-check-interval-seconds: Determina cuánto tiempo esperar, en segundos, entre chequeos de retención de logs agregados.

Con valor 0 ó negativos, el valor es computado a una décima parte del tiempo de retención del log agregado. Evitar poner valores muy bajos, porque supondría bombardear al D*ataNode*. El valor por defecto es -1.

YARN.NodeManager.remote-app-log-dir: Directorio donde los *nodemanagers* agregan los logs. Valor por defecto: /tmp/logs.

Su localización no debería estar en un sistema de fichero local, para permitir agregar logs a todos los *NodeManager* y permitir al servicio history server poder mostrarlo. Una buena ubicación es en *HDFS*.

YARN.NodeManager.aux-services: *MapReduce*_shuffle

YARN proporciona una opción para configurar servicios *AUX* usados durante la ejecución de aplicaciones. Un administrador necesita definir el servicio *shuffle* de *MapReduce* como un servicio *AUX* para el *cluster YARN*.

Con la propiedad ajustada al valor *mapreduce_shuffle*, activa el servicio *shuffle* que es, por tanto, un servicio auxilar de larga ejecución en el *cluster YARN* que se ejecuta en los *NodeManagers* y proporciona ordenación del tipo *MapReduce*.

El servicio *Shuffle* necesita estar habilitado para las aplicaciones que usen el paradigma *MapReduce*.

YARN.NodeManager.auxservices.*MapReduce*_shuffle.class: nombre de clase

Después de decirle en la propiedad anterior *yarn.nodemanager.aux-services* que implementen el servicio *shuffle* mediante el valor *mapreduce_shuffle* , le damos el medio para hacerlo, proporcionándole una clase, dándole el valor: *org.apache.hadoop.mapred.ShuffleHandler,* a esta propiedad.

Configuración en cluster del Aula Anita Borg

En nuestro *cluster*, al fichero *yarn-site.xml*, en el bloque de propiedades correspondientes al servicio *NodeManager*, le hemos añadido:

```
<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>
```

Por tanto, parte del fichero tendría el siguiente aspecto:

```
<!-- For NodeManager -->
<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>
```

Imagen 6. yarn-site.xml: propiedades específicas para NodeManager .

Configuraciones para historyserver

YARN.log-aggregation.retain-seconds: Cuánto tiempo, en segundos, esperar para borrar los logs agregados.

Con valor –1 u otros valores negativos deshabilita el borrado de los logs agregados. Evitar poner valores demasiados bajos, porque supondría sobrecargar al *NameNode*.

YARN.log-aggregation.retain-check-interval-seconds: Cuánto tiempo esperar entre chequeos de retención de logs agregados.

Con valor 0 ó negativos, el valor es computado como una décima parte del tiempo de retención del log agregado. Evitar poner valores demasiados bajos, porque supondría sobrecargar al *NameNode*.

Configuración en cluster del Aula Anita Borg

En nuestro *cluster* de aula Anita Borg, las propiedades del fichero *yarn-site.xml*, correspondientes al servicio *HistoryServer sería:*

```
<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>node2</value>
</property>
</configuration>
```

Por tanto, parte del fichero tendría el siguiente aspecto:

```
            <!-- For HistoryServer -->
            <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>node2</value>
            </property>
        </configuration>
```

Imagen 7. yarn-site.xml: propiedades específicas para HistoryServer.

En el anexo 3, se puede acceder a parte del contenido del fichero *YARN*-default.xml para que el lector se haga una idea de su contenido.

4.3.2.4 mapred-site.xml

Contiene información relacionada con el entorno *MapReduce* en el *cluster* que ajusta la configuración de los servicios *MapReduce*: *jobtracker* y *tasktrackers*.

Agruparemos, por cuestiones de claridad, las propiedades en dos grupos de configuración. Para:

•Aplicaciones *MapReduce*
• JobHistoryServer

Configuraciones para aplicaciones MapReduce

MapReduce.Framework.name: *YARN*

Define el entorno de ejecución para lanzar trabajos *MapReduce*. Los valores pueden ser local, clásico o *YARN*..

•*yarn*: si se define este entorno, hay que definir también un host para el servicio *ResourceManager* configurando la propiedad *yarn.resourcemanager.hostname* de *yarn-site.xml*.

•*local (*valor por defecto*)*: si las aplicaciones *MapReduce* se ejecutan localmente en un proceso sobre una máquina cliente sin usar ningún recurso del *cluster*.

Este proceso local ejecutará las tareas *map* y las tareas *reduce* para un trabajo dado y, no necesita barajar datos desde la salida de la tarea *map* en un servidor a la entrada de la tarea *reduce* de otro servidor.

•*clásico*: Esta propiedad existe para dar cabida a situaciones imprevistas donde hay problemas de compatibilidad con versiones anteriores con trabajos de *MapReduce* de los usuarios y la necesidad de un envío de trabajos a un *JobTracker* es inevitable.

MapReduce.jobtracker.address host:puerto donde el trabajo de *MapReduce* se ejecuta y el *jobtracker* escucha.

Para ejecutar *MapReduce*, se necesita designar una máquina como *jobtracker*, la cual en un *cluster* no muy grande puede estar en la misma máquina que el *namenode*. El puerto 8021 es una elección común.

MapReduce.tasktracker.*map*.tasks.maximum: Máximo número de tareas *MAP* que se ejecutarán de forma simultánea en cada tarea *tracker*. Valor por defecto, 2.

MapReduce.tasktracker.*reduce*.tasks.maximum: Máximo número de tareas *REDUCE* que se ejecutarán de forma simultánea en cada tarea *tracker*. Valor por defecto, 2.

MapReduce.tasktracker.report.address: Interface y puerto donde el servidor de tareas *tracker* escucha. Debería ser cambiado su valor solo si tu host no tiene configurada la interface *loopback*[26].

[26] interfaz de red virtual, en el rango de direcciones '127.0.0.0/8' que se suele utilizar cuando una transmisión de datos tiene como destino el propio host y en tareas de diagnóstico de conectividad y validez del protocolo de comunicación.

MapReduce.jobtracker.system.dir: Directorio donde *MapReduce* almacena ficheros de control.

MapReduce usa un sistema de ficheros distribuido para compartir ficheros con los *tasktrackers* que ejecutan las tareas *MapReduce*. Esta propiedad nos dice el directorio donde estos ficheros pueden ser almacenados y que se construye de forma relativa al sistema de fichero por defecto configurado en la propiedad *fs.defaultFS,* que usualmente está en *HDFS*.

MapReduce.cluster.local.dir: Directorio local donde *MapReduce* almacena ficheros de datos intermedios.

Durante un trabajo *MapReduce* los datos intermedios y ficheros de trabajo son escritos en ficheros locales temporales. Ya que los datos tienen la posibilidad de tener grandes salidas de las tareas de *MAP* es necesario asegurar que las particiones de disco soporten este volumen.

MapReduce.task.io.sort.mb: La cantidad total o límite superior de buffer de memoria en MB para usar mientras ordena ficheros. Valor por defecto 100 MB.

MapReduce.task.io.sort.factor: Número de flujos a fusionar a la vez mientras se ordena el fichero. Esto determina el numero de identificadores de ficheros abiertos. Defecto: 10

MapReduce.map.memory.mb: Límite de recurso de memoria para *MAP*.

Cantidad de memoria a solicitar desde el planificador de cada tarea de *MAP*. Valor por defecto 1024.

MapReduce.map.Java.opts: Límite de tamaño de *Heap* en máquinas virtuales de child para *MAP*. Valor por defecto: -Xmx200m.

MapReduce.reduce.memory.mb: Límite de recurso de memoria para *REDUCE*

Cantidad de memoria requerida desde el planificador para cada tarea *REDUCE*. Valor por defecto, 1024.

MapReduce.reduce.Java.opts: Límite de tamaño de *Heap* en máquinas virtuales de child para *REDUCE*.

MapReduce.reduce.shuffle.parallelcopies: Máximo número de copias paralelas ejecutadas por *reduce* para extraer salidas de una gran catidad de *maps*. Es decir, número de transferencias paralelas ejecutadas por *Reduce* durante la fase (*shuffle*) de copia. Por defecto 5.

Configuración en cluster del Aula Anita Borg

En nuestro *cluster*, al fichero *mapred-site.xml*, en el bloque de propiedades correspondientes a las aplicaciones *MapReduce*, le hemos añadido:

```
<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>
```

Por tanto, el fichero tendría el siguiente aspecto:

```
<configuration>
    <!-- For MapReduce Applications -->
    <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>
```

Imagen 8. mapred-site.xml: propiedades específicas para aplicaciones MapReduce.

Configuraciones para el Servidor JobHistoryServer

HistorySever mantiene información sobre aplicaciones *MapReduce* ejecutadas sobre el *cluster*. La *API Rest* [27] proporciona contadores, información de configuración de trabajos y tareas.

MapReduce.jobhistory.address: Host y puerto donde la aplicación *MapReduce* enviará el histórico del trabajo vía su propio protocolo interno. Puerto por defecto 10020.

MapReduce.jobhistory.webapp.address: Dirección *MapReduce HistoryServer* donde un administrador o usuario puede ver los detalles de los trabajos de *MapReduce* que se han completado. Puerto por defecto: 19888.

MapReduce.jobhistory.intermediate-done-dir : Directorio donde los archivos históricos son escritos por los trabajos de *MapReduce*.

MapReduce.jobhistory.done-dir: Directorio donde los archivos históricos son gestionados por el *Servidor JobHistory MR*.

YARN.app.*MapReduce*.am.staging-dir: Cuando un trabajo *MapReduce* es enviado a *YARN*, la *ApplicationMaster* de *MapReduce* creará temporalmente datos en *HDFS* para el trabajo y necesitará un *staging area*[28] para sus datos. Este área también será usada por el trabajo del *history server*.

Configuración en cluster del Aula Anita Borg

En nuestro *cluster*, al fichero *mapred-site.xml*, en el bloque de propiedades correspondientes a *MapReduce JobHistory Server*, le hemos añadido las siguientes líneas:

```
<!-- For MapReduce JobHistory Server -->
  <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>
```

[27] API (Representational State Transfer) es una interfaz entre sistemas que utiliza HTTP para obtener datos o indicar la ejecución de operaciones sobre los datos.

[28] Directorio donde reunir todos los ficheros y carpetas necesarios para el proyecto.

Por tanto, parte del fichero tendría el siguiente aspecto:

```
    <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 9. mapred-site.xml: propiedades específicas para MapReduce JobHistory Server.

En el anexo 4, se puede acceder a parte del contenido del fichero *mapred-default.xml* para que el lector se haga una idea de su contenido.

4.3.3 Logging

Hadoop usa log4j de Apache para mostrar mensajes de información.

Es una librería adicional de *Java* que permite a nuestra aplicación mostrar mensajes de información de lo que está sucediendo en ella. Es lo que habitualmente se conoce como un log. Tiene la ventaja de ser dinámico y, por tanto, configurable.

Para personalizar la configuración de logs de los servicios de *Hadoop* habría que editar el fichero *conf/log4j.properties*.

En nuestro *cluster* del Aula Anita Borg hemos dejado el fichero con los valores por defecto.

4.3.4 Slaves file

Normalmente se elige una máquina para que actúe como *NameNode* (en nuestro caso EIP6029) y otra para que actúe como *ResourceManager* (en nuestro caso EIP6001), exclusivamente. El resto de máquinas actúan de *DataNode* y el *NameNode* se referiere a ellas como *slaves*, esclavas.

Figura x: Roles *Hadoop* de los equipos en el Aula Anita Borg

En el fichero situado en `conf/slaves` se listan todos los *hostnames* o direcciones IP, uno por línea, de los *slaves*. En total, 30 equipos.

```
158.42.215.14
158.42.215.28
158.42.215.29
158.42.215.197
158.42.215.221
158.42.215.228
158.42.215.30
158.42.215.49
158.42.215.60
158.42.215.39
158.42.215.56
158.42.215.59
158.42.215.47
158.42.215.57
158.42.215.61
158.42.215.48
158.42.215.58
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
158.42.215.68
```

Imagen 10. Fichero slaves de Hadoop.

4.3.5 Pasos para la configuración

Los pasos para la configuración serían:

1.-Acceder al equipo EIP6029:

2.- Editar con la configuración que se haya diseñado, los ficheros de:
⇨variables de entorno de los servicios (*.sh):

hadoop-env.sh
yarn-env.sh
mapred-env.sh

⇨propiedades de los servicios(*.xml):

core-site.xml
hdfs-site.xml
yarn-site.xml
mapred-site.xml

3.-Crear los siguientes directorios y configurarlos en los ficheros anteriores:

```
mkdir "${HOME}/cluster/var/Hadoop"
mkdir "${HOME}/cluster/var/Hadoop/Hadoop-DataNode"
mkdir "${HOME}/cluster/var/Hadoop/Hadoop-NameNode"
mkdir "${HOME}/cluster/var/Hadoop/mr-history"
mkdir "${HOME}/cluster/var/Hadoop/mr-history/done"
mkdir "${HOME}/cluster/var/Hadoop/mr-history/tmp"
```

4.- Crear el fichero *slaves*, con el contenido de un nombre de equipo o dirección *IP* de los esclavos, por línea.

```
~/cluster/opt/hadoop/etc/Hadoop/slaves
```

5.- Configurar variables de entorno:

a.-Crear el fichero ~/.*cluster*_rc, y añadir 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 JAVA_HOME="/opt/jdk1.8.0_25"
```

Nota: *HADOOP*_PREFIX y *SPARK*_PREFIX deben adecuarse a la instalación.

b.-Añadir a los ficheros ~/.bashrc y ~/.profile del usuario, la siguiente línea:

```
. ~/.cluster_rc
```

4.4 Configuración de Apache Spark

Apache *Spark* es una plataforma de *cluster* computing diseñado para ser rápido y de propósito general.

En este apartado, vermos las tres formas de *Spark* de configurar el sistema: ***propiedades de Spark***, ***variables de entorno*** y **logging** y, una breve guía de *tunning*[29] donde se nos ofrecen buenas prácticas para optimizar el rendimiento y el uso de la memoria.

4.4.1 Propiedades de Spark

Las propiedades de *Spark* controlan la mayoría de los parámetros de la aplicación. Veamos cómo especificar de 3 formas qué propiedades ponerle a una aplicación:

1.-De forma estática, en el código fuente, pasando al objeto *Spark*Context de tu aplicación, el *objeto* ***SparkConf***.

Por ejemplo, mostramos un fragmento de código de una aplicación python donde *SparkConf* configura algunas propiedades con pares clave-valor a través del método set().

```
val conf = new SparkConf()
        .setMaster("local[2]")
        .setAppName("CountingSheep")
val sc = new SparkContext(conf)
```

2.-Dinámicamente, facilitando los valores a las *propiedades de Spark* en tiempo de ejecución, mediante *flag*s (*flag --conf* o *flag*s especiales) en la llamada, en la línea de comandos, de ***spark-submit*** o el shell de *Spark*.

[29] Afinar la configuración de hardware y software para optimizar su rendimiento.

Por ejemplo, mostramos una forma de lanzar una aplicación al *cluster:*

```
../bin/Spark-submit --name "My app" --master local[4] --conf
Spark.shuffle.spill=false --conf "Spark.executor.extraJavaOptions=-
XX:+PrintGCDetails -XX:+PrintGCTimeStamps" myApp.jar
```

Para mostrar una lista de todas las opciones posibles, ejecuta:

```
$ ./bin/Spark-submit --help
```

3.-Dinámicamente, leyendo *bin/spark-submit* las propiedades de *Spark* de *conf/**spark-defaults.conf*** , en el cual cada línea contiene una clave y un valor separado por espacios en blanco.

```
Spark.master            Spark://5.6.7.8:7077
Spark.executor.memory   4g
Spark.eventLog.enabled  true
Spark.serializer        org.apache.Spark.serializer.KryoSerializer
```

Hay que tener en cuenta que en el código fuente de las opciones anteriores 2 y 3, se crearía un *Spark*Conf vacío:

```
val sc = new SparkContext(new SparkConf())
```

No obstante, los valores tanto como si son especificados con *flag*s o en el fichero de propiedades, serán finalmente pasados a la aplicación y fusionados con estos a través de *Spark*Conf.

Las propiedades puestas directamente en el *SparkConf* toman la más alta precedencia, luego los *flag*s pasado a *Spark*-submit o *Spark*-shell, luego las opciones en el fichero *defaults.conf*.

Una vez lanzada la aplicación al *cluster Spark* podemos ver en la aplicación *web* UI, en *http://<driver>:4040*, accediento a la pestaña "*Environment*" la lista de propiedades *Spark* que hemos configurado explícitamente a través *spark-defaults.conf, SparkConf*, o la línea de comandos. Para las demás propiedades de configuración, puedes aumir que el valor por defecto es el usado.

Existen numerosas propiedades de *Spark* a configurar. En el Anexo pondremos una representación de ellas.

Están divididas en los siguientes grupos: *Application Properties, Runtime Environment, Shuffle Behavior, Spark UI, Compression and Serialization, Execution Behavior, Networking, Scheduling, Dynamic Allocation, Security, Encryption, Spark Streaming, SparkR*.

Mostraremos algunas de ella por su interés en la configuración del *cluster* y para su estudio en la mejora del tiempo de ejecución de nuestra aplicación.

Las unidades de tiempo y almacenamiento que observaremos en las Tablas de las propiedades de *Spark* son:

Unidades de tiempo	descripción
ms	milliseconds
s	seconds
m	Minutes
h	hour
d	days
y	Years

Tabla 12. Unidades de tiempo mostradas en las tablas de las propiedades de Spark.

Unidades de Capacidad	descripción
1b	1 byte = 8 bits
1k o 1kb	1 kilobyte = 1024 bytes
1m or 1mb	1 megabyte = 1024 kilobytes
1g or 1gb	1 gigabyte = 1024 mebibytes
1t or 1tb	1 terabyte= 1024 gigabytes
1p or 1pb	1 petabyte = 1024 tebibytes

Tabla 13. unidades de capacidad mostradas en las tablas de las propiedades de Spark.

Grupo: Application Properties.		
Nombre de la Propiedad	Valor por defecto	Significado
Spark.app.name	(none)	El nombre de la aplicación. Aparecerá en la *WEB* IU y en el log de datos.
Spark.driver.*cores*	1	Número de *cores* a usar por el proceso driver. (sólo válido en modo *cluster*).
Spark.driver.maxResultSize	1g	Límite del tamaño total de resultados serializados de todas las particiones para cada acción *Spark*. Por ejemplo en *collect,* debería ser al menos 1M, o 0 sin límite. Los trabajos serán cancelados si el tamaño total pasa de este límite. Tomar un límite alto puede causar errores *out-of-memory* en el driver (depende de *spark.driver.memory* y la sobrecarga de la memoria de los objetos en *JVM*). Establecer un límite apropiado puede proteger al driver de errores out-of-memory.
Spark.driver.memory	1g	Cantidad de memoria a usar para el proceso driver. (En modo cliente, consultar)
Spark.executor.memory	1g	Cantidad de memoria a usar por *executor*. (Por ejemplo, 2g, 8g).
Spark.lo*GC*onf	false	Registar *Spark*Conf para todos los servicos como INFO cuando inicie *Spark*Context
Spark.master	(none)	Conexión al *cluster* manager. *La lista de URLs master permitidas: Ver apartado 3.2.*
Grupo: Runtime Environment		
Nombre de la Propiedad	Valor por defecto	Significado
Spark.driver.extra*Java*Options	(none)	Un string de opciones extra para la máquina virtual de *Java* a pasar al driver.
Spark.executor.extra*Java*Options	(none)	Un string de opciones extra para la máquina virtual de *Java* a pasar al *executor*.
Spark.python.worker.memory	512m	Cantidad de memoria a usar por los procesos python en los worker durante la aggregation (ej. 512m, 2g). Si durane la agregación sobrepasa esta cantidad, volcará datos a disco.
Grupo:Shuffle Behavior		
Nombre de la Propiedad	Valor por defecto	Significado
Spark.shuffle.memoryFraction	0.2	Fración del tamaño de *Heap* de *Java* para usar en *aggregation* and *cogroups* durante la operación shuffles, si *spark.shuffle.spill* es *True*. En un tiempo dado, el tamaño colectivo total usado para la operación shuffles está limitado por este límite, más allá del cual, el contenido comenzará a cachear a disco. Si el cacheo se produce muy a menudo, puede aumentar este valor a expensas de *spark.storage.memoryFraction*.
Spark.shuffle.sort.bypassMergeThresh*Old*	200	(Avanzado) En la gestión de la operación Shuffle basados en la ordenación, evitar los datos merge-sorting data si no hay agregación en el lado *map* y hay en la mayoría de estos muchas particiones *Reduce*.

Grupo: Compression and Serialization		
Nombre de la Propiedad	Valor por defecto	Significado
Spark.broadcast.compress	true	Si desea comprimir las variables *broadcast* antes de enviarlas. Generalmente una buena idea.
Spark.io.compression.codec	snappy	codec usado para comprimir datos internos como particiones RDD, variables *broadcast* y salidas shuffle. Por defecto, *Spark* provee tres codecs: lz4, lzf, and snappy. Puedes también usar nombres de clases para especificar el codec: *org.apache.spark.io.LZ4CompressionCodec, org.apache.spark.io.LZFCompressionCodec, org.apache.spark.io.SnappyCompressionCodec.*
Spark.io.compression.lz4.blockSize	32k	Tamaño de bloque usado en la comprensión LZ4, en el caso de que esta sea la comprensión configurada en el sistema. Bajar de este tamaño de bloque también bajará el uso de la memoria Shuffle cuando LZ4 sea usado.
Spark.io.compression.snappy.blockSize	32k	Tamaño de bloque usado en la comprensión LZ4, en el caso de que esta sea la comprensión configurada en el sistema. Bajar de este tamaño de bloque también bajará el uso de la memoria snappy cuando LZ4 sea usado.
Grupo: Execution Behavior		
Nombre de la Propiedad	Valor por defecto	Significado
Spark.storage.memoryFraction	0.6	Fracción del *Heap* de *Java* para usar la memoria caché de *Spark*.
Spark.broadcast.blockSize	4m	Tamaño de cada pieza de un bloque para *TorrentBroadcastFactory*. Valores demasiado grandes decrementan el paralelismo durante el *broadcast* y lo hacen más lento.; sin embargo, si es demasiado pequeño podría afectar al rendimiento del BlockManager.
Spark.broadcast.factory	org.apache.*Spark.broadcast*.Torrent*BroadcastF*actory	Qué implementaión *broadcast* usar.
Grupo: Networking		
Nombre de la Propiedad	Valor por defecto	Significado
Spark.akka.frameSize	128	Máximo tamaño de mensaje para permitir comunicación en "control plane"; generalmente sólo se aplica para mapear la información del tamaño de salida enviado entre *executor* y driver. Incrementa este si tu estás ejecutando trabajos con muchos miles de tareas *map* y *reduce* y ves mensajes acerca del tamaño del frame.
Spark.akka.threads	4	Número de hilos para usar en las comunicaciones. Puede ser útil para incrementar en grandes *cluster*s cuando el driver tiene muchos *core*s.

Tabla 14. Propiedades de Spark.

4.4.2 Variables de entorno

Ciertos ajustes de *Spark* pueden ser configurados a través de estas variables que pueden ser usadas para establecer la configuración de cada máquina, a través del script */conf/Spark-env.sh* del directorio donde *Spark* está instalado en cada nodo; como por ejemplo, la dirección *IP*.

En los modos *Standalone* y *Mesos*, este fichero puede dar información específica de la máquina tal y como el *hostname*.

Nota: *conf/spark-env.sh* no existe por defecto cuando *Spark* está instalado. Sin embargo, tu puedes obtenerlo a partir *conf/spark-env.sh.template* para después personalizarlo.

Las siguientes variables pueden ser puestas en *spark-env.sh*:

Variable entorno	Significado
JAVA_HOME	Localización de la instalación de *Java*
PY*SPARK*_PYTHON	Ejecutable binario de Python para usar con Py*Spark* tanto en driver como *workers* (defecto: "python").
PY*SPARK*_DRIVER_PYTHON	Ejecutable binario de Python para usar con Py*Spark* en driver (defecto: PY*SPARK*_PYTHON).
*SPARK*R_DRIVER_R	Ejecutable binario de R para usar con *Spark*R shell (defecto: R).
*SPARK*_LOCAL_IP	Dirección IP de enlace a la máquina.
*SPARK*_PUBLIC_DNS	*Hostname* público desde donde se ejecuta tu programa *SPARK*.

Tabla 15. Variables de Entorno de Spark.

Además, hay otras opciones específicas de un *cluster* Standalone:

Variable entorno	Significado
*SPARK_MASTER*_IP	Enlaza el *master* a una dirección IP específica.
*SPARK_MASTER*_PORT	Puerto de escucha del *master* (defecto: 7077).
*SPARK_MASTER*_WEBUI_PORT	Puerto de escucha de la Interfaz de Usuario *WEB* del *master* (defecto: 8080).
*SPARK_MASTER*_OPTS	Porpiedades de configuración para el *master* "-Dx=y" (default: none). La lista de las posibles opciones: Ver documentación oficial.
*SPARK*_LOCAL_DIRS	Directorio para usar el espacio "scratch" en *Spark*, incluyendo el fichero de salida de *map* y *RDDs* que se almacenan en disco. Debería estar ubicado en un disco local rápido en tu sistema. Pueden haber varios directorios separados por coma y en diferentes discos.
*SPARK*_WORKER_*CORES*	Número total de *cores* para permitir a las aplicaciones *Spark* usarlas en la máquina (defecto: todos los *cores* disponibles).
*SPARK*_WORKER_MEMORY	Cantidad total de memoria para uso de las aplicaciones de *Spark* en la máquina. (ej. 1000m, 2g). Tener en cuenta que la memoria de cada aplicación individual es configurada usando su propiedad *spark.executor.memory*.
*SPARK*_WORKER_PORT	Puerto de escucha del worker *Spark* (default: aleatorio).
*SPARK*_WORKER_WEBUI_PORT	Puerto de escucha de la *WEB* UIdel worker (default: 8081).
*SPARK*_WORKER_INSTANCES	Número de instancias del worker para ejecutar en cada máquina (defecto:1)
*SPARK*_WORKER_DIR	Directorio para ejecutar aplicaciones, las cuales incluirán espacio logs y scratch. (defecto: *SPARK_HOME*/work).
*SPARK*_WORKER_OPTS	Propiedades de configuración que se aplican a los worker en la forma "-Dx=y" (default: none). La lista de las posibles opciones: ver la documentación oficial.
*SPARK*_DAEMON_MEMORY	Memoria para asignar al *master* de *Spark* y los servicios de los *workers* (defecto: 1g).
*SPARK*_DAEMON_*JAVA*_OPTS	Opciones *JVM* para el *master Spark* y los servicios worker en la forma "-Dx=y" (defecto: none).
*SPARK*_PUBLIC_DNS	Nombre de la DNS pública del *master* de *Spark master* y los *workers* (defecto: none).

Tabla 16. Variables de Entorno de Spark para cluster Standalone.

4.4.3 Logging

Spark, como Hadoop, usa log4j para registro de logs como hemos comentado anteriormente en la sección 4.3.3.

En este caso sí que hemos hecho un ajuste para que Spark no nos escriba tanta información. Hemos editado la línea "*log4j.root category= INFO*" cambiando *INFO* por *WARN* en el fichero ~/cluster*/opt/spark/conf/log4j.properties*

4.4.4 Tunning (Ajustes)

Los programas de *Spark* pueden ser cuellos de botella por cualquier recurso en el *cluster*: *CPU*, banda ancha de red o memoria.

Veremos 3 aspectos para la mejora del rendimiento del sistema:
A.-Serialización de datos,
B.-Memory *tunning* y
C.-Otros aspectos.

4.4.4.1 Serialización de datos

Juega un papel importante en el rendimiento de algunas aplicaciones distribuidas.
Formatos que son lentos para serializar objetos o que consumen un gran número de bytes, ralentizarán considerablemente la computación, y a menudo, serán los primeros en tunearse para optimizar la aplicación *Spark*.

Spark tiene como objetivo lograr un equilibrio entre la comodidad de permitirte trabajar con cualquier tipo de clase *Java* en tus operaciones y el rendimiento, y para ello proporciona dos librerías de serialización:

•*Serializacion Java* (por defecto): La serialización de *Java* es flexible pero a menudo bastante lenta y conduce a grandes formatos serializados de muchas clases.

•*Serialization Kryo*: *Kryo* es significativamente más rápido y más compacto que las serialización de *Java* (sobre unas 10 veces), pero no soporta tipos serializables y requiere que registres las clases que usarás en el programa.

4.4.4.2 Memory tunning

Hay tres consideraciones en el uso de *memory tunning:*
•la cantidad de memoria usada por tus objetos: puedes querer ajustar a memoria el *dataset* entero.
•el coste de acceder a estos objetos
•el sobrecoste de *garbage collection*, si existe una alta rotación en términos de objetos.

Por defecto los objetos de *Java* son rápidos de acceder pero consumen un factor de x2.5 más de espacio que los datos en *bruto* dentro de sus campos. Esto es debido a varias razones, por ejemplo, por citar un par de ellas:
•Cada objeto *JAVA* tiene un objeto cabecera de unos 16 Bytes y contiene información tal como un puntero a su clase.
•Los strings de *Java* tienen cerca de 40 Bytes de sobrecarga sobre los datos string en crudo, ya que lo almacenan en un *array* de chars y guardan datos extra como la longitud y almacenan cada carácter como 2 Bytes debido al uso interno de string de codificación UTF-16.

Veremos como determinar el uso de memoria de tus objetos y cómo mejorarlo, cambiando la estructuras de datos o almacenando los datos en formato serializado. Veremos entonces el tamaño de caché de *Spark* y el *garbage collector* de *Java*.

Determinar consumo de memoria

La mejor forma de clasificar la cantidad de consumo de memoria un *dataset* requerirá crear un RDD, ponerla en caché y mirar la interfaz de usuario *web* en la pestaña "*Storage*", donde se muestra el consumo de memoria del RDD.

Para estimar el consumo de memoria de un objeto particular, usa un método de estimación de *SizeEstimator*, útil para experimentar con diferentes disposición de datos y recortar el uso de la memoria, así como la determinación de la cantidad de espacio de una variable *broadcast* que ocupará en el *Heap* de cada *executor*.

Ajustes de la estructuras de datos

La primera forma de reducir consumo de memoria es evitar características *JAVA* que añadan sobrecoste, como estructuras de datos basadas en punteros y objetos *wrapper*.

Hay varias formas de hacerlo:
•Diseñar tu estructura de datos dando prioridad a *arrays* de objetos y tipos primitivos, en vez de colecciones de clase de *Java standard* o *scala* (por ejemplo, *HashMap*), que la librería *fastutil* te puede proporcionar.
•Evitar estructuras anidadas con muchos pequeños objetos y punteros en la medida de lo posible.
•Considerar usar *IDs* numéricos u objetos enumerados en vez de *strings* para claves.
•Si tienes menos de 32 GB de RAM, ajustar el *flag* de *JVM -XX:+UseCompressedOops* para que los punteros sean de 4 Bytes, y no 8 en *Spark*-env.sh.

Almacenamiento de los RDD serializados

Cuando tus objetos son demasiado grandes para almacenarlos eficientemente a pesar del *tunning*, una forma más simple de reducir el uso de memoria es almacenarlos en forma serializada usando *StorageLevels* serializado del *API* que da al RDD persistencia, tal como MEMORY_ONLY_SER

Spark almacenará entonces cada partición de RDD como un gran *array* de bytes. La única desventaja de almacenar los datos de forma serializada es un menor tiempo de acceso debido a que hay que "*deserializar*" cada objeto al vuelo; por lo que se recomienda profusamente el uso de *Kryo* si se quiere cachear datos en forma serializada ya que conduce a tamaños mucho más pequeños que la *serialización* de *Java*.

4.4.4.3 Ajuste del Garbage Collection(GC)

La *"recolección de basura"* en *JVM* puede ser un problema cuando se tiene una gran rotación en términos de *RDDs* almacenados por tu programa.

Cuando *Java* necesita desalojar objetos antiguos para dejar espacio a los nuevos, tendrá que rastrear a través de todos sus objetos *Java* y encontrar las no utilizadas y esto tiene un coste proporcional al número de objetos *Java*. Por eso, usando estructura de datos con menos objetos se *reduce* en gran medida su coste.

GC puede también ser un problema debido a las interferencias entre la cantidad de espacio necesario para ejecutar las tareas y las *RDDs* cacheadas en los nodos. Veremos cómo mitigar esto.

Medida del impacto de GC

El primer paso en el tuneo de *GC* es recoger estadísticas de la frecuencia de la "recolección de basura" y la cantidad de tiempo empleada: añadir -verbose:*GC* -XX:+Print*GC*Details -XX:+Print*GC*TimeStamps a las opciones de *Java*.

La siguiente vez que se ejecute un trabajo de *Spark*, se imprimirán mensajes en el log de los *workers* cada vez que una "recolección de basura" ocurra.

Cache Size Tuning

spark.storage.memoryFraction es un parámetro importante para *GC*. *E*s la cantidad de memoria que debería ser usada para *RDDs* de almacenamiento en caché. Por defecto, *Spark* usa el 60% de la memoria configurada para el *executor* (*spark.executor.memory*) para cachear *RDDs*, lo que significa que el 40% restante está disponible para los objetos creados durante la ejecución de la tarea.

En caso de que tus tareas se ralenticen y encuentres que tu *JVM* está *recoleccionando* basura frecuentemente o ejecutándose sin memoria, bajar este valor te ayudará a reducir el consumo de memoria.

Combinando con el uso de una caché serializada más pequeña debería ser suficiente para mitigar la mayoría de problemas de *GC*

Advanced GC Tuning

Entendamos primero información básica acerca de la gestión de memoria en la *JVM* para después ajustar bien el *GC*:

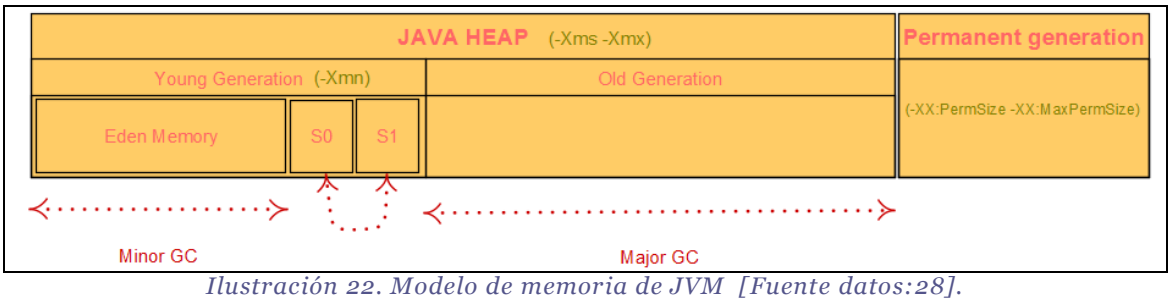


Ilustración 22. Modelo de memoria de JVM [Fuente datos:28].

•El espacio de *Heap* de *Java* está dividido en dos regiones: joven y vieja. La generación joven está pensada para objetos de corta duración , mientras que la vieja generación está diseñada para objetos con tiempos de vida más largos.
•La generación joven está dividida en tres regiones: Eden, Survivor0, Survivor1.

Una descripción simplificada del procedimiento de *GC*: Cuando Eden está lleno, un *GC* menor se ejecuta sobre él y Survivor0 y los objetos que sobreviven son copiados a Survivor1. Las regiones *Survivors* son intercambiadas. Si un objetos es demasiado viejo o Survivor1 está lleno, éste es movido a *Old*. Finalmente cuando *Old* está lleno, un completo *GC* es invocado.

El objetivo de la puesta a punto del *GC* de *Spark* es asegurar que sólo los *RDDs* de larga vida son almacenados en la vieja generación y que la joven generación se encuentre bien dimensionada para almacenar objetos de corta vida. Esto nos ayudará a evitar *GCs* llenos para coleccionar objetos temporales creados durante la ejecución de tareas. Algunos pasos útiles pueden ser:

•Chequear si hay demasiadas "recolecciones de basura" mediante la recogida de estadísticas de *GCs* . Si una *GC* llena es invocada varias veces antes de que una tarea se complete significa que no hay demasiada memoria disponible para ejecutar tareas.

•En las estadísticas que son impresas, si la vieja generación está casi llena, hay que reducir la cantidad de memoria usada para cacheo mediante la propiedad *spark.storage.memoryFraction*. Es mejor cachear pocos objetos que ralentizar la ejecución de la tarea!

Si hay demasiada *GCs menores* de objetos de corta duración y no demasiado *GC completo* de objetos de larga duración, reasignar más memoria para Eden podría ayudar a evitarlo. Como valor inicial de pruebas, se le puede asignar un tamaño a la generación joven de -Xmn=4/3*E., siendo E el tamaño del Eden.

Como un ejemplo, si tu tarea está leyendo datos de *HDFS*, la cantidad de memoria usada por la tarea puede ser estimada usando el tamaño de los bloques de datos leídos de *HDFS*. El tamaño del bloque descomprimido es 2 ó 3 veces el tamaño del bloque; por eso, si nosotros deseamos tener un espacio de trabajo de valor 3 ó 4 tareas y el tamaño de bloque es 64 MB, se puede estimar el tamaño de Eden en 4*3*64 MB

•Supervisar cómo la frecuencia y el tiempo tomado por el *GC* cambia con la nueva configuración.

4.4.4.4 Otros aspectos

Nivel de paralelismo

Los *cluster* no serán plenamente utilizados a no ser que pongas el nivel de paralelismo para cada operación lo suficientemente alto.

Spark automáticamente pone el número de:

•tareas *map* a ejecutar en cada fichero, acorde a su tamaño (aunque se puede controlar este a través de los parámetros opcionales a *Spark*Context.textFile)

•tareas *reduce*, tales como *groupByKey* y *reduceByKey*, acorde al mayor número de particiones *RDDs* padre.

Se puede pasar el nivel de paralelismo como un segundo argumento (ver documentación de *Spark.PairRDDFunctions*, o poner la propiedad de configuración *spark.default.parallelism* con un valor recomendado de 2-3 tareas por *core* en tu *cluster*.

Uso de memoria de tareas Reduce

A veces se obtiene un error de memoria no porque los *RDDs* no se ajusten en memoria, sino porque el conjunto de trabajo de una de las tareas, como una tarea *reduce* en un groupByKey, es demasiado grande.

Operaciones de *Spark* (*sortByKey*, *groupByKey*, *reduceByKey*, *join*, etc) construyen una tabla *hash* con cada tarea para realizar el *grouping*, el cual puede ser grande.

La solución más fácil es incrementar el nivel de paralelismo, de manera que la entrada de cada tarea sea más pequeña.

Spark puede soportar eficientemente tareas de tan solo 200 ms ya que reutiliza un *JVM executor* a través de muchas tareas y esta tiene un bajo coste de lanzamiento por lo que se puede aumentar con más seguridad el nivel de paralelismo que el número de *core*s en el *cluster*.

Broadcasting de grandes variables

El uso de la funcionalidad *broadcast* disponible en *SparkContext* puede reducir en gran medida el tamaño de cada tarea serializada y el coste de lanzamiento de un trabajo sobre el *cluster*.

Si tus tareas desde el *driver* usan un objeto muy grande (por ejemplo una tabla de consulta estática), puede ser interesante convertirla en una variable *broadcast*.

Spark imprime el tamaño serializado de cada tarea en el *master*, y puedes mirar si tu tarea es demasiado grande, normalmente siendo más grande que 20 KB ya vale la pena la optimización.

Localidad de datos

La localidad de datos puede tener un impacto importante en el rendimiento de los trabajos de *Spark*.

Si los datos y el código que opera en ellos están juntos, la computación tiende a ser más rápida. Normalmente *Spark* mueve el código hacia los datos debido a que es más pequeño, construyendo *Spark* su planificación alrededor de este principio general de localidad.

Hay varios niveles de localidad basado en la localización actual de los datos, desde el más cercano al más lejano:

Nivel de localidad	Descripción
PROCESS_LOCAL	Los datos están en la misma *JVM* donde se ejecuta el código. Es la mejor localidad posible.
NODE_LOCAL	Un poco más lento, porque los datos tienen que viajar entre procesos.
NO_PREF	Los datos son accesibles igual de rápido desde cualquier sitio y no hay una preferencia de localidad
*RACK*_LOCAL	Los datos están en el mismo *rack* de servidores y en diferentes servidores del mismo, por eso necesitan se enviados sobre la red, normalmente a través de un solo *switch*
ANY	Los datos están en cualquier sitio de la red y no en el mismo *rack*

Tabla 17. Nivel de localidad de los datos en memoria.

Spark prefiere planificar todas las tareas en una mejor localidad, pero esto no siempre es posible. En situaciones donde no hay datos sin procesar en un *executor* ocioso, *Spark* cambia a un nivel más bajo la localidad. Hay dos opciones:
1.-esperar a que una *cpu* se libere para empezar una tarea sobre los datos en el mismo servidor.
2.-empezar inmediatamente una nueva tarea en un sitio más lejano que requiera mover los datos allí.

Spark normalmente espera un poco con la esperanza de que una *CPU* ocupada se libere. Una vez que el tiempo de espera expira, comienza a mover los datas hacia la *cpu* libre.

El tiempo de espera para replegarse entre cada nivel puede configurarse individualmente o todos juntos en un parámetro, *spark.locality*.

Deberías incrementar estos valores si las tareas son de larga duración y pobre localidad, aunque los parámetros por defecto suelen funcionar bien.

4.3.4 Slaves file

Normalmente se elige una máquina para que actúe como *Master* (En nuestro caso EIP6029) y el resto de máquinas, 30, actúan como esclavas (las máquinas EIP60XX a excepción de EIP6029 y EIP6001).

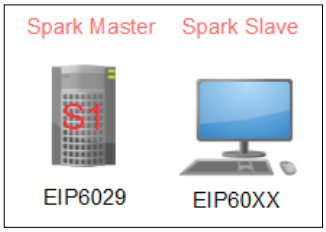


Ilustración 23. Roles Spark de los equipos en el Aula Anita Borg.

En el fichero situado en *conf/slaves* se listan todos los hostnames o direcciones *IP*, uno por línea, de los *slaves*. En total, 30 equipos.

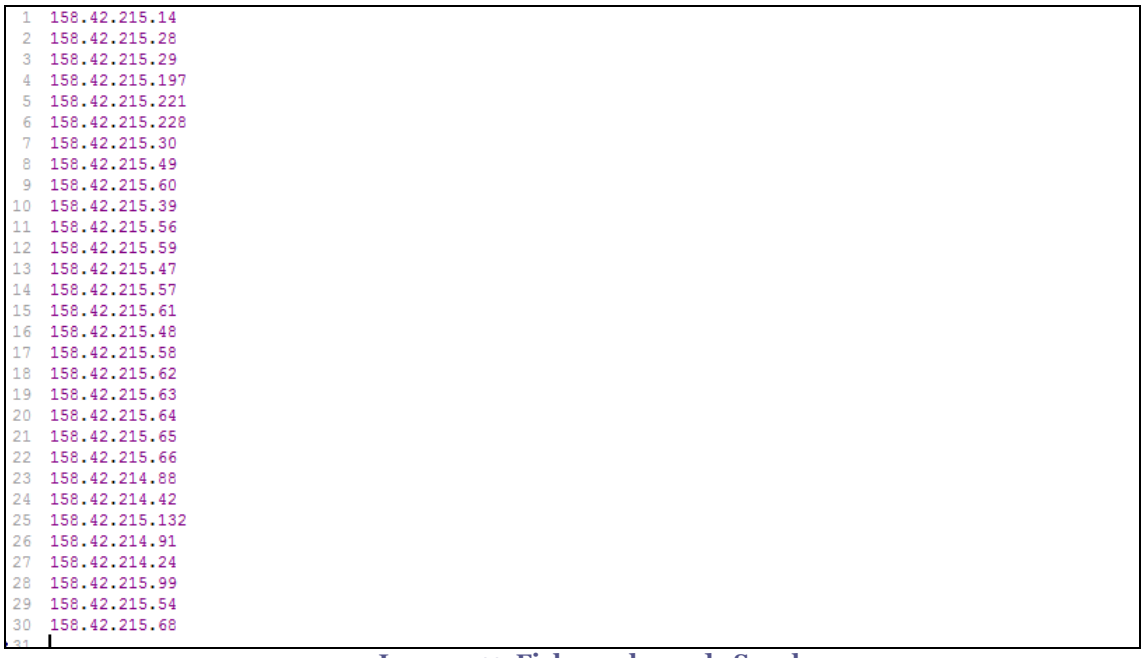


Imagen 11. Fichero slaves de Spark.

4.4.5 Pasos de Configuración

1.-Configurar variables de entorno, añadiendo al fichero ~/*.cluster_rc*, ya creado en el apartado 4.3.5, las siguientes líneas:

```
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
```

Nota: *SPARK*_PREFIX deben adecuarse a la instalación.

2.- Crear el fichero *slave*, con el contenido de un nombre o dirección *IP* de *workernode* por línea.

```
~/cluster/opt/spark/conf/slaves
```

4.5 Despliegue Apache Hadoop y Apache Spark

Hemos instalado y configurado *Hadoop* y *Spark* en el equipo EIP6029.

Seguidamente, lo desplegaremos en los 31 equipos restantes , siguiendo los pasos:

1.- Debe haber finalizado con éxito el acceso remoto al resto de equipos.

2.- Entrar al ordenador EIP6029

3.- Ubicarse en el directorio *HOME* del usuario *'usulocal'*, en este caso.

4.- Copiar mediante el comando *rsync* el contenido del directorio '*cluster*' al nuevo *workernode*.

```
rsync -avH --delete cluster usulocal@<workdernode>
```

5.-Copiar mediante el comando *scp* al equipo destino, los ficheros:

- ~/*.bashrc*

• ~/*.profile*
• ~/*.cluster*_rc

Dado que tenemos que realizar la operación 31 veces, vamos a realizar un script que nos lo automatice al que llamaremos *desplegar-hadoop-spark.sh, con el siguiente contenido*:

```
#!/bin/bash

i=0
IPS[i++]=215.14
IPS[i++]=215.28
IPS[i++]=215.29
IPS[i++]=215.197
IPS[i++]=215.221
IPS[i++]=215.228
IPS[i++]=215.30
IPS[i++]=215.49
IPS[i++]=215.60
IPS[i++]=215.39
IPS[i++]=215.56
IPS[i++]=215.59
IPS[i++]=215.47
IPS[i++]=215.57
IPS[i++]=215.61
IPS[i++]=215.48
IPS[i++]=215.58
IPS[i++]=215.62
IPS[i++]=215.63
IPS[i++]=215.64
IPS[i++]=215.65
IPS[i++]=215.66
IPS[i++]=214.88
IPS[i++]=214.42
IPS[i++]=215.132
IPS[i++]=214.91
IPS[i++]=214.24
IPS[i++]=215.99
IPS[i++]=215.54
IPS[i++]=215.68

rm -f slaves.temp
for ip in ${IPS[@]}
do
        workernode="192.68.10.${ip}"
        echo ${workernode}
        echo ${workernode} >>slaves.temp

        # Despliega la configuración realizada en el resto de equipos
        rsync -avH ~/cluster usulocal@${workernode}:~/cluster
        # Despliega ficheros de entorno de Spark en el resto de equipos
        scp ~/.cluster_rc  usulocal@${workernode}:.cluster_rc
        scp ~/.bashrc      usulocal@${workernode}:.bashrc
        scp ~/.profile     usulocal@${workernode}:.profile
done
```

Para ejecutarlo, nos situaremos en el directorio del equipo EIP6029 donde esté el fichero a ejecutar:

```
./ desplegar-Hadoop-Spark.sh
```

5. Guía resumen instalación, configuración y ejecución

1.-Diseñar el mapa del cluster, por ejemplo con 5 ordenadores:

Ordenador 1: *HDFS NameNode* + *Spark Master*

Ordenador 2: *YARN ResourceManager*+ JobHistoryServer+ProxyServer

Ordenador 3, 4 y 5: *HDFS DataNode*+*YARN NodeManager*+*Spark* Slave

2.-Actividades previsas: en todos los ordenadores del cluster:

•Crear un usuario Hadoop, por ejemplo "usulocal"

```
$ sudo useradd -m usulocal
$ sudo passwd usulocal
```

•Debe haber instalada, una versión de JRE.

-Para comprobarlo y saber qué versión está instalada, ejecutar el comando:

```
$ Java -version
```

-Si no estuviera instalado, acceder a la página oficial para informarse [22].

-Para saber qué valor poner en $JAVA_HOME, consultar tutorial [25] y una vez encontrado el *path*, omitir los directorios finales */bin/java*, ya que en los ficheros *.sh se les añade.

•Debe haber instalada una versión de SSH para generar un par de claves RSA y autenticar por SSH:

-En el ordenador 1 creamos el par de claves publica/privada RSA.

```
$ cd
$ mkdir -p .ssh
$ cd .ssh
$ ssh-keygen -t dsa -P '' -f id_dsa  #ojo: "son dos comillas simples sin espacio
$ ls -l  #scomprobamos que se han creado los ficheros id_dsa.pub  y id_dsa.pub
```

- Para cada ordenador, realizamos dos pasos:

a.-Añadir el fichero *id_dsa.pub* al fichero ~/.ssh/authorized_keys.

```
# Desde el ordenador 1
$ cd
$ cd .ssh
$ ssh usulocal@<ip_worker-node>  mkdir -p .ssh
$ cat id_dsa.pub | ssh usulocal@<ip_worker-node> "cat - >>.ssh/authorized_keys"
```

b.-Modificar los permisos para poder acceder sin contraseña.

```
# Desde el ordenador 1 accedemos al ordenador destino y modificamos permisos
$ ssh usulocal@<ip_worker-node>  #sustituir <ip_worker-node>  por una IP
# Solicita contraseña y se le facilita
$ cd ~/.ssh
```

```
$ chmod go-rwx .
$ chmod go-rw authorized_keys
```

c.-Y ya podemos acceder de forma remota desde el ordenador 1 al resto de equipos sin introducir contraseña alguna:

```
ssh usulocal@<ip_worker-node> #sustituir <ip_worker-node>  por una IP
```

3.-Obtener los enlaces http a los instalables de Apache Hadoop version 2.6.2 y Apache Spark version 1.5.2

Hadoop:

http://*Hadoop*.apache.org/

Opción de menú *Release*s, pulsamos *Download*.

Pulsar binary de la versión 2.6.2

Apuntamos uno cualquiera de los enlaces para usar posteriormente,

Apache Spark

http://*Spark*.apache.org/

Pulsamos en ***download***.

Seleccionamos:

-versión de despliegue: *1.5.2*

-tipo de paquete: *Pre-built for Hadoop 2.6 and later*.

-tipo de *download*: *Apache Mirror*

Pulsamos en Download Spark: *spark-1.5.2-bin-hadoop2.6.tgz*.

Apuntamos uno cualquiera de los enlaces para usar posteriormente,

3.-Instalación en Ordenador 1: Apache Hadoop y Apache Spark.

```
# Nos situamos en ordenador 1. Descargamos instalables de Hadoop y Spark
$ cd
$ mkdir -p cluster/opt
$ cd cluster/opt
$ wget http://apache.rediris.es/Hadoop/common/Hadoop-2.6.2/Hadoop-2.6.2.tar.gz
$ wget http://apache.rediris.es/Spark/Spark-1.5.2/Spark-1.5.2-bin-Hadoop2.6.tgz
# Desplegamos, descomprimiendo, Hadoop en el directorio /cluster/opt
$ tar zxvf  Hadoop-2.6.2.tar.gz
# Creamos un enlace dinámico paa trabajar con /cluster/opt/Hadoop
$ rm Hadoop
$ ln -s Hadoop-2.6.2 Hadoop
# Desplegamos, descomprimiendo,  Spark en el directorio /cluster/opt
$ cd ~/cluster/opt
$ tar zxvf Spark-1.5.2-bin-Hadoop2.6.tgz
$ rm Spark
$ ln -s Spark-1.5.2-bin-Hadoop2.6 Spark
```

4.-Configuración Hadoop.

a.-Editamos con la configuración adecuada los ficheros yarn-env.sh, *hadoop*-env.sh, mapred-env.sh, core-site.xml, *hdfs-site.xml, yarn*-site.xml y *mapred-site.xml.*

b.-Creamos los siguientes directorios para Hadoop, cuyos valores deben ajustarse en los ficheros de configuración.

```
mkdir "${HOME}/cluster/var/hadoop"
mkdir "${HOME}/cluster/var/hadoop/hadoop-DataNode"
mkdir "${HOME}/cluster/var/hadoop/hadoop-NameNode"
mkdir "${HOME}/cluster/var/hadoop/mr-history"
mkdir "${HOME}/cluster/var/hadoop/mr-history/done"
mkdir "${HOME}/cluster/var/hadoop/mr-history/tmp"
```

c.- Crear el fichero *slaves*, con el contenido de un nombre de equipo o dirección *IP* de los *workers* por línea.

```
~/cluster/opt/hadoop/etc/hadoop/slaves
```

d.- Configurar variables de entorno:

-Crear el fichero ~/.*cluster*_rc y añadir las siguientes líneas (después con Spark añadiremos algunas líneas más):

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

nota: *HADOOP*_PREFIX y *SPARK*_PREFIX deben adecuarse a la instalación.

-Añadir al fichero ~/.bashrc y al fichero ~/.profile la siguiente línea:

```
. ~/.cluster_rc
```

5.-Configuración Spark.

a.- Crear el fichero *slaves*, con el contenido de un nombre de equipo o dirección *IP* de los *workers* por línea.

```
~/cluster/opt/spark/conf/slaves
```

b.- Configurar variables de entorno:

-Editar el fichero ~/.*cluster*_rc añadiendo las líneas que corresponden a Spark, quedando definitivamente:

```
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"
```

nota: *SPARK*_PREFIX deben adecuarse a la instalación..

c.- Editar el fichero ~/cluster*/opt/spark/conf/log4j.properties*.

Cambiendo INFO por WARN en la línea "*log4j.root category= INFO*" nos saldrán menos mensajes.

6.-Despliegue de lo configurado al resto de equipos.

Entrar en el ordenador 1 y ubicarse en el directorio *HOME* del usuario "usulocal" y hacer las siguientes operaciones con el resto de equipos:

-Copiar el contenido del directorio '*cluster*'

/cluster

-Copiar mediante el comando *scp* al equipo destino, los ficheros:

~/*.bashrc*

~/*.profile*
~/*.cluster*_rc

a.-Lo más cómodo es realizar un script que nos lo automatice, *desplegar-hadoop-spark.sh*:

```
#!/bin/bash

i=0
# las IP del Ordenador 2, 3, 4 y 5. Estas cuatro líneas están a modo de #ejemplo:
IPS[i++]=.50
IPS[i++]=.51
IPS[i++]=.52
IPS[i++]=.53

rm -f slaves.temp
for ip in ${IPS[@]}
do
        workernode="192.68.10.${ip}"
        echo ${workernode}
        echo ${workernode} >>slaves.temp

        # Despliega la configuración realizada en el resto de equipos
        rsync -avH ~/cluster usulocal@${workernode}:~/cluster
        # Despliega ficheros de entorno de Spark en el resto de equipos
        scp ~/.cluster_rc  usulocal@${workernode}:.cluster_rc
        scp ~/.bashrc     usulocal@${workernode}:.bashrc
        scp ~/.profile    usulocal@${workernode}:.profile
done
```

Para ejecutarlo, nos situaremos en el directorio del Ordenador 1 donde esté el fichero a ejecutar:

```
./ desplegar-Hadoop-Spark.sh
```

7.-Levantar el cluster

En ${HADOOP_HOME} tenemos el valor ~/cluster/opt/hadoop

En ${SPARK_HOME} tenemos el valor ~/cluster/opt/spark

•En el Ordenador 1:

a.-Formateamos HDFS. Esta operación la hacemos la primera vez solamente.

```
$ ./${HADOOP_HOME}/bin/hdfs namenode -format
```

b.-Levantamos los servicios de HDFS

```
$ ./$ {HADOOP_HOME}/sbin/start-dfs.sh
```

•En el ordenador 2:

c.-Levantamos los servicios de Y*arn*

```
$ ./$ {HADOOP_HOME}/sbin/start-yarn.sh
```

d.-Levantamos los servicios de J*obhistoryserver*

```
$ ./$ {HADOOP_HOME}/sbin/mr-jobhistoryserver start historyserver
--config{HADOOP_CONF_DIR}
```

•En el ordenador 1:

e.-Levantamos los servicios de *Spark*

```
$ ./$ {SPARK_HOME}/sbin/start-all.sh
```

7.-Ejecutar una aplicación en el cluster.

a.-Tenemos una aplicación escrita en el lenguaje de programación python, llamado *aplicacion_1.py situada el directorio ~/app del Ordenador 2, por ejemplo.*

b.-En ese mismo directorio, creamos el fichero *lanzar_a_cluster_aplicacion_1.sh*:

```
#!/bin/bash

export PYTHONPATH="${HOME}/python"

time spark-submit --master yarn-cluster \
        --driver-memory 6G \
        --executor-memory 6G \
        --num-executors 10 \
        --total-executor-cores 40 \
        --py-files ${PYTHONPATH}/mypythonlib.tgz \
         aplicacion_1.py
```

c.-Y lo ejecutamos:

```
$ ./ lanzar_a_cluster_aplicacion_1.sh
```

8.-Seguimiento ejecución.

a.-Desde *http://IP_Ordenador 1:50070* podemos acceder al almacenamiento de nuestros fichero en HDFS.

b.-Desde *http://IP_Ordenador 1:8088* podemos acceder a lo que está sucediendo en Spark mientras se ejecuta la aplicación.

9.-Parar el cluster.

•En el ordenador 1:

a.-Paramos los servicios de *Spark*

```
$ ./$ {SPARK_HOME}/sbin/stop-all.sh
```

•En el ordenador 2:

b.-Paramos los servicios de Y*arn*

```
$ ./$ {HADOOP_HOME}/sbin/stop-yarn.sh
```

c.-Paramos los servicios de J*obhistoryserver*

```
$ ./$ {HADOOP_HOME}/sbin/mr-jobhistoryserver stop historyserver
--config{HADOOP_CONF_DIR}
```

•En el ordenador 1:

d.-Paramos los servicios de HDFS

```
$ ./$ {HADOOP_HOME}/sbin/stop-dfs.sh
```

6. Evaluación de rendimiento

Los puntos previos del TFG nos han ido ayundando a llegar donde estamos ahora. Estamos delante del cluster y vamos a lanzar en él 2 aplicaciones y veremos cómo responden tanto el cluster como nuestras aplicaciones.

El *cluster* es un entorno nuevo y antes de enfrentarnos a él, y para tener una guía inicial, preguntamos a Dº Adrián Fernández Manzano, de Treelogic y Dº Rubén Casado Tejedor, de Accenture sobre cómo podemos ajustar la configuración de Spark para que nuestros algoritmos tengan la mejor configuración y puedan convivir óptimamente con otra aplicaciones en el *cluster*.

La respuesta es que no hay una guía mágica. Hay que lanzar la aplicación con distintas configuraciones al *cluster* y eliges la que mejor se adapte a tu algoritmo o tu conjunto de algoritmos. Y eso hicimos.

6.1 Algoritmo Contar líneas y palabras

Tenemos una aplicación, llamada *wordCount3-2.py,* escrita en el lenguaje de programación *python*[30] que dado un directorio en HDFS cuenta las líneas y las palabras de los ficheros que lo contienen, de tres formas distintas.

```
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/test-carga"
    output_dir="hdfs://eip6029.inf.upv.es:9000//user/usulocal/output/wc3-1.d"

    sc = SparkContext( appName="wordCount3" )
    file_contents = sc.textFile( input_dir ,60) # For loading the contents of all the files in a directory

    file_contents.persist()

    lines_count_1 = file_contents.count()
    print( "Procesadas un total de %d lineas con count()"    % lines_count_1 )
    lines_count_2 = file_contents.map( lambda line : 1 ).reduce( lambda c1,c2 : c1+c2 )
    print( "Procesadas un total de %d lineas con map+reduce" % lines_count_2 )
    lines_count_3 = file_contents.map( lambda line : 1 if len(line.strip()) > 0 else 0 ).reduce( lambda c1,c2 : c1+c2 )
    print( "Procesadas un total de %d lineas no vacias con map+reduce" % lines_count_3 )

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

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

    word_counter = sc.accumulator(0)
    file_contents.foreach( function_word_counter_1 )
    print( "Palabras contadas con acumulador 1: %d " % word_counter.value )

    word_counter.value = 0
    file_contents.foreach( function_word_counter_2 )
    print( "Palabras contadas con acumulador 2: %d " % word_counter.value )

    word_counts.unpersist()
    file_contents.unpersist()

    sc.stop()
```

Imagen 12. Fichero wordCount3-2.py con código fuente en python.

[30] Lenguaje interpretado, multiparadigma (orientado a objetos, imperativo y algo de funcional), multiplataforma y usa tipado dinámico, cuya filosofía hace hincapié en una sintaxis que favorezca un código legible.

Funcionamiento del algoritmo.

El algoritmo almacena toda la información en un RDD y posteriormente hace varias operaciones, sobre él.

•Cuenta líneas de 3 formas distintas:

1.-Contando los elementos del RDD, que coinciden cada uno con una línea.

2.-Haciendo uso de las funciones *Map* y *Reduce*. *Map asigna a cada línea el valor 1 y* la función *Reduce* las va sumando.

3.-La función Map asigna el valor 1 si la línea tiene una longitud mayor que cero, y el valor 0 si es una línea en blanco. La funcion *Reduce* las va sumando.

•Y cuenta las palabras de 3 formas distintas también:

1.-Con la función *flatmap* divide las líneas del *RDD* en palabras que le llegan a la función *Map, la cual* le asigna un valor 1 a cada una de ella. Ya la función *reduce* las va sumando.

2 y 3 utilizan un bucle para obtener cada línea y a cada una de ella le aplica una función distinta para obtener el número de palabras que las va insertando en un acumulador.

Script de lanzamiento al cluster

Creamos el *script sparkFusion.sh* para lanzar nuestro algoritmo al cluster.

```
#!/bin/bash

export PYTHONPATH="${HOME}/python"

hdfs dfs -rm -r /user/usulocal/output/wc3-1.d

time spark-submit --master yarn-cluster \
        --driver-memory 6G \
        --executor-memory 6G \
        --num-executors 30 \
        --total-executor-cores 120 \
        --py-files ${PYTHONPATH}/mypythonlib.tgz \
    wordCount3-2.py 2>wc3.log
```

Imagen 13. Scritp sparkFusionsh .

El significado de los parámetros que utilizaremos es:

-executor-memory: Cantidad de memoria a usar por *executor*.

-num-executor: número de executors, se corresponde con el número de workernode.

-total-executor-cores: total cores para todos *los executors*.

Combinamos *num-executor* y *total-executor-cores* para asegurarnos de utilizar todos los *executor* con todos sus *cores*.

Al lanzarse sparkFusion.sh al cluster vemos en la *Captura 12* que el algoritmo ha contado 772.8 millones de palabras de 35 millones de líneas.

```
usulocal@lxeip6001 ~/examples2 $ ./sparkFusion.sh
16/09/02 13:34:22 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted /user/usulocal/output/wc3-1.d
Procesadas un total de 35049320 lineas con count()
Procesadas un total de 35049320 lineas con map+reduce
Procesadas un total de 34786912 lineas no vacias con map+reduce
Palabras contadas con map.reduce:     772847137
Palabras contadas con acumulador 1: 772847137
Palabras contadas con acumulador 2: 772847137

real    1m3.373s
user    0m19.173s
sys     0m1.377s
usulocal@lxeip6001 ~/examples2 $
```

Captura 12. Lanzamiento al cluster con sparkfusion.sh.

Ajustes de configuración en el cluster

Los recursos que más influyen en la ejecución óptima de un algoritmo suelen ser: memoria RAM, núcleos de procesador y velocidad de red. Veamoslo.

Velocidad de red.

Comprobamos que la velocidad de red no influía en modo alguno para que fuera susceptible de ser configurado con la finalidad de que se ejecute más rápido nuestro algoritmo.

Según vemos en la *Imagen 14*, la evolución de la gráfica de red de un workernode, mientra se ejecuta el fichero *wordCount3-2.py, no llega al máximo de 12MB/s*, situándose su techo en 1MB/s aproximadamente. No es un recurso crítico.

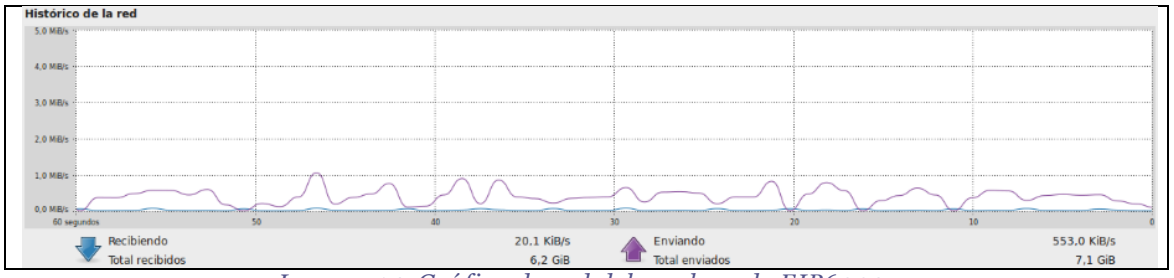


Imagen 14. Gráfica de red del workenode EIP6002.

Memoria RAM y Núcleos de procesador

Ejecutaremos el algoritmo cambiando los parámetros *executor-memory, num-executor y total-executor-cores,* para buscar la mejor combinación. Nos dá los tiempos mostrados en la Tabla 18.

Num-executor / Executor-memory	1GB	3GB	6GB
1	758	498	483
5	190	203	176
10	111	102	100
20	77	77	74
30	66	72	63

Tabla 18. Tiempo de ejecución wordCount3-2.py (en segundos).

Veamoslo mejor en un gráfico.

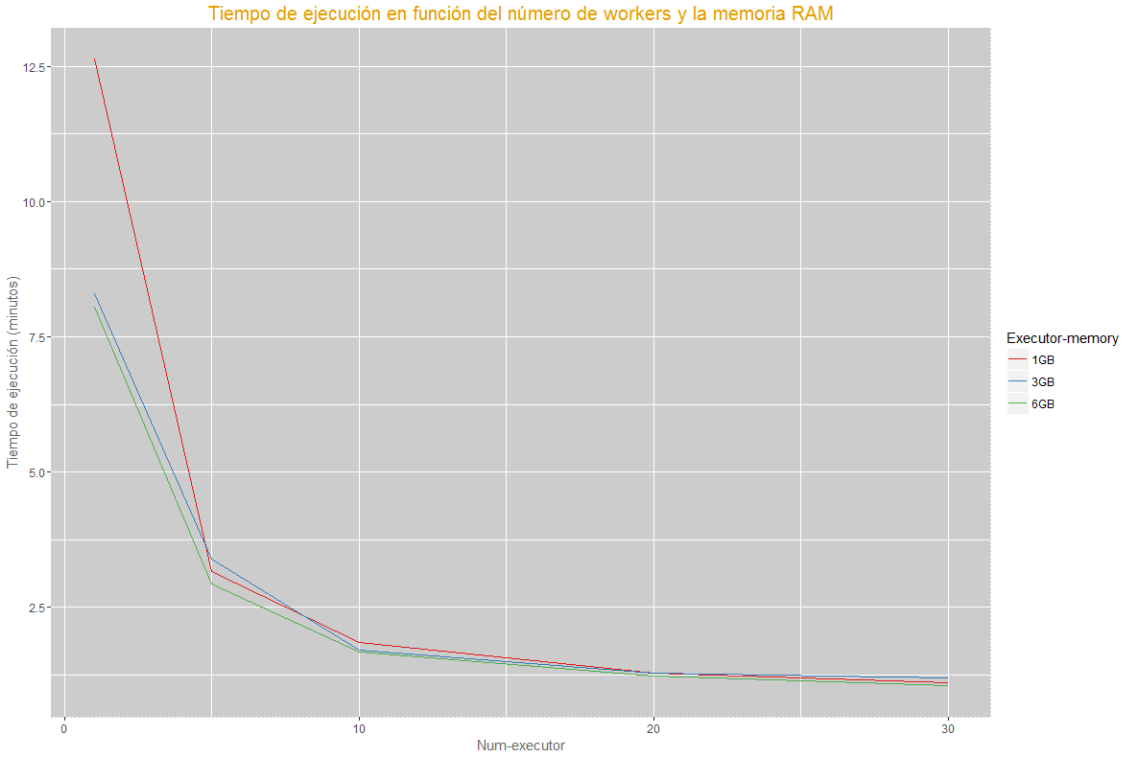


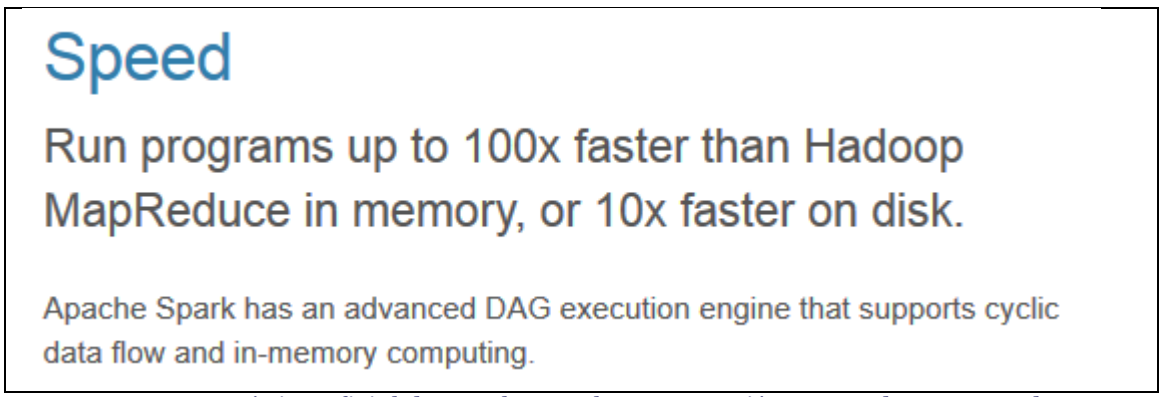
Gráfico 5. Tiempo de ejecución wordCount3-2.py (en minutos)

En el G*ráfico 5*, observamos que el *número de executor* sí que influye a la hora de obtener un buen tiempo de ejecución de nuestro algoritmo; en cambio el *executor-memory*, no influye.

Por tanto, en función de las necesidades, nos hacemos una idea de como podemos repartir los recursos de un *cluster* en el que normalmente se estarán ejecutando otros algoritmos.

Spark vs Yarn

Spark dice en su página oficial que sus programas se ejecutan hasta 100 veces más rápidos que Hadoop MapReduce. Vamos a comprobarlo.



Captura 13. Página oficial de Apache Spark. Comparación con Hadoop MapReduce.

Para tomar tiempos de ejecución, dejamos fijos todos los parámetros, excepto

•*num-executor: número de executors, se corresponde con el número de workernode.*

•*total-executor-cores: total cores* para todos *los executors.*

•*tipo de gestor del cluster:*

spark-submit --master yarn-cluster

spark-submit --master spark://eip6029.inf.upv.es:7077

Los resultados fueron:

Num-executor / Type cluster	Yarn	Spark	Proporción
1	1946	483	4.03
5	510	176	2.90
10	340	100	3.40
20	338	74	4.57
30	332	63	5.27

Tabla 19. Tiempo de ejecución wordCount3-2.py (en segundos).

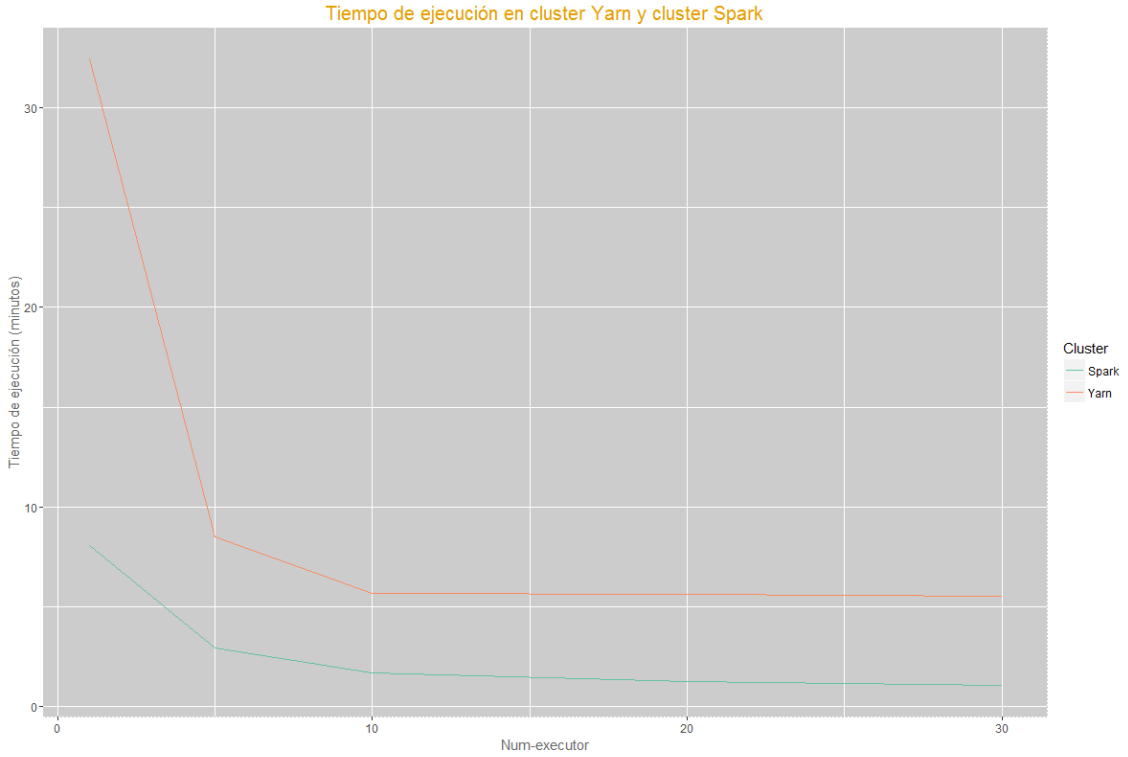


Gráfico 6 Tiempo de ejecución wordCount3-2.py (en minutos)

Spark se ejecuta con más rápido que Yarn.

Con este algoritmo y esta carga de trabajo no logra ser 10 veces más rápido que Yarn, pero sí que hay una diferencia significativa llegando a ser 5 veces más rápido.

Para hacer un estudio más serio por nuestra parte, habría que ver qué diferencias existen con otro tipo de algoritmos y con distintas cargas de trabajo para ver cómo se comportan ambos tipos de gestores.

No obstante, nosotros nos quedamos con Spark.

6.2 Algoritmo GMM - Gaussian Mixture Models

GMM (*Gaussian mixture models*) es un modelo probabilístico que representa la presencia de "*subpoblaciones*", o clases, dentro de una población general, sin necesidad de que un conjunto de datos deba identificar la "*subpoblación*" a la que pertenece, y que han sido generados por un número finito de distribuciones *Gaussianas* con parámetros desconocidos.

Con la ayuda de *MLE* (*Maximum likelihood estimation*), iremos calculando tantas "*subpoblaciones*" como clases le digamos, haciendo las subdivisiones necesarias del conjunto de datos.

El fichero test_MLE_2.py contiene el código fuente que lanzaremos al *cluster*. Su código lo mostramos en la *Captura 15*.

Script de lanzamiento al cluster

El fichero *run-test-MLE.sh* lanzará al cluster nuestro fichero *test_MLE_2.py*.

Para estudiar su comportamiento iremos combinando distintos valores de los parámetros, en distintas ejecuciones, para lograr obtener aquellos que nos den un menor tiempo de ejecución.

```
#!/bin/bash

export PYTHONPATH="${HOME}/python"


function standalone()
{
    dataset="$1"
    python test_MLE_2.py --standalone --dataset ${dataset}
}

function spark_master()
{
    dataset="$1"
    #spark-submit --master local[4] test_MLE_2.py

    spark-submit --master spark://eip6029.inf.upv.es:7077 \
             --driver-memory 4G \
             --executor-memory 4G \
             --total-executor-cores 120 \
             --py-files ${PYTHONPATH}/mypythonlib.tgz \
             test_MLE_2.py  --num-slices 120  --batch-size 1000 --dataset ${dataset}
}

dataset="etc/rodrigo-index-raw5.txt"
spark_master datasets/Rodrigo/data/embedded5
```

Captura 14. run-test-MLE.sh.

Carga de datos

Como prueba inical intentamos ejecutar nuestro algoritmo en el cluster pero obteniendo los datos del sistema de ficheros local. Al intentar cargar un RDD de 3.5 GB de datos en memoria, tardó 53 minutos. Y una vez, cargado se producía un error de memoria en la *JVM*.

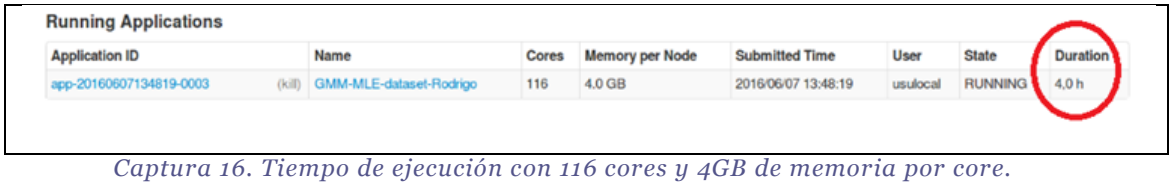
Aunque había otras alternativas, como manipular el tamaño de memoria de la JVM, sencillamente optamos por pasar los datos a HDFS.

```
"""
    Author: Jon Ander Gomez Adrian (jon@dsic.upv.es, http://www.dsic.upv.es/~jon)
    Version: 2.0
    Date: June 2016
    Universitat Politecnica de Valencia
    Technical University of Valencia TU.VLC

    Testing Maximum Likelihood Estimation

"""

import sys
import numpy

import machine_learning

try:
    from pyspark import SparkContext
except:
    pass


def load_samples_from_file( filename ):
    f=open( filename, "rt" )
    line = f.readline()
    dim_x = len(line.split())
    counter = 1
    for line in f: counter+=1
    f.close()
    X = numpy.zeros( [ counter, dim_x ] )
    f=open( filename, "rt" )
    n=0
    for line in f:
        X[n,:] = [ float(x) for x in line.split() ]
        n+=1
    f.close()
    return X,None

def load_samples( index_filename ):
    f=open( index_filename, "rt" )
    X = []
    Y = []
    for filename in f:
        print( "loading file " + filename.strip() )
        _x_, _y_ = load_samples_from_file( filename.strip() )
        X.append( _x_ )
        Y.append( _y_ )
        #print( _x_.shape )
    f.close()
    return X,Y

if __name__ == "__main__":

    """
    Usage: spark-submit --master local[4]  python/gmm-mle.py  --base-dir .   --dataset data/samples.txt.gz  --covar full      --max-components 10
dev/null
           spark-submit --master local[4]  python/gmm-mle.py  --base-dir .   --dataset data/samples.txt.gz  --covar diagonal  --max-components  5
dev/null
    """

    verbose=False
    covar_type='diagonal'
    max_components=300
    dataset_filename=None
    base_dir='.'
    standalone=False
    spark_context = None
    slices=0
    batch_size = 100

    for i in range(len(sys.argv)):
        if sys.argv[i] == "--covar":
            covar_type = sys.argv[i+1]
        elif sys.argv[i] == "--max-components":
            max_components = int(sys.argv[i+1])
        elif sys.argv[i] == "--dataset":
            dataset_filename = sys.argv[i+1]
        elif sys.argv[i] == "--verbosity":
            verbose = int(sys.argv[i+1])
        elif sys.argv[i] == "--standalone":
            standalone = True
        elif sys.argv[i] == "--num-slices":
            slices = int(sys.argv[i+1])
        elif sys.argv[i] == "--batch-size":
            batch_size = int(sys.argv[i+1])

    if not standalone :
        spark_context = SparkContext( appName="GMM-MLE-dataset-Rodrigo" )

    #os.makedirs( base_dir+'/log',    exist_ok=True )
    #os.makedirs( base_dir+'/models', exist_ok=True )

    if spark_context is not None:

        """
            Load all the lines in a file (or files in a directory) into an RDD of text lines.

            It is assumed there is no header, each text file contains an undefined number or lines.
            - Each line represents a sample.
            - All the lines **must** contain the same number of values.
            - All the values **must** be numeric, integers or real values.
        """
        text_lines = spark_context.textFile( dataset_filename )
        print( "file(s) loaded " )
        text_lines.persist()
        num_samples = text_lines.count()
        text_lines.unpersist()
        print( "loaded %d samples " % num_samples )

        """
            Convert the text lines into numpy arrays.

            Taking as input the RDD text_lines, a map operation is applied to each text line in order
            to convert it into a numpy array, as a result a new RDD of numpy arrays is obtained.

            Nevertheless, as we need an RDD with blocks of samples instead of single samples, we
            associate with each sample a random integer number in a specific range.

            So, instead of an RDD with of numpy arrays we get an RDD with tuples [ int, numpy.array ]
        """
        K = (num_samples + batch_size - 1) / batch_size
        K = ((K // slices)+1)*slices
        samples = text_lines.map( lambda line: ( numpy.random.randint(K), numpy.array( [ float(x) for x in line.split() ] ) ) )

        # HAS BEEN UNPERSISTED ABOVE: text_lines.unpersist() # This RDD is no longer needed

        # Shows an example of each element in the temporary RDD of tuples [key, sample]
        print( samples.first() )
        print( type(samples.first()) )

        """
            Convert the RDD of tuples to the definitive RDD of blocks of samples
        """
        samples = samples.map( lambda x: x[1] )

        # Shows an example of each element in the temporary RDD of blocks of samples
        print( samples.first() )
        print( type(samples.first()) )

        samples.persist()
        print( "we are working with %d blocks of approximately %d samples " % ( samples.count(), batch_size ) )

        # Shows an example of shape of the elements in the temporary RDD of blocks of samples
        print( samples.first().shape )
        # Gets the dimensionality of samples in order to create the object of the class MLE.
        dim_x = samples.first().shape[1]

        mle = machine_learning.MLE( covar_type=covar_type, dim=dim_x, log_dir=base_dir+'/log', models_dir=base_dir+'/models' )

        mle.fit_with_spark( spark_context=spark_context, samples=samples, max_components=max_components )

        samples.unpersist()
        spark_context.stop()
    else:
        X_train,Y_train = load_samples( dataset_filename )
        dim_x = 0
        if type(X_train) == list :
            dim_x = X_train[0].shape[1]
        elif type(X_train) == numpy.ndarray:
            dim_x = X_train.shape[1]
        else:
            raise Exception( 'Non accepted data structure! :: %s' % (type(X_train)) )

        mle = machine_learning.MLE( covar_type=covar_type, dim=dim_x, log_dir=base_dir+'/log', models_dir=base_dir+'/models' )
        mle.fit standalone( samples=X train, max components=max components, batch size=500 )
```

Captura 15. Fichero test_MLE_2.py con código fuente en python.

Prueba de estrés

Lo primero que hicimos fue realizar una carga de estrés para ver si funcionaba. Después de 4 horas funcionando y ver que funcionaba con normalidad, paramos el proceso.

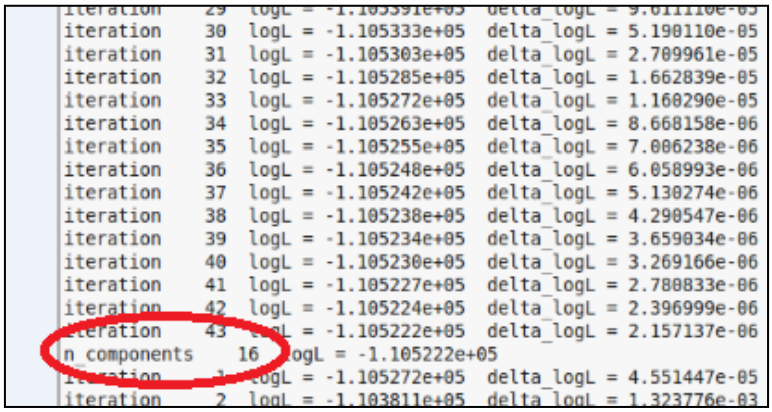


Running Applications

Application ID		Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20160607134819-0003	(kill)	GMM-MLE-dataset-Rodrigo	116	4.0 GB	2016/06/07 13:48:19	usulocal	RUNNING	4,0 h

Captura 16. Tiempo de ejecución con 116 cores y 4GB de memoria por core.

El número de componentes o clases calculado fue de 16.



```
iteration    29  logL = -1.105391e+05  delta_logL = 9.611110e-05
iteration    30  logL = -1.105333e+05  delta_logL = 5.190110e-05
iteration    31  logL = -1.105303e+05  delta_logL = 2.709961e-05
iteration    32  logL = -1.105285e+05  delta_logL = 1.662839e-05
iteration    33  logL = -1.105272e+05  delta_logL = 1.160290e-05
iteration    34  logL = -1.105263e+05  delta_logL = 8.668158e-06
iteration    35  logL = -1.105255e+05  delta_logL = 7.086238e-06
iteration    36  logL = -1.105248e+05  delta_logL = 6.058993e-06
iteration    37  logL = -1.105242e+05  delta_logL = 5.130274e-06
iteration    38  logL = -1.105238e+05  delta_logL = 4.290547e-06
iteration    39  logL = -1.105234e+05  delta_logL = 3.659034e-06
iteration    40  logL = -1.105230e+05  delta_logL = 3.269166e-06
iteration    41  logL = -1.105227e+05  delta_logL = 2.780833e-06
iteration    42  logL = -1.105224e+05  delta_logL = 2.396999e-06
iteration    43  logL = -1.105222e+05  delta_logL = 2.157137e-06
n_components    16  logL = -1.105222e+05
iteration     1  logL = -1.105272e+05  delta_logL = 4.551447e-05
iteration     2  logL = -1.103811e+05  delta_logL = 1.323776e-03
```

Captura 17. Log resultado de la ejecución de la Captura 16.

Ajustes de configuración en el cluster

Nuestro ***objetivo*** será, ahora, minimizar el tiempo de ejecución de nuestro algoritmo en el cluster y hemos definido la estrategia para cumplir con el objetivo, en estos 4 pasos:

Paso 1.- Definir parámetros de optimización en spark-submit y el algoritmo.

Paso 2. Obtener un tamaño de problema razonablemente medible

Paso 3.- Elegir el conjunto de párametros que mejor cumpla nuestro objetivo.

Paso 4.- Toma de tiempo con dos componentes distintas.

•Paso 1.- Definir parámetros de optimización en spark-submit y el algoritmo.

A priori hemos determinado dos tipos de ajustes. Ajustes en los :

-parámetros del algoritmo:

batch_size: las muestras de datos se reorganizan en lotes para facilitar a los procesadores su cálculo de forma masiva y eficaz.

max_components: número de clases en los que clasifica los datos y que nos determina el fin de la ejecución del algoritmo.

slices: le pasamos el valor de ***repartition***(*numPartitions*) para que reorganice aleatoriamente los datos del *RDD* (datos cargados en la memoria global de Spark) y cree de forma balanceada el número de particiones que se le pasa como parámetro.

-parámetros de spark-submit:

--executor_memory: Cantidad de memoria a usar por procesador.

--driver-memory: Memoria para el driver.

--num_executors: número de *workernodes*, es decir, de *executors*.

--total-executor-cores: Total *cores* para todos los *executors*.

--executor-cores: Número de *cores* por *executor*.

--driver-cores NUM: *Cores* por *driver* (Por defecto: 1)

•Paso 2. Obtener un tamaño de problema razonablemente medible.

En un principio la idea era conseguir determinar un tamaño de problema adecuado para obtener conclusiones de la combinación de parámetros lanzada.

max_components = 11. Dado que la evolución que hemos visto en la salida del fichero del log *OUT que* escribe nuestro algoritmo, hay un salto en el número de componentes calculados, que suele ir, hasta donde hemos probado, de 7 a 11 y dado que 7 componentes, es demasiado temprano como para apreciar, en todos los casos, diferencias significativas, elegimos como tamaño del problema **11 componentes**.

Una vez determinado el tamaño, ya empezamos a combinar los parámetros de configuración para ver qué combinación era la más adecuada.

No obstante, después de varias ejecuciones, vimos que podíamos deducir mucho antes de que acabara la simulación que esa combinación era mala o buena, porque en un momento dado empezaban a mostrar una tendencia.
Lo que no se podía averiguar era cuándo esa tendencia iba a ocurrir.

Dado que el algoritmo es lineal esa tendencia iba a mantenerse en el futuro. Así que no hacía falta que en nuestra simulación acabáramos ese tamaño de problema porque podíamos obtener conclusiones antes y ahorramos mucho tiempo.

Esta decisión era válida sobre todo para un primer descarte.

•Paso 3.- Elegir el conjunto de párametros que mejor cumpla nuestro objetivo.

Después de varias ejecuciones, se iban fijando parámetros a su mejor valor. Y estas son las primeras conclusiones:

driver-memory: 4G como mejor valor en ejecuciones tempranas.

executor-memory: 4G como mejor valor en ejecuciones tempranas.

slices:

Los datos de nuestro problemas están almacenados físicamente en 88 bloques del sistema de ficheros HDFS.

Cuando cargamos estos datos en memoria, al RDD, las particiones que hacía eran, curiosamente 88.

Nuestra intención era cambiar el número de particiones de datos a 116. Tantas como cores tenemos en nuestro *cluster*. De esa forma cada core tendrá sus propios datos iniciales con los que trabajar.

Para cambiar las particiones del RDD a 116, modificamos el valor de ***slices*** a 116 pero no surtía efecto y seguían saliendo 88 particiones del RDD.

Insistimos pues tenemos la finalidad de poder enviar una partición de los datos a cada *core*, introdujimos el parámetro ***--total-executor-cores*** =116 en la llamada de spark-submit y, aunque inicialmente los carga tal y como los coge del HDFS, en 88 particiones; posteriormente, ya en el cálculo, los recoloca en 116 particiones.

22	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 17:14:02	0,8 s	1/1 (2 skipped)	116/116 (176 skipped)
21	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 17:14:01	0,9 s	1/1 (2 skipped)	116/116 (176 skipped)
20	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 17:14:00	0,8 s	1/1 (2 skipped)	116/116 (176 skipped)
19	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 17:13:59	0,8 s	1/1 (2 skipped)	116/116 (176 skipped)
18	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 17:13:58	0,9 s	1/1 (2 skipped)	116/116 (176 skipped)
17	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 17:13:57	0,8 s	1/1 (2 skipped)	116/116 (176 skipped)
16	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 17:13:56	0,8 s	1/1 (2 skipped)	116/116 (176 skipped)
15	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 17:13:56	0,8 s	1/1 (2 skipped)	116/116 (176 skipped)
14	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 17:13:55	0,8 s	1/1 (2 skipped)	116/116 (176 skipped)
13	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 17:13:54	1 s	1/1 (2 skipped)	116/116 (176 skipped)
12	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 17:13:53	0,9 s	1/1 (2 skipped)	116/116 (176 skipped)
11	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 17:13:52	1 s	1/1 (2 skipped)	116/116 (176 skipped)
10	count at /home/usulocal/python/machine_learning/MLE.py:143	2016/06/29 17:13:51	0,5 s	1/1 (2 skipped)	116/116 (176 skipped)
9	runJob at PythonRDD.scala:393	2016/06/29 17:13:51	0,5 s	1/1 (2 skipped)	1/1 (176 skipped)
8	runJob at PythonRDD.scala:393	2016/06/29 17:13:50	0,5 s	1/1 (2 skipped)	1/1 (176 skipped)
7	count at /home/usulocal/examples2/test_MLE_2.py:162	2016/06/29 17:13:30	20 s	1/1 (2 skipped)	116/116 (176 skipped)
6	runJob at PythonRDD.scala:393	2016/06/29 17:13:27	3 s	1/1 (2 skipped)	1/1 (176 skipped)
5	runJob at PythonRDD.scala:393	2016/06/29 17:13:23	3 s	1/1 (2 skipped)	1/1 (176 skipped)
4	runJob at PythonRDD.scala:393	2016/06/29 17:13:20	3 s	1/1 (2 skipped)	1/1 (176 skipped)
3	runJob at PythonRDD.scala:393	2016/06/29 17:11:34	1,8 min	3/3	177/177
2	runJob at PythonRDD.scala:393	2016/06/29 17:11:33	2 s	1/1	1/1
1	runJob at PythonRDD.scala:393	2016/06/29 17:11:33	78 ms	1/1	1/1
0	count at /home/usulocal/examples2/test_MLE_2.py:106	2016/06/29 17:10:00	1,5 min	1/1	88/88

Captura 18. Pantallazo del entorno de monitorización de Spark.

total-executor-cores: 116 como mejor valor. Tenemos 29 máquinas activas (había una estropeada) con 4 cores cada una, 116 cores en total.

batch_size: 10k ó 15k

Cambiando el tamaño de la muestra que inicialmente estaba 100 y aumentándolo, el tiempo de ejecución mejoraba.

Las muestras de datos que tenemos (***num_sample***) están agrupadas en ***batch_size.*** Como hemos visto las agrupamos en bloques múltiplo del número de particiones (***num_slices*** (116)) para enviarlo bien empaquetado a cada uno de los 116 *core*s, para lo cual modificó el código:

```
    """
        Load all the lines in a file (or files in a directory) into an RDD of text lines.
        - Each line represents a sample and  contain the same number of values.
    """
    text_lines = spark_context.textFile( dataset_filename )
    print( "file(s) loaded " )
    text_lines.persist()
    num_samples = text_lines.count()
    text_lines.unpersist()
    print( "loaded %d samples " % num_samples )
    """
        -Convert the text lines into numpy arrays.
        -Taking as input the RDD text_lines, a map operation is applied to each text line in order
          to convert it into a numpy array, as a result a new RDD of numpy arrays is obtained.
        -Nevertheless, as we need an RDD with blocks of samples instead of single samples, we
          associate with each sample a random integer number in a specific range.
        -So, instead of an RDD with of numpy arrays we get an RDD with tuples [ int, numpy.array ]
    """
    K = (num_samples + batch_size - 1) / batch_size
    K = ((K // slices)+1)*slices
    samples = text_lines.map( lambda line: ( numpy.random.randint(K), numpy.array( [ float(x) for x in line.split() ] ) ) )
    """
        Thanks to the random integer number used as key we can build a new RDD of blocks  of samples, where
        each block contains approximately the number of samples specified  in batch_size.
    """
    samples = samples.reduceByKey( lambda x, y: numpy.vstack( [ x, y ] ) )
    # Repartition if necessary
    if samples.getNumPartitions() < slices:
        samples = samples.repartition( slices )
        print( "rdd repartitioned to %d partitions" % samples.getNumPartitions() )
```

Captura 19. Fragmento fichero test_MLE_2.py: manipulando el tamaño de la muestras

Probemos el algoritmo, con los siguientes parámetros de lanzamiento fijos donde solo iremos variando el ***batch_size*** para obtener su mejor valor:

```
    spark-submit --master spark://eip6029.inf.upv.es:7077 \
                 --driver-memory 4G \
                 --executor-memory 4G \
                 --total-executor-cores 116 \
                 --py-files ${PYTHONPATH}/mypythonlib.tgz \
                 test_MLE_2.py  --num-slices 116  --batch-size 1000 --dataset ${dataset}
}
```

Captura *20. Fragmento del fichero run-test-MLE.sh: parámetros ajustados*

Se lanzaron ejecuciones, con los valores de ***--batch_size*** a 1.000, 2.000, 5000, 10.000 y 15.000.

Los mejores resultados nos lo dieron: 10K (=10.000) y 15k. Entre ellos no había mucha diferencia.

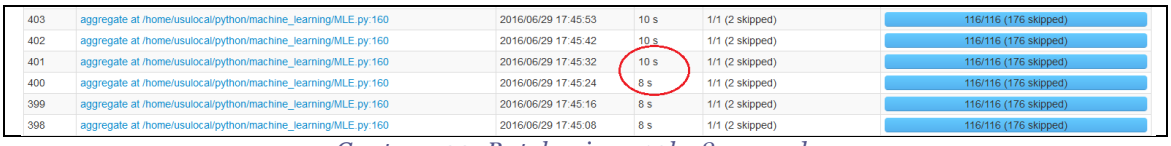
Veamos algunas capturas que ilustran el proceso con un *max_components* =11:

Completed Jobs (402)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
401	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 18:31:07	12 s	1/1 (2 skipped)	116/116 (176 skipped)
400	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 18:30:57	9 s	1/1 (2 skipped)	116/116 (176 skipped)
399	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 18:30:48	9 s	1/1 (2 skipped)	116/116 (176 skipped)
398	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 18:30:39	9 s	1/1 (2 skipped)	116/116 (176 skipped)

Captura 21. Batch_size= 2k: 9 segundos

Ese círculo rojo de la Captura 21 refleja que se ha calculado la componente 11 y empieza a calcular otra componente, lógicamente más costosa porque el algoritmo es lineal.



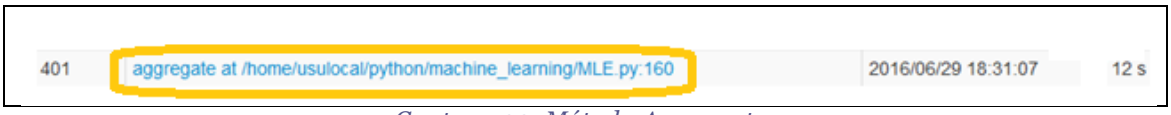
403	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 17:45:53	10 s	1/1 (2 skipped)	116/116 (176 skipped)
402	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 17:45:42	10 s	1/1 (2 skipped)	116/116 (176 skipped)
401	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 17:45:32	10 s	1/1 (2 skipped)	116/116 (176 skipped)
400	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 17:45:24	8 s	1/1 (2 skipped)	116/116 (176 skipped)
399	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 17:45:16	8 s	1/1 (2 skipped)	116/116 (176 skipped)
398	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 17:45:08	8 s	1/1 (2 skipped)	116/116 (176 skipped)

Captura 22. Batch_size= 10k: 8 segundos

Vemos que con *batch_size*=2k, a cada método *aggregate* le costaba 9 segundos. Con 10k, le costaba 8 segundos. Por tanto, batch_size= 10 minimiza mejor nuestro tiemo de ejecución.

Esta es la pista que comentabamos anteriormente que tenemos para determinar si vamos mejor o peor que otros algoritmos antes de llegar a la finalización de este.

Por otro lado, comentar que para el cálculo de una componente el algoritmo ejecuta iterativamente en el *cluster* únicamente el método *aggregate*, tantas veces como que el resultado de sus últimas 5 iteraciones sea menor a un valor dado, que entonces dá por buena la división de las clases.



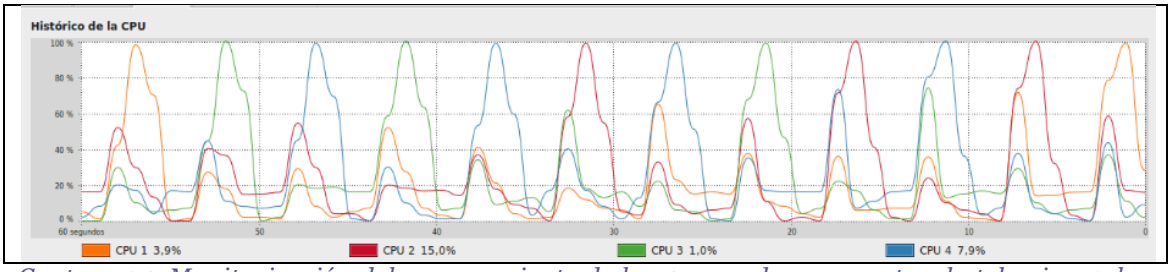
401	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 18:31:07	12 s

Captura 23. Método Aggregate.

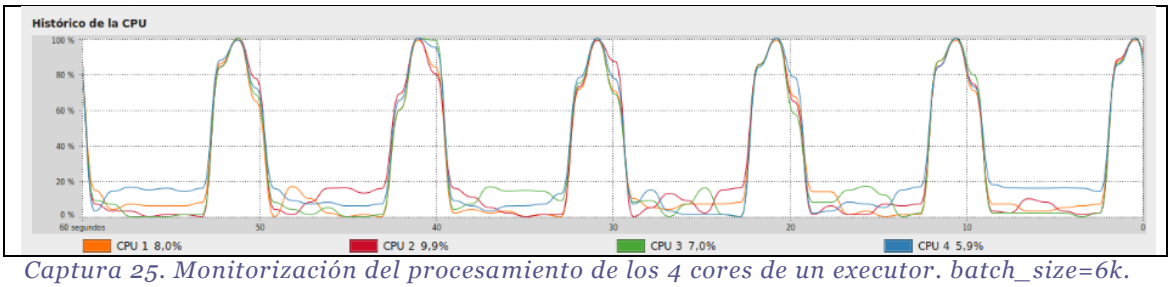
Seguimos con el estudi de batch_size:

Aunque con valores de 10k y 15k, el tiempo de ejecución era un poco mejor, nos dimos cuenta que la carga de procesamiento de cada *core* de los *executor* no era equilibrada, y, en tamaños de problemas cortos igual no es importante, pero conforme aumenten puede descompensar el cálculo y hacernos perder rendiemiento.

Por tanto, hemos ido en busca del equilibrio de esa carga.



Captura 24. Monitorización del procesamiento de los 4 cores de un executor. batch_size=15k.



Captura 25. Monitorización del procesamiento de los 4 cores de un executor. batch_size=6k.

Primeras conclusiones.

Los parámetros que han mejorado notablemente el tiempo de ejecución ha sido *batch_size,* executor-memory y *total-executor-cores*.

Con los demás parámetros y propiedades planteados no ha habido una mejora tan perceptible en el tiempo de ejecución.

Por tanto, los valores fijados de los parámetros los mostramos en la *Tabla 20*:

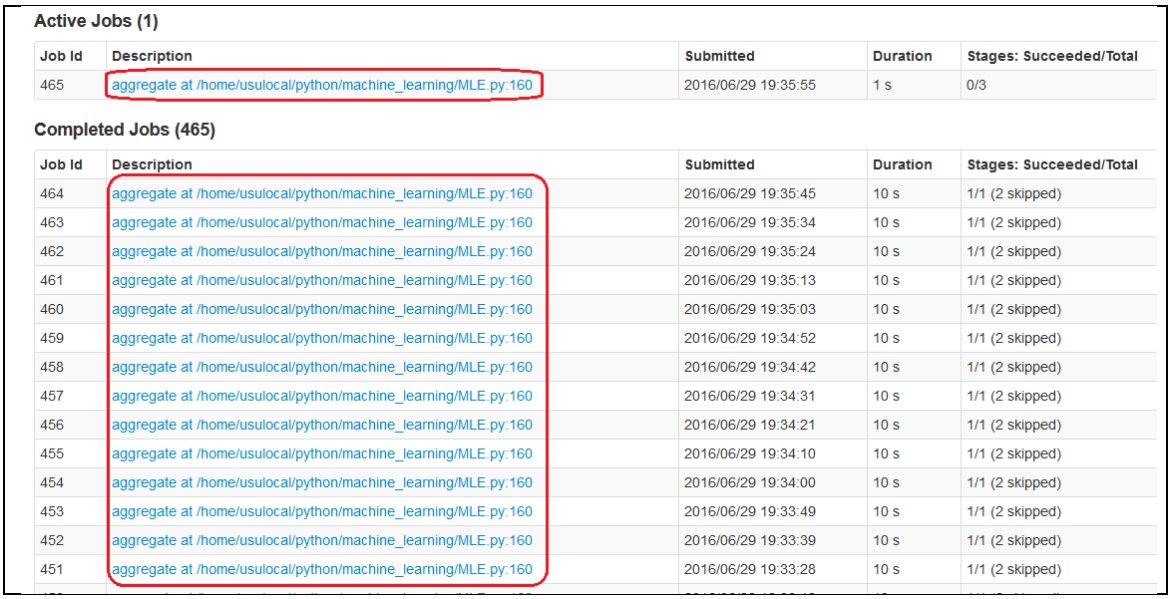
Propiedad	Valor	Explicación
--batch-size	6000	Tamaño de datos que equilibra la carga de las 4 CPU en los *datanodes*.
--max-components	12	Se ejecutaba hasta obtener 12 clases.
--num-slices	120	Dividimos los datos en 120 particiones, tantas como cores totales en cluster
--total-executor-cores	120	Utilizamos todos los cores del cluster
--excecutor_memory	4g	Valor que aumentándolo no mejoraba el tiempo de ejecución
--executor-cores	--	cores ya definidos en ámbito global. No necesitamos afinar por *executor*.
--num-executors	--	cores ya definidos en ámbito global. No necesitamos afinar número de *executors*.
--driver-memory	4g	No significativo, por el momento. Explicación más adelante.
--driver-cores NUM	--	No significativo, por el momento. Explicación más adelante.
--deploy_mode		No significativo, por el momento. Explicación más adelante.

Tabla 20. Tabla con los parámetros de lanzamiento al cluster seleccionados.

Nueva estrategia

No nos conformamos. Estas primeras propiedades han sido elegidas porque pensamos, de forma intuitiva, que podían influir en el tiempo de ejecución. Nos replanteamos la estrategia y con el bagaje de nuestras primeras conclusiones, nos centramos ahora en la ejecución del algoritmo en el cluster de *Spark*.

¿Qué trabajo hace *Spark*? *Spark* ejecuta iterativamente la línea 160 del fichero MLE.py.



Active Jobs (1)

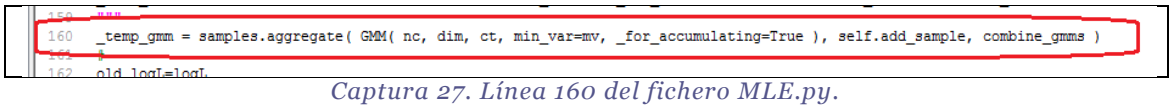
Job Id	Description	Submitted	Duration	Stages: Succeeded/Total
465	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 19:35:55	1 s	0/3

Completed Jobs (465)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total
464	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 19:35:45	10 s	1/1 (2 skipped)
463	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 19:35:34	10 s	1/1 (2 skipped)
462	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 19:35:24	10 s	1/1 (2 skipped)
461	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 19:35:13	10 s	1/1 (2 skipped)
460	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 19:35:03	10 s	1/1 (2 skipped)
459	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 19:34:52	10 s	1/1 (2 skipped)
458	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 19:34:42	10 s	1/1 (2 skipped)
457	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 19:34:31	10 s	1/1 (2 skipped)
456	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 19:34:21	10 s	1/1 (2 skipped)
455	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 19:34:10	10 s	1/1 (2 skipped)
454	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 19:34:00	10 s	1/1 (2 skipped)
453	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 19:33:49	10 s	1/1 (2 skipped)
452	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 19:33:39	10 s	1/1 (2 skipped)
451	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/06/29 19:33:28	10 s	1/1 (2 skipped)

Captura 26.Interfaz Web de Usuario de Spark. Pestaña Jobs.

¿Qué hace la línea 160 de MLE.py?



Captura 27. Línea 160 del fichero MLE.py.

La línea 160 hace uso de *aggregate*. ¿Qué es *aggregate*?

En *pySpark*, el *API* de *python* para *Spark*, tenemos la clase *RDD* (*Resilient Distributed Dataset*), que es una abstracción de *Spark* que representa una colección de elementos particionados que pueden ser tratado en paralelo. Y uno de sus métodos es *aggregate*, el cual: "Aggregate *the elements of each partition, and then the results for all the partitions, using a given combine functions and a neutral "zero value.*""

El *driver* compone la información que recibe de cada *executor* y una vez la ha ensamblado, las manda a todos ellos.

Cada *executor*, al recibirlo, hace los cálculos correspondientes a su parcela. Y lo envía al driver.

El *driver* vuelve a recibir cada "trozo" de datos que le llega de cada *executor*, y vuelve a recomponer todos los "trozos" y vuelve a enviarselo a todos ellos.

Y así una y otra vez.

Veamos si tiene correspondencia con los gráficos de la monitorización:

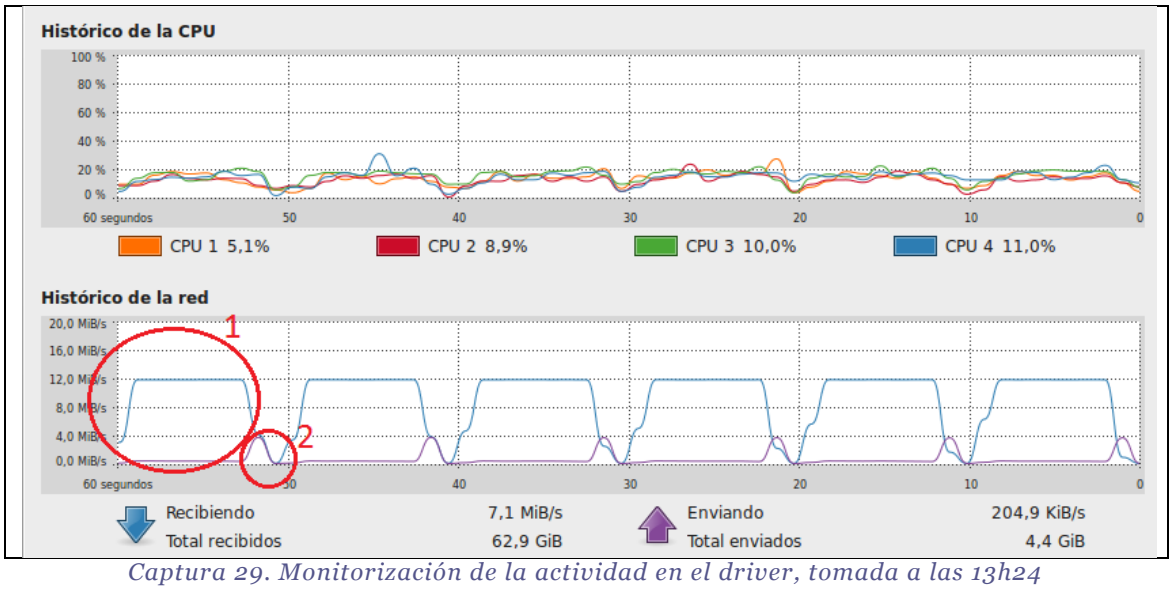
¿Qué pasa en los *executors*?

En el paso 1, el *executor* recibe los datos. En el 2, hace los cálculos de su porción y en el 3, lo envía al *driver*. Y se vuelven a repetir los mismos pasos de forma sistemática entendemos que por cada instrucción *aggregate*.

Captura 28. Monitorización de la actividad en el executor, tomada a las 13h24.

¿Qué pasa en el *driver*?

En el paso 1 recibe datos de los demás *executors*. Con lo recibido de cada *executor*, recompone los datos, sin apenas esfuerzo computacional de sus cores. Y en el paso 2 lo envía a todos los *executors*.



Captura 29. Monitorización de la actividad en el driver, tomada a las 13h24

¿Qué está pasando en el *driver*?

El *driver* tarda casi 8 segundos en cada paso 1 (recepción de los datos de todos los *executors*) debido a que llega al límite de la velocidad de la electrónica de red que , en nuestro *cluster* es, 100Mbit/s = 12 MByte/s.

Si no tuviéramos esta limitación tardaríamos menos y mejoraría nuestro tiempo de ejecución. ¿Cuánto menos?

Cada *aggregate*, a las 13h24 que es cuando se tomó la imagen de la monitorización del driver, *Spark* tardaba en realizar cada *aggregate*, 10 segundos, por lo que hay un recorrido muy importante de mejora.

373	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/07/07 13:24:54	10 s
372	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/07/07 13:24:44	10 s
371	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/07/07 13:24:34	10 s
370	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/07/07 13:24:24	10 s
369	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/07/07 13:24:14	10 s
368	aggregate at /home/usulocal/python/machine_learning/MLE.py:160	2016/07/07 13:24:04	10 s

Captura 30.Interfaz Web de Usuario de Spark. Pestaña Jobs a las 13h24.

Líneas de actuación

Cambios en el hardware: Cambiando la electrónica de red aumentaría notablemente la velocidad de transferencia entre los nodos de nuestro cluster.

• Paso 4.- Toma de tiempo con dos componentes distintas.

Con los parámetros ya elegidos, ejecutamos para obtener 7 y 11 clases y los resultados son:

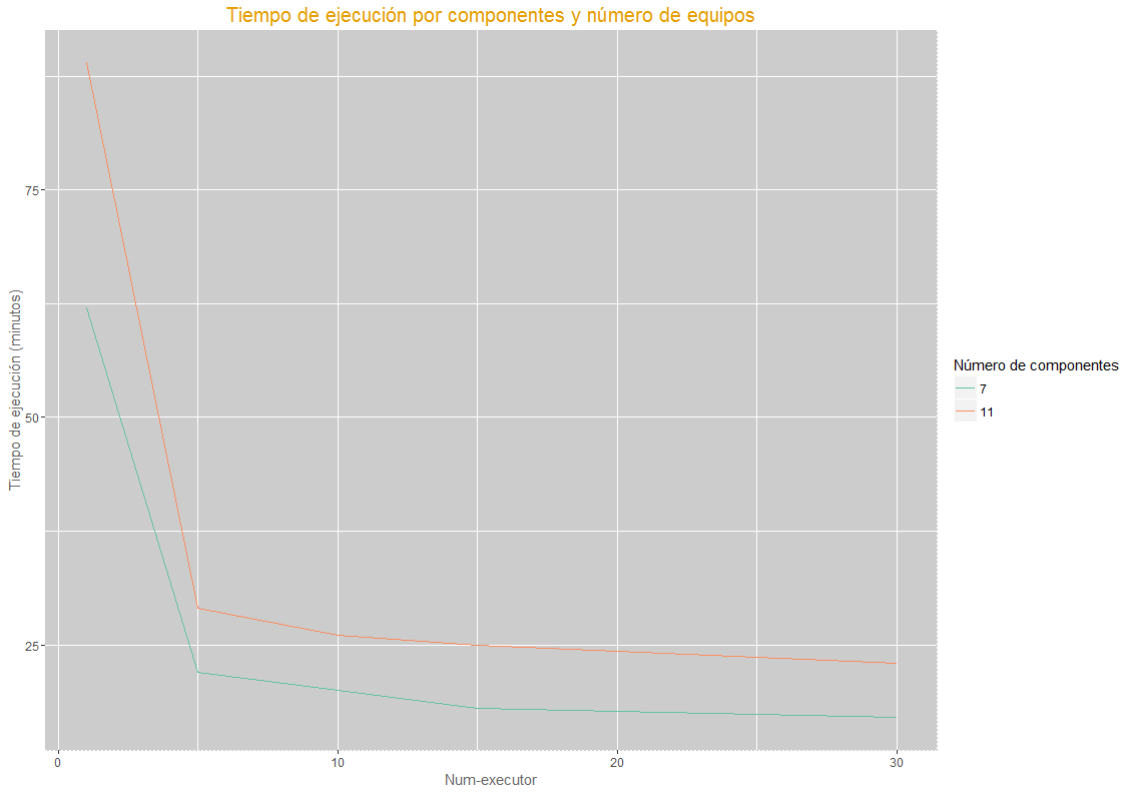


Gráfico 7. Tiempo de ejecución test_MLE_2.py (en minutos)-

Con los parámetros ya fijados tras el estudio, ejecutamos nuestro algoritmo con el cálculo del tiempo de ejecución con dos números de componentes distintas: 7 y 11 clases.

En la gráfica observamos que a mayor número de executors, menor tiempo de ejecución.

A partir de 15 *executors* no hay una mejoría notable, aunque quedaría por saber si cambiando la electrónica de red, el tiempo de ejecución, que disminuiría notablemente, mantendría las mismas proporciones entre los *executors* o no.

7. Conclusiones

Apasionante TFG donde he podido aprender mucho acerca de los contenidos aquí expresados.

He logrado acercarme a *Big Data* conociendo su universo de datos y las herramientas de las que hace uso, como, por ejemplo *Hadoop* y *Spark*.

Además, una vez realizado el TFG, creo que son las dos mejores herramientas que se pueden elegir del ecosistema de herramientas *Big Data* para empezar a aprender porque engloba en su esencia de lo que trata *Big Data*. *Hadoop* para poder almacenar datos que no caben en un ordenador y *Spark* para procesar datos, cuando no tienes suficiente memoria como para procesarlo en un ordenador.

A partir de ahí, es cuestión de seguir aprendiendo herramientas como por ejemplo, *Zookeeper*, que acude al rescate de *Hadoop* y bases de datos *NoSQL* proporcionando un sistema de coordinación que para estas aplicaciones distribuidas es muy útil pues hacen seguimientos como saber qué nodos están en el *cluster* o seleccionar un *master* cuando este falla, por ejemplo.

Por otro lado soy capaz de instalar y configurar un *cluster Hadoop + Spark* tanto con 32 ordenadores físicos como con 7 máquinas virtuales, que pienso que es una capacitación muy valiosa y más teniendo en cuenta que los cluster se construyen con *commodity hardware*, y va a ser más habitual que las infraestructuras caras y poco accesible a compañías no tan inmensas que hasta ahora había.

El hecho de probar un par de algoritmo en el *cluster* me ha hecho conocer la forma de trabajar con ellos en el *cluster* y cómo ajustarlos para que mejore su tiempo de ejecución.

Y ya no solamente para esta mejora en el tiempo de ejecución, sino el hecho de darle los recursos necesarios para que se ejecute en un tiempo razonable, a la vez que otros algoritmos se están ejecutando en el mismo cluster.

Queda mucho trabajo y experiencia por ganar en este terreno, pero como primer paso ha sido un buen primer paso.

Índice de Tablas

Índice de gráficos

Índice de ilustraciones

Índice de imágenes

Índice de capturas

Bibliografía

Capítulo 1 y 2

[1]. IBM Pc. Wikipedia. [online] [Fecha de consulta: 19-07-2016]. Disponible en: https://es.wikipedia.org/wiki/IBM_PC

[2]. IBM Pc. Wikipedia. [online] [Fecha de consulta: 19-07-2016]. Disponible en: https://es.wikipedia.org/wiki/Internet

[3]. Internet growth statistics. Internet world stats. [online] [Fecha de consulta: 20-08-2016]. Disponible en: http://www.internetworldstats.com/emarketing.htm

[4]. The Dawn of the Zettabyte Era [INFOGRAPHIC] by Thomas Barnett, Jr. blogs.cisco.com. [online] [Fecha de consulta: 20-08-2016]. Disponible en: http://blogs.cisco.com/news/the-dawn-of-the-zettabyte-era-infographic

[5]. *Big data*: 20 Mind-Boggling Facts Everyone Must Read, by Bernard Marr. Forbes.com. [online] [Fecha de consulta: 20-08-2016]. Disponible en: http://www.forbes.com/sites/bernardmarr/2015/09/30/big-data-20-mind-boggling-facts-everyone-must-read/#1a57bad86c1d

[6]. Data has always existed, the key is the right data, posted by George Psistakis. Data Science Central. [online] [Fecha de consulta: 20-07-2016]. Disponible en: http://www.bigdatanews.com/profiles/blogs/data-has-always-existed-the-key-is-the-right-data

[7]. ¿Aún hablas de '*big data*'? Estás obsoleto, por Elena Arrieta. Expansion.com. [online] [Fecha de consulta: 29-08-2016]. Disponible en: http://www.expansion.com/economia-digital/innovacion/2016/02/08/56b4f652268e3efa588b45f2.html

[8]. mumbrella360–What did we learn?, by Natasha Carroll. Red Agency. [online] [Fecha de consulta: 29-08-2016]. Disponible en: http://redagency.com.au/mumbrella360-what-did-we-learn/

[9]. Una nueva revolución industrial, por David Fernández. Elpais.com. [online] [Fecha de consulta: 29-08-2016]. Disponible en: http://economia.elpais.com/economia/2014/12/19/actualidad/1419006216_471179.html

[10]. José Hernández-Orallo. Apuntes: Challenges, goals, solutions and opportunities del *Master Big data* Analytics.

[11] Rubén Casado Tejedor. Apuntes: *Big data* y tiempo real del *Master Big data* Analytics.

[12]. Three *Big data* Trends for 2016, por John McCure. bigdatanews.com. [online] [Fecha de consulta: 29-08-2016]. Disponible en: http://www.bigdatanews.com/profiles/blogs/three-big-data-trends-for-2016

Capítulo 3.

[13]. What Is Apache *Hadoop*?. Apache.org. [online] [Fecha de consulta: 17-06-2016]. Disponible en: http://*Hadoop*.apache.org/

[14] **Tom White**. *Hadoop: the definitive guide 3ª Ed*. Editorial O'Reilly. ISBN: 978-1-449-31152-0.

[15] **Akhil Arora y Shrey Mehrotra**. *Learning Yarn*. Editorial Packt Publishing. ISBN: 978-1-78439-396-0.

[16]. *MapReduce*. Yahoo Developer Network. [online] [Fecha de consulta: 12-07-2016]. Disponible en: https://developer.yahoo.com/*Hadoop*/tutorial/module4.html

[17]. *MapReduce* tutorial. Apache *Hadoop*. [online] [Fecha de consulta: 12-07-2016]. Disponible en: https://*Hadoop*.apache.org/docs/r2.6.2/*Hadoop-MapReduce*-client/*Hadoop-MapReduce*-client-*core*/MapReduceTutorial.html

[18] **Arun C. Murthy y Vinod Kumar Vavilapalli**. *Apache Hadoop™ YARN*. Editorial O'Reilly. ISBN: 978-0-321-93450-5.

[19]. Programming with *Hadoop MapReduce*, por Vasilis Efthymiou. Computer Science Department University of Crete, Greece. [online] [Fecha de consulta: 15-07-2016]. Disponible en http://www.csd.uoc.gr/~hy562/labs/Frontistirio2.pdf

[20] **H*Old*en Karau, Andy Konwinski, Patrick Wendell y Matei Zaharia**. *Learning Spark*. Editorial O'Reilly. ISBN: 978-1-449-35862-4.

Capítulo 4:

[21] . Concepto de topología de red. Wikipedia [online][Fecha de consulta: 27-05-2016]. Disponible en: https://es.wikipedia.org/wiki/Topolog%C3%ADa_de_red

[22]. Descarga de *JAVA*. Oracle Corporation (US). [online] [Fecha de consulta: 27-05-2016]. Disponible en: https://www.*Java*.com/es/download/

[23]. Qué es JRE, *JVM* y JDK?, by Fausto Almeida Campos. Code-monkey [online] [Fecha de consulta: 26-05-2016]. Disponible en: http://coudmonqui.blogspot.com.es/2013/05/que-es-la-*JVM*-el-jre-y-el-jdk-y-con-que.html

[24]. *JVM* memory model, by Christophe. Coding Geek. A blog about IT, programming and *Java* [online][Fecha de consulta: 26-05-2016]. Disponible en: http://coding-geek.com/*JVM*-memory-model/

[25]. Encontrar el path de *Java*. Álvaro Lara Programación & Media. [online] [Fecha de consulta: 27-05-2016]. Disponible en: http://www.alvarolara.com/2015/08/24/como-saber-donde-esta-instalado-*Java*-en-ubuntu/

[26]. Manual OpenSSH. OpenSSH. [online] [Fecha de consulta: 04-06-2016]. Disponible en: http://www.openssh.com/manual.html

[27] . Configuración *Hadoop*, versión 2.6.2 [online][Fecha de consulta: 31-12-2015]. Disponible en: https://*Hadoop*.apache.org/docs/r2.6.2/

[28]. *Java* (*JVM*) Memory Model – Memory Management in *Java*, by Pankaj. JournalDev. [online] [Fecha de consulta: 26-07-2016].Disponible en: http://www.journaldev.com/2856/*Java*-*JVM*-memory-model-memory-management-in-*Java*

Anexos

Anexo 1: core-default.xml

El fichero *core*-default.xml contiene 175 propiedades. Una muestra son:

Name	***Value***	***description***
Hadoop.common.configuration.version	0.23.0	version of this configuration file
Hadoop.tmp.dir	/tmp/*Hadoop*-${user.name}	A base for other temporary directories.
io.native.lib.available	True	Should native *Hadoop* libraries, if present, be used.
Hadoop.http.filter.initializers	org.apache.*Hadoop*.http.lib.StaticUserWebFilter	A comma separated list of class names. Each class in the list must extend org.apache.*Hadoop*.http.FilterInitializer. The corresponding Filter will be initialized. Then, the Filter will be applied to all user facing jsp and servlet *web* pages. The ordering of the list defines the ordering of the filters.
Hadoop.security.authorization	False	Is service-level authorization enabled?
Hadoop.security.instrumentation.requires.admin	false	Indicates if administrator ACLs are required to access instrumentation servlets (JMX, METRICS, CONF, STACKS).
Hadoop.security.authentication	Simple	Possible values are simple (no authentication), and kerberos
Hadoop.security.group.mapping	org.apache.*Hadoop*.security.JniBasedUnixGroupsMappingWithFallback	Class for user to group mapping (get groups for a given user) for ACL. The default implementation, org.apache.*Hadoop*.security.JniBasedUnixGroupsMappingWithFallback, will determine if the *Java* Native Interface (JNI) is available. If JNI is available the implementation will use the *API* within *Hadoop* to resolve a list of groups for a user. If JNI is not available then the shell implementation, ShellBasedUnixGroupsMapping, is used. This implementation shells out to the *Linux*/Unix environment with the bash -c groups command to resolve a list of groups for a user.
Hadoop.security.groups.cache.secs	300	This is the config controlling the validity of the entries in the cache containing the user->group mapping. When this duration has expired, then the implementation of the group mapping provider is invoked to get the groups of the user and then cached back.
Hadoop.security.groups.negative-cache.secs	30	Expiration time for entries in the the negative user-to-group mapping caching, in seconds. This is useful when invalid users are retrying frequently. It is suggested to set a small value for this expiration, since a transient error in group lookup could temporarily lock out a legitimate user. Set this to zero or negative value to disable negative user-to-group caching.
Hadoop.security.groups.cache.warn.after.ms	5000	If looking up a single user to group takes longer than this amount of milliseconds, we will log a warning message.
Hadoop.security.group.mapping.ldap.url		The URL of the LDAP server to use for resolving user groups when using the LdapGroupsMapping user to group mapping.
Hadoop.security.group.mapping.ldap.ssl	false	Whether or not to use SSL when connecting to the LDAP server.
Hadoop.security.group.mapping.ldap.ssl.keystore		File path to the SSL keystore that contains the SSL certificate required by the LDAP server.
Hadoop.security.group.mapping.ldap.ssl.keystore.password.file		The path to a file containing the password of the LDAP SSL keystore. IMPORTANT: This file should be readable only by the Unix user running the daemons.
Hadoop.security.group.mapping.ldap.bind.user		The distinguished name of the user to bind as when connecting to the LDAP server. This may be left blank if the LDAP server supports anonymous binds.
Hadoop.security.group.mapping.ldap.bind.password.file		The path to a file containing the password of the bind user. IMPORTANT: This file should be readable only by the Unix user running the daemons.
Hadoop.security.group.mapping.ldap.base		The search base for the LDAP connection. This is a distinguished name, and will typically be the root of the LDAP directory.
Hadoop.security.group.mapping.ldap.search.filter.user	(&(objectClass=user)(sAMAccountName={0}))	An additional filter to use when searching for LDAP users. The default will usually be appropriate for Active Directory installations. If connecting to an LDAP server with a non-AD schema, this should be replaced with (&(objectClass=inetOrgPerson)(uid={0}). {0} is a special string used to denote where the username fits into the filter.
Hadoop.security.group.mapping.ldap.search.filter.group	(objectClass=group)	An additional filter to use when searching for LDAP groups. This should be changed when resolving groups against a non-Active Directory installation. posixGroups are currently not a supported group class.
Hadoop.security.group.mapping.ldap.search.attr.member	member	The attribute of the group object that identifies the users that are members of the group. The default will usually be appropriate for any LDAP installation.
Hadoop.security.group.mapping.ldap.search.attr.group.name	cn	The attribute of the group object that identifies the group name. The default will usually be appropriate for all LDAP systems.
Hadoop.security.group.mapping.ldap.directory.search.timeout	10000	The attribute applied to the LDAP SearchControl properties to set a maximum time limit when searching and awaiting a result. Set to 0 if infinite wait period is desired. Default is 10 seconds. Units in milliseconds.
Hadoop.security.service.user.name.key		For those cases where the same RPC protocol is implemented by multiple servers, this configuration is required for specifying the principal name to use for the service when the client wishes to make an RPC call.
Hadoop.security.uid.cache.secs	14400	This is the config controlling the validity of the entries in the cache containing the userId to userName and groupId to groupName used by NativeIO getFstat().
Hadoop.rpc.protection	authentication	A comma-separated list of protection values for secured sasl connections. Possible values are authentication, integrity and privacy. authentication means authentication only and no integrity or privacy; integrity implies authentication and integrity are enabled; and privacy implies all of authentication, integrity and privacy are enabled. *Hadoop*.security.saslproperties.resolver.class can be used to override the *Hadoop*.rpc.protection for a connection at the server side.
Hadoop.security.saslproperties.resolver.class		SaslPropertiesResolver used to resolve the QOP used for a connection. If not specified, the full set of values specified in *Hadoop*.rpc.protection is used while determining the QOP used for the connection. If a class is specified, then the QOP values returned by the class will be used while determining the QOP used for the connection.

Anexo 2: hdfs-default.xml

El fichero *HDFS*-default.xml contiene 214 propiedades. Una muestra son:

Nombre	Valor	Descripción
Hadoop.HDFS.configuration.version	1	version of this configuration file
dfs.*NameNode*.logging.level	info	The logging level for dfs *NameNode*. Other values are "dir" (trace namespace mutations), "block" (trace block under/over replications and block creations/deletions), or "all".
dfs.*NameNode*.rpc-address		RPC address that handles all clients requests. In the case of HA/Federation where multiple *NameNode*s exist, the name service id is added to the name e.g. dfs.*NameNode*.rpc-address.ns1 dfs.*NameNode*.rpc-address.EXAMPLENAMESERVICE The value of this property will take the form of nn-host1:rpc-port.
dfs.*NameNode*.rpc-bind-host		The actual address the RPC server will bind to. If this optional address is set, it overrides only the *hostname* portion of dfs.*NameNode*.rpc-address. It can also be specified per name node or name service for HA/Federation. This is useful for making the name node listen on all interfaces by setting it to 0.0.0.0.
dfs.*NameNode*.servicerpc-address		RPC address for *HDFS* Services communication. BackupNode, *DataNodes* and all other services should be connecting to this address if it is configured. In the case of HA/Federation where multiple *NameNode*s exist, the name service id is added to the name e.g. dfs.*NameNode*.servicerpc-address.ns1 dfs.*NameNode*.rpc-address.EXAMPLENAMESERVICE The value of this property will take the form of nn-host1:rpc-port. If the value of this property is unset the value of dfs.*NameNode*.rpc-address will be used as the default.
dfs.*NameNode*.servicerpc-bind-host		The actual address the service RPC server will bind to. If this optional address is set, it overrides only the *hostname* portion of dfs.*NameNode*.servicerpc-address. It can also be specified per name node or name service for HA/Federation. This is useful for making the name node listen on all interfaces by setting it to 0.0.0.0.
dfs.*NameNode*.secondary.http-address	0.0.0.0:50090	The secondary *NameNode* http server address and port.
dfs.*NameNode*.secondary.https-address	0.0.0.0:50091	The secondary *NameNode* HTTPS server address and port.
dfs.*DataNode*.address	0.0.0.0:50010	The *DataNode* server address and port for data transfer.
dfs.*DataNode*.http.address	0.0.0.0:50075	The *DataNode* http server address and port.
dfs.*DataNode*.ipc.address	0.0.0.0:50020	The *DataNode* ipc server address and port.
dfs.*DataNode*.handler.count	10	The number of server threads for the *DataNode*.
dfs.*NameNode*.http-address	0.0.0.0:50070	The address and the base port where the dfs *NameNode web* ui will listen on.
dfs.*NameNode*.http-bind-host		The actual adress the HTTP server will bind to. If this optional address is set, it overrides only the *hostname* portion of dfs.*NameNode*.http-address. It can also be specified per name node or name service for HA/Federation. This is useful for making the name node HTTP server listen on all interfaces by setting it to 0.0.0.0.
dfs.https.enable	false	Deprecated. Use "dfs.http.policy" instead.
dfs.http.policy	HTTP_ONLY	Decide if HTTPS(SSL) is supported on *HDFS* This configures the HTTP endpoint for *HDFS* daemons: The following values are supported: - HTTP_ONLY : Service is provided only on http - HTTPS_ONLY : Service is provided only on https - HTTP_AND_HTTPS : Service is provided both on http and https
dfs.client.https.need-auth	false	Whether SSL client certificate authentication is required
dfs.client.cached.conn.retry	3	The number of times the *HDFS* client will pull a socket from the cache. Once this number is exceeded, the client will try to create a new socket.
dfs.https.server.keystore.resource	ssl-server.xml	Resource file from which ssl server keystore information will be extracted
dfs.client.https.keystore.resource	ssl-client.xml	Resource file from which ssl client keystore information will be extracted
dfs.*DataNode*.https.address	0.0.0.0:50475	The *DataNode* secure http server address and port.
dfs.*NameNode*.https-address	0.0.0.0:50470	The *NameNode* secure http server address and port.
dfs.*NameNode*.https-bind-host		The actual adress the HTTPS server will bind to. If this optional address is set, it overrides only the *hostname* portion of dfs.*NameNode*.https-address. It can also be specified per name node or name service for HA/Federation. This is useful for making the name node HTTPS server listen on all interfaces by setting it to 0.0.0.0.
dfs.*DataNode*.dns.interface	default	The name of the Network Interface from which a data node should report its IP address.
dfs.*DataNode*.dns.nameserver	default	The host name or IP address of the name server (DNS) which a *DataNode* should use to determine the host name used by the *NameNode* for communication and display purposes.
dfs.*NameNode*.backup.address	0.0.0.0:50100	The backup node server address and port. If the port is 0 then the server will start on a free port.
dfs.*NameNode*.backup.http-address	0.0.0.0:50105	The backup node http server address and port. If the port is 0 then the server will start on a free port.
dfs.*NameNode*.replication.considerLoad	true	Decide if chooseTarget considers the target's load or not
dfs.default.chunk.view.size	32768	The number of bytes to view for a file on the browser.
dfs.*DataNode*.du.reserved	0	Reserved space in bytes per volume. Always leave this much space free for non dfs use.
dfs.*NameNode*.name.dir	file://${*Hadoop*.tmp.dir}/dfs/name	Determines where on the local filesystem the DFS name node should store the name table(*fsimage*). If this is a comma-delimited list of directories then the name table is replicated in all of the directories, for redundancy.
dfs.*NameNode*.name.dir.restore	false	Set to true to enable *NameNode* to attempt recovering a previously failed dfs.*NameNode*.name.dir. When enabled, a recovery of any failed directory is attempted during checkpoint.
dfs.*NameNode*.fs-limits.max-component-length	255	Defines the maximum number of bytes in UTF-8 encoding in each component of a path. A value of 0 will disable the check.
dfs.*NameNode*.fs-limits.max-directory-items	1048576	Defines the maximum number of items that a directory may contain. A value of 0 will disable the check.
dfs.*NameNode*.fs-limits.min-block-size	1048576	Minimum block size in bytes, enforced by the *NameNode* at create time. This prevents the accidental creation of files with tiny block sizes (and thus many blocks), which can degrade performance.
dfs.*NameNode*.fs-limits.max-blocks-per-file	1048576	Maximum number of blocks per file, enforced by the *NameNode* on write. This prevents the creation of extremely large files which can degrade performance.
dfs.*NameNode*.edits.dir	${dfs.*NameNode*.name.dir}	Determines where on the local filesystem the DFS name node should store the transaction (*edits*) file. If this is a comma-delimited list of directories then the transaction file is replicated in all of the directories, for redundancy. Default value is same as dfs.*NameNode*.name.dir
dfs.*NameNode*.shared.edits.dir		A directory on shared storage between the multiple *NameNode*s in an HA *cluster*. This directory will be written by the active and read by the standby in order to keep the namespaces synchronized. This directory does not need to be listed in dfs.*NameNode*.edits.dir above. It should be left empty in a non-HA *cluster*.
dfs.permissions.enabled	true	If "true", enable permission checking in *HDFS*. If "false", permission checking is turned off, but all other behavior is unchanged. *Switch*ing from one parameter value to the other does not change the mode, owner or group of files or directories.

Anexo 3: yarn-default.xml

El fichero *YARN*-default.xml contiene 185 propiedades. Una muestra son:

name	value	description
YARN.ipc.client.factory.class		Factory to create client IPC classes.
YARN.ipc.server.factory.class		Factory to create server IPC classes.
YARN.ipc.record.factory.class		Factory to create serializeable records.
YARN.ipc.rpc.class	org.apache.*Hadoop.YARN*.ipc.*HadoopYARN*ProtoRPC	RPC class implementation
YARN.ResourceManager.hostname	0.0.0.0	The *hostname* of the RM.
YARN.ResourceManager.address	${*YARN.ResourceManager.hostname*}:8032	The address of the applications manager interface in the RM.
YARN.ResourceManager.bind-host		The actual address the server will bind to. If this optional address is set, the RPC and webapp servers will bind to this address and the port specified in *YARN.ResourceManager*.address and *YARN.ResourceManager*.webapp.address, respectively. This is most useful for making RM listen to all interfaces by setting to 0.0.0.0.
YARN.ResourceManager.client.thread-count	50	The number of threads used to handle applications manager requests.
YARN.dispatcher.drain-events.timeout	300000	Timeout in milliseconds when *YARN* dispatcher tries to drain the events. Typically, this happens when service is stopping. e.g. RM drains the ATS events dispatcher when stopping.
YARN.am.liveness-monitor.expiry-interval-ms	600000	The expiry interval for application *master* reporting.
YARN.ResourceManager.principal		The Kerberos principal for the resource manager.
YARN.ResourceManager.scheduler.address	${*YARN.ResourceManager.hostname*}:8030	The address of the scheduler interface.
YARN.ResourceManager.scheduler.client.thread-count	50	Number of threads to handle scheduler interface.
YARN.http.policy	HTTP_ONLY	This configures the HTTP endpoint for *YARN* Daemons.The following values are supported: - HTTP_ONLY : Service is provided only on http - HTTPS_ONLY : Service is provided only on https
YARN.ResourceManager.webapp.address	${*YARN.ResourceManager.hostname*}:8088	The http address of the RM *web* application.
YARN.ResourceManager.webapp.https.address	${*YARN.ResourceManager.hostname*}:8090	The https adddress of the RM *web* application.
YARN.ResourceManager.resource-tracker.address	${*YARN.ResourceManager.hostname*}:8031	
YARN.acl.enable	false	Are acls enabled.
YARN.admin.acl	*	ACL of who can be admin of the *YARN cluster*.
YARN.ResourceManager.admin.address	${*YARN.ResourceManager.hostname*}:8033	The address of the RM admin interface.
YARN.ResourceManager.admin.client.thread-count	1	Number of threads used to handle RM admin interface.
YARN.ResourceManager.connect.max-wait.ms	900000	Maximum time to wait to establish connection to *ResourceManager*.
YARN.ResourceManager.connect.retry-interval.ms	30000	How often to try connecting to the *ResourceManager*.
YARN.ResourceManager.am.max-attempts	2	The maximum number of application attempts. It's a global setting for all application *master*s. Each application *master* can specify its individual maximum number of application attempts via the *API*, but the individual number cannot be more than the global upper bound. If it is, the *ResourceManager* will override it. The default number is set to 2, to allow at least one retry for AM.
YARN.ResourceManager.container.liveness-monitor.interval-ms	600000	How often to check that *containers* are still alive.
YARN.ResourceManager.keytab	/etc/krb5.keytab	The keytab for the resource manager.
YARN.ResourceManager.webapp.delegation-token-auth-filter.enabled	true	*Flag* to enable override of the default kerberos authentication filter with the RM authentication filter to allow authentication using delegation tokens(fallback to kerberos if the tokens are missing). Only applicable when the http authentication type is kerberos.
YARN.nm.liveness-monitor.expiry-interval-ms	600000	How long to wait until a node manager is considered dead.
YARN.ResourceManager.nodes.include-path		Path to file with nodes to include.
YARN.ResourceManager.nodes.exclude-path		Path to file with nodes to exclude.
YARN.ResourceManager.resource-tracker.client.thread-count	50	Number of threads to handle resource tracker calls.
YARN.ResourceManager.scheduler.class	org.apache.*Hadoop.YARN*.server.*ResourceManager*.scheduler.capacity.CapacityScheduler	The class to use as the resource scheduler.
YARN.scheduler.minimum-allocation-mb	1024	The minimum allocation for every container request at the RM, in MBs. Memory requests lower than this won't take effect, and the specified value will get allocated at minimum.
YARN.scheduler.maximum-allocation-mb	8192	The maximum allocation for every container request at the RM, in MBs. Memory requests higher than this won't take effect, and will get capped to this value.
YARN.scheduler.minimum-allocation-v*core*s	1	The minimum allocation for every container request at the RM, in terms of virtual *CPU core*s. Requests lower than this won't take effect, and the specified value will get allocated the minimum.
YARN.scheduler.maximum-allocation-v*core*s	32	The maximum allocation for every container request at the RM, in terms of virtual *CPU core*s. Requests higher than this won't take effect, and will get capped to this value.
YARN.ResourceManager.recovery.enabled	false	Enable RM to recover state after starting. If true, then *YARN.ResourceManager*.store.class must be specified.
YARN.ResourceManager.fail-fast	${*YARN*.fail-fast}	Should RM fail fast if it encounters any errors. By defalt, it points to ${*YARN*.fail-fast}. Errors include: 1) exceptions when state-store write/read operations fails.
YARN.fail-fast	false	Should *YARN* fail fast if it encounters any errors. This is a global config for all other components including RM,NM etc. If no value is set for component-specific config (e.g *YARN.ResourceManager*.fail-fast), this value will be the default.
YARN.ResourceManager.work-preserving-recovery.enabled	false	Enable RM work preserving recovery. This configuration is private to *YARN* for experimenting the feature.
YARN.ResourceManager.work-preserving-recovery.scheduling-wait-ms	10000	Set the amount of time RM waits before allocating new *containers* on work-preserving-recovery. Such wait period gives RM a chance to settle down resyncing with NMs in the *cluster* on recovery, before assigning new *containers* to applications.

Anexo 4: mapred-default.xml

El fichero mapred-default.xml contiene 205 propiedades. Una muestra son:

name	value	description
MapReduce.jobtracker.jobhistory.location		If job tracker is static the history files are stored in this single well known place. If No value is set here, by default, it is in the local file system at ${*Hadoop*.log.dir}/history.
MapReduce.jobtracker.jobhistory.task.numberprogresssplits	12	Every task attempt progresses from 0.0 to 1.0 [unless it fails or is killed]. We record, for each task attempt, certain statistics over each twelfth of the progress range. You can change the number of intervals we divide the entire range of progress into by setting this property. Higher values give more precision to the recorded data, but costs more memory in the job tracker at runtime. Each increment in this attribute costs 16 bytes per running task.
MapReduce.job.userhistorylocation		User can specify a location to store the history files of a particular job. If nothing is specified, the logs are stored in output directory. The files are stored in "_logs/history/" in the directory. User can stop logging by giving the value "none".
MapReduce.jobtracker.jobhistory.completed.location		The completed job history files are stored at this single well known location. If nothing is specified, the files are stored at ${*MapReduce*.jobtracker.jobhistory.location}/done.
MapReduce.job.committer.setup.cleanup.needed	true	true, if job needs job-setup and job-cleanup. false, otherwise
MapReduce.task.io.sort.factor	10	The number of streams to merge at once while sorting files. This determines the number of open file handles.
MapReduce.task.io.sort.mb	100	The total amount of buffer memory to use while sorting files, in megabytes. By default, gives each merge stream 1MB, which should minimize seeks.
MapReduce.map.sort.spill.percent	0.80	The soft limit in the serialization buffer. Once reached, a thread will begin to spill the contents to disk in the background. Note that collection will not block if this thresh*Old* is exceeded while a spill is already in progress, so spills may be larger than this thresh*Old* when it is set to less than .5
MapReduce.jobtracker.address	local	The host and port that the *MapReduce* job tracker runs at. If "local", then jobs are run in-process as a single *map* and *reduce* task.
MapReduce.local.clientfactory.class.name	org.apache.*Hadoop*.mapred.LocalClientFactory	This the client factory that is responsible for creating local job runner client
MapReduce.jobtracker.http.address	0.0.0.0:50030	The job tracker http server address and port the server will listen on. If the port is 0 then the server will start on a free port.
MapReduce.jobtracker.handler.count	10	The number of server threads for the JobTracker. This should be roughly 4% of the number of tasktracker nodes.
MapReduce.tasktracker.report.address	127.0.0.1:0	The interface and port that task tracker server listens on. Since it is only connected to by the tasks, it uses the local interface. EXPERT ONLY. Should only be changed if your host does not have the loopback interface.
MapReduce.cluster.local.dir	${*Hadoop*.tmp.dir}/mapred/local	The local directory where *MapReduce* stores intermediate data files. May be a comma-separated list of directories on different devices in order to spread disk i/o. Directories that do not exist are ignored.
MapReduce.jobtracker.system.dir	${*Hadoop*.tmp.dir}/mapred/system	The directory where *MapReduce* stores control files.
MapReduce.jobtracker.staging.root.dir	${*Hadoop*.tmp.dir}/mapred/staging	The root of the staging area for users' job files In practice, this should be the directory where users' *HOME* directories are located (usually /user)
MapReduce.cluster.temp.dir	${*Hadoop*.tmp.dir}/mapred/temp	A shared directory for temporary files.
MapReduce.tasktracker.local.dir.minspacestart	0	If the space in *MapReduce.cluster*.local.dir drops under this, do not ask for more tasks. Value in bytes.
MapReduce.tasktracker.local.dir.minspacekill	0	If the space in *MapReduce.cluster*.local.dir drops under this, do not ask more tasks until all the current ones have finished and cleaned up. Also, to save the rest of the tasks we have running, kill one of them, to clean up some space. Start with the *reduce* tasks, then go with the ones that have finished the least. Value in bytes.
MapReduce.jobtracker.expire.trackers.interval	600000	Expert: The time-interval, in miliseconds, after which a tasktracker is declared 'lost' if it doesn't send heartbeats.
MapReduce.tasktracker.instrumentation	org.apache.*Hadoop*.mapred.TaskTrackerMetricsInst	Expert: The instrumentation class to associate with each TaskTracker.
MapReduce.tasktracker.resourcecalculatorplugin		Name of the class whose instance will be used to query resource information on the tasktracker. The class must be an instance of org.apache.*Hadoop*.util.ResourceCalculatorPlugin. If the value is null, the tasktracker attempts to use a class appropriate to the platform. Currently, the only platform supported is *Linux*.
MapReduce.tasktracker.taskmemorymanager.monitoringinterval	5000	The interval, in milliseconds, for which the tasktracker waits between two cycles of monitoring its tasks' memory usage. Used only if tasks' memory management is enabled via mapred.tasktracker.tasks.maxmemory.
MapReduce.tasktracker.tasks.sleeptimebeforesigkill	5000	The time, in milliseconds, the tasktracker waits for sending a SIGKILL to a task, after it has been sent a SIGTERM. This is currently not used on *WINDOWS* where tasks are just sent a SIGTERM.
MapReduce.job.*Maps*	2	The default number of *map* tasks per job. Ignored when *MapReduce*.jobtracker.address is "local".
MapReduce.job.*reduces*	1	The default number of *reduce* tasks per job. Typically set to 99% of the *cluster*'s *reduce* capacity, so that if a node fails the *reduces* can still be executed in a single wave. Ignored when *MapReduce*.jobtracker.address is "local".
MapReduce.jobtracker.restart.recover	false	"true" to enable (job) recovery upon restart, "false" to start afresh
MapReduce.jobtracker.jobhistory.block.size	3145728	The block size of the job history file. Since the job recovery uses job history, its important to dump job history to disk as soon as possible. Note that this is an expert level parameter. The default value is set to 3 MB.
MapReduce.jobtracker.taskscheduler	org.apache.*Hadoop*.mapred.JobQueueTaskScheduler	The class responsible for scheduling the tasks.
MapReduce.job.reducer.preempt.delay.sec	0	The thresh*Old* in terms of seconds after which an unsatisfied mapper request triggers reducer preemption to free space. Default 0 implies that the *reduces* should be preempted immediately after allocation if there is currently no room for newly allocated mappers.
MapReduce.job.max.split.locations	10	The max number of block locations to store for each split for locality calculation.
MapReduce.job.split.metainfo.maxsize	10000000	The maximum permissible size of the split metainfo file. The JobTracker won't attempt to read split metainfo files bigger than the configured value. No limits if set to -1.
MapReduce.jobtracker.taskscheduler.maxrunningtasks.perjob		The maximum number of running tasks for a job before it gets preempted. No limits if undefined.
MapReduce.map.maxattempts	4	Expert: The maximum number of attempts per *map* task. In other words, *Framework* will try to execute a *map* task these many number of times before giving up on it.