EVOLUCIÓN DE LOS ENTORNOS *BIG DATA* Y LOS RETOS PARA EL ARQUITECTO DE DATOS

CARLOS GONZÁLEZ CANTALAPIEDRA
Big Data Architect
EVERLYN VERGARA SOLER
Big Data Consultant

Cuando se habla de entornos y aplicaciones *Big Data*, a menudo nos referimos a Hadoop (1). Se trata de una pieza fundamental en la evolución e implantación de este tipo de tecnologías en el entorno empresarial, pero si nos refiriésemos a *Big Data* como algo exclusivamente referente a Hadoop, nos estaríamos equivocando.

Hadoop fue creado por Doug Cutting para complementar la distribución Nutch, su motor de búsqueda basado en Apache Lucene en el año 2004.

En ese mismo año, Google publica su *paper* (2) sobre MapReduce. Google empezaba a tener problemas para poder indexar la web al ritmo al que ésta iba creciendo y necesitaba encontrar una solución para poder acometer el problema. Esta solución se basaba en un sistema de archivos distribuidos, que le proporcionaba la capacidad de poder procesar en paralelo grandes cantidades de información. La solución al problema es simple, pero a la vez brillante, ya que el sistema era capaz de distribuir la información a procesar en diferentes ordenadores que actuaban de forma independiente cada uno de ellos, pero a la vez formaban parte de un todo.

En el año 2006, Google publica los detalles sobre la tecnología que había desarrollado y esta es acogida con gran interés por la comunidad *Open Source*, que en septiembre del año 2007 libera la primera versión de Hadoop bajo licencia Apache.

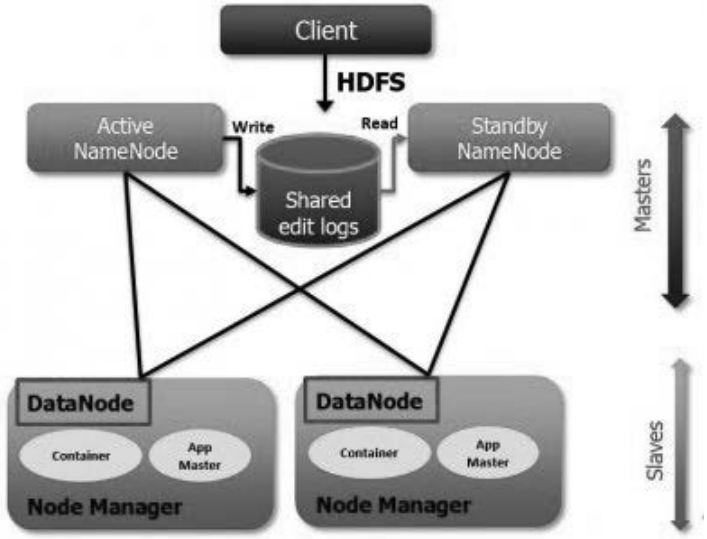
HADOOP, UNA PIEZA CLAVE EN EL ORIGEN DEL ECOSISTEMA *BIG DATA*

La primera versión de Hadoop estaba compuesta por dos componentes principales:

- HDFS: sistema de archivos distribuidos que permite de forma transparente a los desarrolladores almacenar un archivo de datos en diferentes máquinas de un *cluster* de Hadoop. Cada fichero que se almacena en HDFS se divide en bloques de un tamaño definido y es cada uno de estos bloques lo que se replica en diferentes nodos del *cluster*.
- MapReduce: *framework* de desarrollo que permite a los desarrolladores ejecutar procesos en paralelo dentro de un *cluster* de Hadoop abstrayendo la complejidad de dicho procesamiento en paralelo.

Una de las principales características de Hadoop que ha hecho que la tecnología se expanda es que puede ejecutarse sobre lo que se denomina «*commodity hardware*», es decir, máquinas baratas. Si a esto sumamos que Hadoop es *Open Source*, parece que es la combi-

FIGURA 1
ARQUITECTURA YARN



Fuente: https://www.edureka.co/blog/introduction-to-hadoop-2-0-and-advantages-of-hadoop-2-0/

nación perfecta para procesar grandes cantidades de información a un coste asumible para las empresas.

Una vez que la primera versión de Hadoop fue liberada, numerosas empresas empezaron a vislumbrar los beneficios del uso de esta tecnología. Facebook es una de estas primeras empresas que ven el beneficio en el uso de la tecnología, ya que en el año 2008 (3) se generan cantidades ingentes de información debido a las decenas de millones de usuarios que tiene la plataforma y a más del billón de páginas visitadas diariamente. Facebook necesita almacenar esta información para posteriormente analizarla y ver cómo mejorar la experiencia del usuario. Facebook empieza a utilizar el paradigma de programación MapReduce y desarrolla un *framework* para poder ejecutar sentencias SQL sobre datos que residen en HDFS. Este *framework* será liberado a la comunidad *Open Source* posteriormente con el nombre de Hive (4).

A partir de este momento, grandes compañías empiezan a hacer un uso intensivo de esta tecnología y a participar como contribuidores del core de Hadoop. Compañías como Facebook, eBay, IBM o Linkedin se convierten en contribuidores con más de 200.000 líneas de código.

LAS DISTRIBUCIONES COMO PARTE DE LA EVOLUCIÓN DEL ECOSISTEMA *BIG DATA*

En paralelo al desarrollo y evolución de Hadoop, empieza a crearse un ecosistema de productos que se van desarrollando en paralelo. Apache Hive, Apache Pig (5), Apache Zookeeper (6), Apache Sqoop (7) o Apache Oozie (8) son algunos ejemplos claros de cómo este ecosistema va creciendo y enriqueciendo el uso de la plataforma.

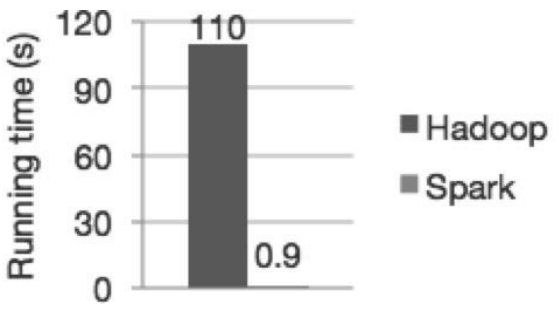
Todos estos nuevos productos nacen cada uno de ellos con un propósito muy concreto dentro del ecosistema. Por ejemplo, Apache Sqoop nace con el propósito de poder comunicar dos mundos muy distintos: las tradicionales bases de datos relacionales con Hadoop. Sqoop permite leer datos desde una base de datos relacional y escribir dichos datos en HDFS para que posteriormente sean procesados por otros productos del ecosistema. Y también permite la comunicación en el sentido contrario, leyendo datos de HDFS y escribiendo dichos datos en bases de datos relacionales.

Otro ejemplo es Apache Oozie, que permite la definición de flujos de trabajo, pudiendo estar escritos cada uno de ellos basándose en *frameworks* o productos diferentes. De esta manera, podemos definir en un solo flujo la lectura de datos con Apache Sqoop desde una base de datos relacional, procesarlos con Apache Hive, y posteriormente construir un *job* con Apache Spark (9) para escribir los resultados del proceso en una base de datos NoSQL como Apache Cassandra (10).

La mayoría de estos productos tienen dependencias con Apache Hadoop e incluso entre ellos mismos, de manera que cuando el ecosistema comienza a crecer, la configuración de estas dependencias entre los distintos componentes añade un grado más de dificultad en la instalación y mantenimiento de las plataformas.

Para reducir esta complejidad surgen nuevos actores alrededor del ecosistema Hadoop. Empresas como Cloudera (11) o Hortonworks (12) nacen con el propósito de crear distribuciones empaquetadas que faciliten la instalación, configuración y monitorización de *clusters* Hadoop y empiezan a incluir dentro de dichas distribuciones más productos del ecosistema que facilitan el almacenamiento y procesamiento de datos.

FIGURA 2
APACHE HADOOP VS APACHE SPARK



Fuente: http://spark.apache.org/images/logistic-regression.png

En enero de 2012 se produce un hito importante dentro de Hadoop. Se libera YARN (*Yet Another Resource Negotiator*) también conocido como MapReduce v2. YARN es una evolución de MapReduce y su propósito es desacoplar la gestión de los recursos de la planificación de las tareas. Por un lado, el ResourceManager se encarga de la gestión de los recursos a nivel global dentro del *cluster* y el ApplicationMaster se encarga de la planificación de las tareas por cada una de las aplicaciones que se ejecutan dentro del *cluster*. La Figura 1 ilustra la arquitectura de YARN.

A partir de este momento e impulsado en gran medida por las distribuciones de Cloudera y Hortonworks, el ecosistema de productos y las integraciones con entornos Hadoop crece de forma exponencial en el ámbito empresarial.

En este momento, Hadoop era un *framework* para realizar procesamiento de datos en batch, no formaba parte aún de este ecosistema una solución para el procesamiento de datos en *real time* o *near real time.*

APACHE SPARK Y PROCESAMIENTO EN *NEAR REAL TIME*

Apache Spark es un *framework open source* que permite el análisis de datos de forma distribuida y en memoria. La llegada de Apache Spark aporta al ecosistema *Big Data* capacidades de procesamiento de datos que hasta ese momento se podían conseguir combinando diferentes componentes, y no posible en todos los casos. Spark provee librerías para el procesamiento de datos en *near real time* (Spark Streaming), *machine learning* (Spark MLlib), grafos (GraphX) y SQL (SQL y DataFrames).

Pero aparte de todos estos módulos, Apache Spark aporta velocidad en el procesamiento de datos con respecto al *framework* MapReduce de Apache Hadoop. Esto conlleva a que muchas organizaciones comiencen a desarrollar sus aplicaciones sobre este *framework*, ya que las ventajas son obvias respecto a otros productos que existen en el mercado. Los *Jobs* basados en MapReduce, que podían durar horas o incluso días (dependiendo del volumen de datos a tratar y de la complejidad de los algoritmos), ven reducido drásticamente su tiempo de ejecución al utilizar este nuevo *framework*. Por ejemplo, un algoritmo de regresión lógica basado en Apache Hadoop y posteriormente basado en Apache Spark, arroja la comparativa en tiempos de ejecución que muestra la Figura 2.

Aparte de la velocidad, Apache Spark aporta una ventaja muy importante desde el punto de vista de un desarrollador de *software*. Cualquier desarrollador que haya trabajado con el *framework* MapReduce de Hadoop sabe que es complejo, y la llegada de Apache Spark aporta mayor facilidad en el desarrollo de aplicaciones de este tipo. Esta facilidad viene dada porque Spark provee diferentes APIs para Scala, Java y Python que abstraen al desarrollador de la complejidad a la que estaban acostumbrados al trabajar con el *framework* MapReduce. Y esto se traduce también en velocidad a la hora de construir aplicaciones.

Uno de los puntos fuertes de Spark es su librería de *streaming o real time*. Hay que puntualizar que sería más adecuado hablar de «*near real time*», ya que Apache Spark no proporciona verdadero tiempo real debido a la manera que gestiona la consumición de eventos o información en «tiempo real», ya que realmente lo que hace Apache Spark es la ejecución de procesos denominados *micro batch*. Estos procesos *micro batch* están constantemente en ejecución y leen información de la fuente cada un periodo de tiempo establecido (en el caso de Apache Spark, un periodo mínimo de 0.5 segundos).

Uno de los grandes aciertos de Apache Spark es su versatilidad para poder ejecutarse sobre distintas plataformas, entre ellas YARN. En el momento en el que aparece Apache Spark, muchas de las grandes empresas que han empezado a tener plataformas *Big Data*, basan dichas plataformas en *clusters* Hadoop.

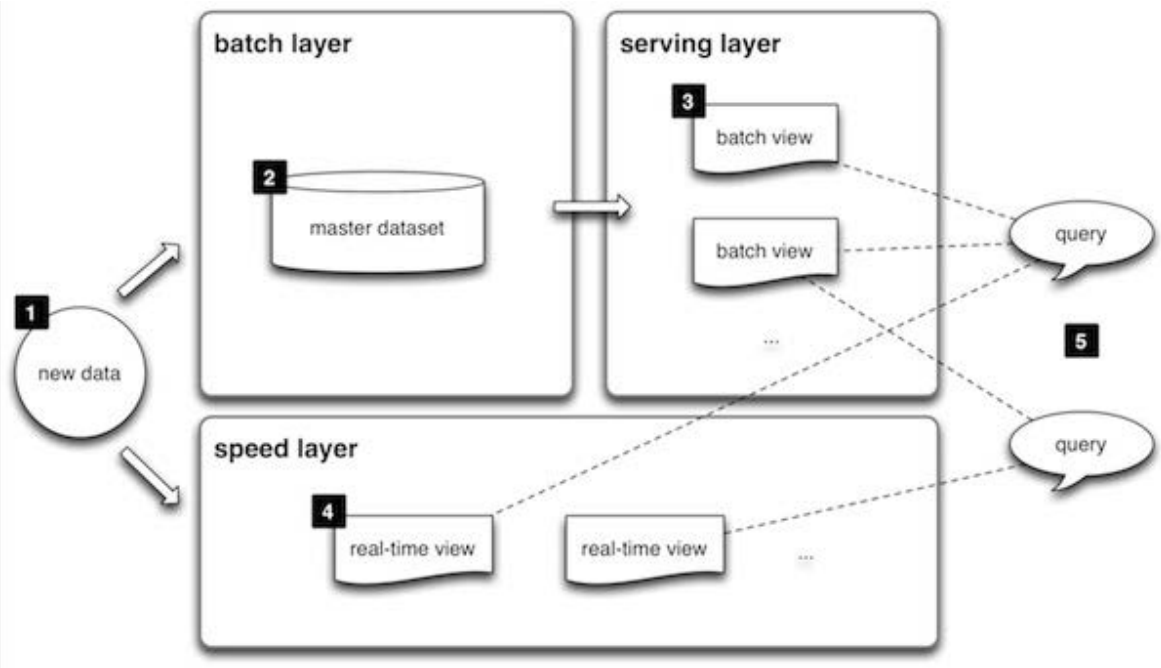
Al poder ejecutar Spark sobre YARN, las empresas no tienen que desprenderse de sus *clusters* Hadoop (y de las aplicaciones que se ejecutan sobre estos *clusters*), es más, lo ven como un nuevo elemento complementario y que les abre un nuevo abanico de posibilidades, sobre todo la posibilidad de procesar información en *near real time*, algo de lo que Hadoop carece de forma nativa.

La aparición de este tipo de *frameworks* -Apache Storm (13), Apache Spark, Apache Flink (14) o Apache Apex (15)- que permiten procesar información en tiempo real deriva en nuevos modelos de arquitectura, como por ejemplo la arquitectura Lambda (16).

ARQUITECTURAS *LAMBDA*

Una arquitectura *Lambda* está compuesta desde el punto de vista lógico por tres elementos principales: una capa *batch*, una capa de *real time* y una capa de *serving*. La Figura 3 muestra desde un punto de vista lógico dichas capas.

FIGURA 3
ARQUITECTURA *LAMBDA*



Fuente: http://lambda-architecture.net/img/la-overview_small.png

La capa *batch* y la capa de *real time* son complementarias, es decir, podemos procesar datos en *real time* y enriquecer con datos calculados en la capa *batch* los datos que vamos procesando de manera continua.

Este tipo de arquitecturas hacen que la «clásica» arquitectura de *clusters* basados en Hadoop ya no sea tan necesaria para ejecutar este tipo de aplicaciones. Si profundizamos un poco más, si nuestra plataforma no tiene ninguna necesidad de una capa *batch*, ¿para qué necesitamos Hadoop?

En efecto, si lo único que queremos hacer es procesar datos en tiempo real (esto es lo se denomina una arquitectura Kappa [17]) no necesitamos un *cluster* de Hadoop. Entonces, si no tenemos un *cluster* de Hadoop y estamos desarrollando por ejemplo nuestra plataforma basada en Apache Spark: ¿sobre qué ejecutamos Spark?; ¿cómo podemos conseguir escalabilidad horizontal?; ¿cómo automatizamos tareas de monitorización?.

En el caso de Apache Spark, tendríamos dos opciones sin un *cluster* de Hadoop (en realidad para Spark necesitaríamos YARN como hemos comentado en líneas anteriores): instalamos un *cluster* de Apache Spark (modo *standalone*) y lo gestionamos nosotros mismos o utilizamos otro tipo de plataforma.

«Configurar, desplegar, monitorizar, securizar o escalar» un *cluster* de Apache Spark puede llegar a ser un proceso complejo. Si por ejemplo necesitamos en un momento dado añadir más capacidad de proceso a ese *cluster* añadiendo nuevas instancias de Spark de forma automatizada, el proceso comienza a complicarse. Es en este momento dónde el rol de un arquitecto *Big Data* pasa a ser determinante porque la selección de la plataforma sobre la que vamos a desplegar y ejecutar nuestra aplicación, es uno de los puntos fundamentales a la hora de diseñar nuestra nueva arquitectura.

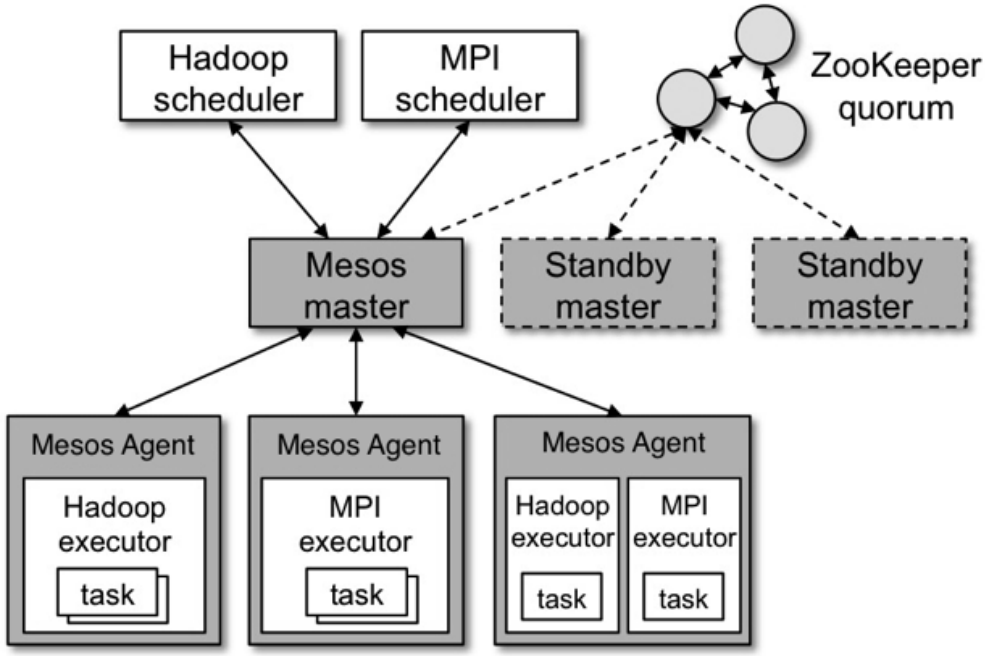
Desde el punto de vista de arquitectura, tenemos que buscar una plataforma que nos proporcione «capacidades de escalabilidad horizontal, monitorización, *scheduling* de tareas, recuperación automática de errores», etc. Si además la plataforma que seleccionemos es capaz de abstraernos de los recursos físicos (el *hardware* propiamente dicho) tales como memoria RAM, CPU y disco sería un punto a favor, ya que veríamos la plataforma de ejecución como un *pool* de recursos, no como un conjunto de máquinas.

APACHE MESOS, UNA NUEVA PIEZA EN EL ECOSISTEMA *BIG DATA*

El nacimiento de Apache Mesos (18) se remonta al año 2011 (19) como una evolución de *Nexus*, un proyecto de investigación que estaba llevando a cabo la universidad de Berkeley. Apache Mesos se define como «*a distributed systems kernel*», proporcionando una abstracción de los recursos físicos subyacentes de los recursos que serán asignados a nuestras aplicaciones. Sus principales características son las siguientes:

– Escalabilidad hasta 10.000 nodos.

FIGURA 4
ARQUITECTURA DE APACHE MESOS



Fuente: http://mesos.apache.org/assets/img/documentation/architecture3.jpgg

– Desarrollo de *frameworks* en diferentes lenguajes de programación.

– Soporte a contenedores Docker (20).

– Aislamiento de recursos entre los diferentes procesos que se ejecutan dentro del *cluster*.

– *Frameworks* disponibles.

– Alta disponibilidad de los componentes principales: *masters* y *slaves*.

La Figura 4 ilustra de forma gráfica la arquitectura de Apache Mesos.

Seleccionando un producto como Apache Mesos como la base de nuestra arquitectura tendremos nuevas capacidades desde el punto de vista del despliegue y la ejecución de nuestras aplicaciones, pero no hay que perder de vista los problemas que podríamos encontrar, si hablamos del mercado español:

- La falta de personal especializado en este tipo de plataformas. Actualmente es muy difícil encontrar en España profesionales que tengan el conocimiento y la experiencia práctica en el diseño, construcción, instalación y explotación de este tipo de plataformas.

- La mayoría de empresas que ya tienen algún tipo de plataforma *Big Data* (bancos, telecos y aseguradoras fundamentalmente) y están obteniendo valor de sus datos, basan sus plataformas en Apache Hadoop. Estas plataformas no tienen más de tres o cuatro años y en la mayoría de los casos supusieron una fuerte inversión para dichas empresas. La migración de sus plataformas *Big Data* a una tecnología como Apache Mesos supone asumir nuevas inversiones cuando, en muchos de los casos, todavía no han amortizado la anterior.

NUEVOS MODELOS DE NEGOCIO Y SERVICIOS PARA SOLUCIONES *BIG DATA*

En los últimos años han aparecido nuevos tipos de servicios en el mercado, especializados en proporcionar *Big Data* de una forma integral, en *cloud*. Es lo que se denomina «*Big Data as a Service*» (BDaaS).

Este tipo de plataformas integran en un solo servicio lo que conocemos como PaaS (*Platform as a service*), SaaS (*Software as a service*) e IaaS (*Infraestructure as a service*).:

– PaaS (*Platform as a service*): Proporciona la plataforma *Big Data*.

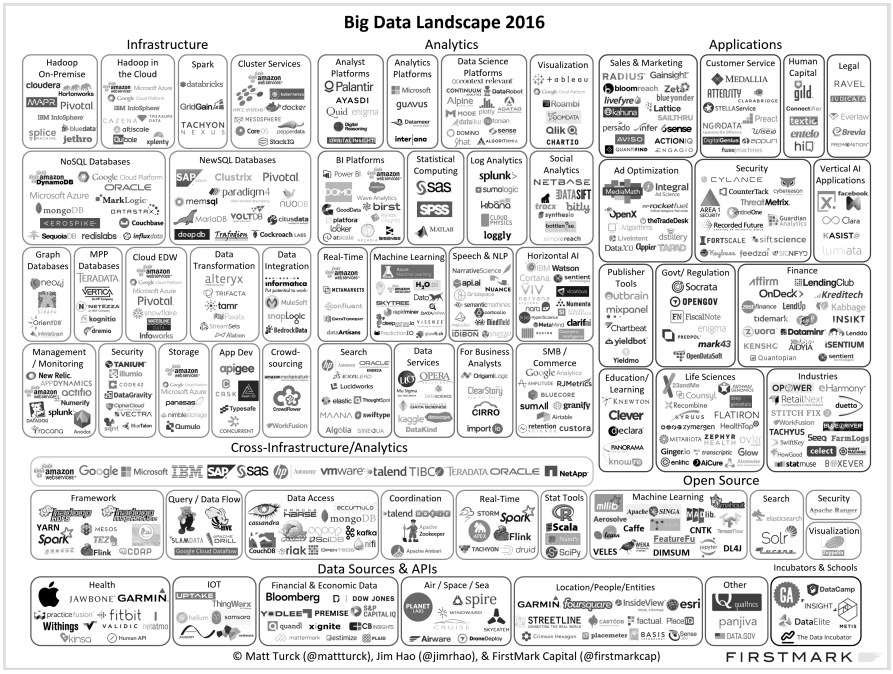
– SaaS (*Software as a service*): Es el *software* que se encargará de interactuar con el PaaS.

– IaaS (*Infraestructure as a service*): Nos proporciona el *hardware* que soportará nuestro PaaS e IaaS.

Las principales ventajas de utilizar una plataforma BDaaS son las siguientes:

1. Proporciona garantías en el nivel de servicio (SLAs): soporte por personal técnico altamente cualifica-

FIGURA 5
BIGDATA LANDSCAPE 2016



Fuente: http://mattturck.com/wp-content/uploads/2016/01/matt_turck_big_data_landscape_full.png

do, monitorización del servicio, actualizaciones de la tecnología y seguridad a nivel de plataforma.

2. Escalabilidad y alta disponibilidad de la plataforma asegurando la continuidad del servicio.

Empresas como Doopex (21) (empresa española pionera en este tipo de plataformas en Europa), Qubole (22) o Cazena (23) son un ejemplo claro de la expansión de este tipo de tecnologías *cloud*.

CLAVES PARA AFRONTAR EL RETO DEL DISEÑO DE LA ARQUITECTURA *BIG DATA*

Vamos a suponer un caso de negocio muy extendido y conocido por todos: una empresa de *eCommerce*, con un gran número de clientes y que vende productos a través de una plataforma web.

Se prevé que la plataforma tendrá un elevado volumen de transacciones, con múltiples formatos de entrada y requerimientos de procesamiento en tiempo real. Los conocidos retos del *Big Data* para todos, las tres V: Velocidad, Volumen y Variedad.

Esta plataforma tiene dos requerimientos funcionales que impactan en la toma de decisión de la arquitectura a implementar:

– La recomendación de productos en tiempo real.

– Predecir nuevas tendencias de consumo.

Si nos ceñimos exclusivamente al ámbito técnico, los arquitectos que tienen que diseñar la nueva arquitectura se encuentran ante un reto apasionante. Dado el actual *landscape* (24) de productos *Big Data*, una decisión errónea puede suponer que en el corto o medio plazo la arquitectura no sea la adecuada para los requerimientos iniciales, con las repercusiones en coste e impacto en el nivel de servicio que esto tendría.

La Figura 5 muestra las plataformas, *frameworks*, aplicaciones, productos de analítica, *machine learning*, visualización, etc. que podemos seleccionar para diseñar nuestra arquitectura. Como se puede observar, el abanico de posibilidades es inmenso.

El caso de negocio tomado como ejemplo requiere que, a nivel arquitectura, demos respuesta a dos aspectos:

- Disponer de un «repositorio» donde se almacenarán las compras que realizan los clientes.
- Utilizar un componente que sea capaz de monitorizar la navegación del usuario por la web, identificar los productos que consulta/compra y en ofrecerle recomendaciones en «tiempo real» lo más ajustadas posible a sus gustos.

Una «arquitectura Lambda» parece muy adecuada para soportar nuestro caso de uso, ya que la capa *batch* será la encargada de procesar toda la información histórica de compras que tenemos almacenada y la capa de *real time* procesará en tiempo real la navegación del usuario y podrá ofrecerle sugerencias de compra de productos.

Una vez que se ha decidido a alto nivel cual va a ser el modelo de arquitectura, hay que seleccionar las tec-

nologías e infraestructuras que serán la base del nuevo sistema.

Una importante decisión a tomar es si el nuevo sistema va a residir en un *data center* de la compañía (*on-premise*) o se desplegará en la nube (*cloud*).

La decisión sobre si elegir el *data center* de la compañía o la nube, dependerá en gran medida de la estrategia a nivel de IT de la compañía. Cada vez más compañías seleccionan la nube para desplegar sus sistemas, pero todavía existe cierta reticencia en el entorno empresarial para desplegar sus sistemas fuera de sus instalaciones. Fundamentalmente, estas decisiones se argumentan en base a la creencia de que el *cloud* no es seguro o sobre el desconocimiento acerca de dónde se encuentran almacenados realmente sus datos. Actualmente hay proveedores de *cloud* que ofrecen un nivel de servicio y garantizan un nivel de seguridad igual o mejor que el que muchas compañías pueden permitirse.

Una vez decidido dónde se desplegará la infraestructura del nuevo sistema, comienza el verdadero reto para un arquitecto *Big Data*: definir la tecnología que va a soportar la arquitectura Lambda antes seleccionada.

Como se ha mostrado anteriormente en la Figura 5, la variedad de productos actualmente disponibles en el mercado para diseñar e implementar una plataforma *Big Data* es muy amplia y variada, por lo que la selección de los componentes requiere por parte del profesional experiencia, análisis y conocimiento de las tecnologías que se van a seleccionar para la plataforma sobre la que el nuevo sistema se va a desplegar y ejecutar.

El diseño de la capa *batch* y en concreto del componente que se va a encargar de almacenar todos los datos históricos es el primer paso. Hemos indicado que se prevé gestionar un elevado volumen de datos y además, realizaremos la ingesta de información desde diferentes fuentes.

El repositorio (sistema de ficheros distribuidos) que se ha seleccionado para almacenar el *master dataset* es HDFS, ya que proporcionará la redundancia necesaria para asegurar que no se pierde ni un *byte* de información en caso de que se produzca un fallo en alguno de los nodos. Otra de las razones para seleccionar HDFS como repositorio, es la escalabilidad.

En un principio se debe realizar una estimación tanto del número de nodos donde se instalará el HDFS, como del espacio en disco que se va a necesitar, pero el producto seleccionado tiene que ser lo suficientemente flexible como para añadir más capacidad en cualquier momento, sin interrupciones del servicio que se esté prestando.

A día de hoy, la selección de HDFS como repositorio donde almacenar el *master dataset* es una de las más sencillas, ya que no hay muchas más alternativas en el universo *Big Data* y el funcionamiento del mismo, desde el punto de vista de rendimiento y escalabilidad, es aceptable.

Una vez seleccionado el producto donde se va a almacenar la información, se debe seleccionar un *framework* o producto que sea capaz de conectarse a HDFS, leer la información y procesarla. En este punto, el abanico de posibilidades es enorme (Spark, Hive, Flink, MapReduce, Impala (25), etc.), así como los factores a analizar para tomar la decisión (tiempos de respuestas, conectores disponibles para herramientas de visualización, la compatibilidad de las versiones, etc.) Se podría incluso, en función de la complejidad, llegar a seleccionar más de un producto que realice el procesamiento de información en esta capa de la arquitectura.

El objetivo final es obtener valor de nuestros datos, aplicando por ejemplo algoritmos de *machine learning* sobre ellos. También tenemos que tener en cuenta la velocidad de proceso que, aunque sean procesos *batch*, tampoco queremos que éstos tarden días o semanas en procesar la información. Tomando los dos puntos anteriores como variables de entrada para nuestra decisión, más el requerimiento de que los productos seleccionados tienen que ser *open source*, ya podemos realizar una selección inicial de *frameworks* y productos con los que empezar.

La selección inicial debe tener en cuenta, además de la propia funcionalidad que nos proporciona, aspectos como:

- Cuáles son los líderes del mercado.
- Lo maduro que es el producto.
- La comunidad de desarrolladores y el soporte que tiene el producto.
- Opciones de seguridad.
- El *roadmap* de cada producto (evolución de sus versiones).

En este punto, para el caso práctico que nos ocupa, se decide seleccionar Apache Spark y Apache Flink, para la capa pura de proceso. También vamos a evaluar Apache Hive como *framework* que nos va a permitir crear una estructura de datos que podría ser explotada desde otros productos o sistemas externos a nuestra arquitectura.

La primera recomendación es comenzar realizando diferentes pruebas de concepto con ambos productos, para evaluar cuál es el que mejor se adapta a las necesidades planteadas. Además de los propios algoritmos que el equipo de desarrollo va a implementar, hay que tener en cuenta la escalabilidad, la recuperación del sistema cuando se produzca un fallo, la monitorización y las diferentes formas de despliegue de ambos.

En esta fase, todavía no se ha decidido cuál va a ser la plataforma sobre la que se va a desplegar y ejecutar la arquitectura Lambda, pero teniendo en cuenta el requerimiento previo de que la plataforma tiene que ser

open source, podemos reducir el número de opciones a dos: utilizar un *cluster* de Apache Hadoop o utilizar un *cluster* de Apache Mesos. Esta es una decisión clave en el proceso y ambas tienen sus pros y sus contras:

Apache Hadoop: si decidimos utilizar un *cluster* de Apache Hadoop, significa que tanto Apache Spark como Apache Flink van a ser ejecutados sobre YARN. Pero a su vez, llegados a este punto, también tenemos dos opciones: podemos decidir utilizar alguna de las distribuciones que existen en el mercado o instalar nosotros mismos un *cluster* de Hadoop sin utilizar una distribución.

Las distribuciones de Hadoop líderes del mercado en estos momentos son Cloudera, Hortonworks y MapR (26).

Las distribuciones de Apache Hadoop han contribuido en gran medida a la expansión del *Big Data* en las empresas. La razón es sencilla: estas distribuciones son un empaquetado de Hadoop más numerosos productos del ecosistema *Big Data* (Apache Hive, Apache Oozie, Apache Kafka (27), etc.) y proporcionan facilidades en la instalación, configuración y monitorización de dichos *clusters*.

Parece claro que la utilización de una distribución de Hadoop puede ahorrarnos mucho tiempo en la instalación y configuración de la plataforma, y cuando nuestro sistema se encuentre en producción, nos proporcionarán funcionalidades de monitorización. En este punto, la idea de instalar nosotros mismos un *cluster* de Apache Hadoop queda descartada, ya que implicaría mucho más esfuerzo y no obtendríamos ningún beneficio claro.

Se ha descartar MapR ya que no incluye HDFS como parte de sus componentes. En lugar de HDFS, MapR incluye su propio sistema de ficheros denominado MapRFS, que proporciona un mejor rendimiento que HDFS, pero por el contrario es un sistema propietario y no *open source.*

Vamos a analizar las dos distribuciones de Hadoop candidatas a ser la base de nuestra plataforma:

- ✓ Cloudera: fue fundada en el año 2008 por ingenieros procedentes de Facebook, Google, Oracle y Yahoo, y la primera empresa en ofrecer un producto que empaquetaba Hadoop como una distribución. El *core* de la distribución es Apache Hadoop, pero Cloudera ha desarrollado sus propios componentes, como por ejemplo Cloudera *Manager*, que permite la instalación y configuración de la plataforma desde una interfaz web. Actualmente es la distribución que más clientes poseen (alrededor de 350) y es muy utilizada, por ejemplo, en el sector bancario español. La versión 5.9 (28) es la actual distribución de la plataforma, incluyendo Hadoop 2.6, Spark 1.6, Apache Hive 1.1.0, Apache Oozie 4.1.0 o Apache Sqoop 1.4.6. Si nos decidimos a utilizar Cloudera (en su versión *community*), tenemos que tener en cuenta que la versión de Spark es la 1.6 y que no incluye Apache Flink.
- ✓ Hortonworks: fundada en el año 2011 por ingenieros de Yahoo, es la única distribución donde todos sus componentes son puro *open source*. Esto significa que, en teoría, podemos sustituir cualquiera de los componentes que empaqueta por versiones más modernas o incluso modificar cierto código fuente de alguno de esos componentes en el caso de que fuera necesario, algo que desde mi punto de vista no deberíamos realizar a no ser que fuera algo realmente necesario, ya que, si lo hacemos, cuando actualicemos de versión, perderemos las modificaciones que hemos realizado.

La otra opción que tenemos es contribuir a la comunidad con dichas modificaciones o nuevas funcionalidades, y de esa manera, dentro de futuras versiones, nuestra contribución puede beneficiar a toda la comunidad (además de a nuestro propio sistema). La versión actual de Hortonworks es la 2.5 (HDP 2.5), que entre otros componentes, empaqueta (29) Apache Hadoop 2.7, Apache Spark 1.6 y 2.0 (2.0 en modo experimental, lo cual significa que no es *production-ready*), Apache Falcon 0.10.0, Apache Oozie 4.2.0, Apache Ranger 0.6.0 o Apache Hive 1.2.1. Al igual que sucedía con Cloudera, tenemos que tener en cuenta las versiones de Spark que incluye la distribución y que Apache Flink no está incluida todavía.

No existe una distribución mejor que otra, todo depende de los requerimientos que tengamos. Si la estrategia IT de nuestra compañía es utilizar productos puro *open source*, Hortonworks debería ser nuestra elección. A diferencia de Cloudera, Hortonworks (HDP) solamente tiene una distribución, que simplemente la podemos descargar de su web y comenzar con su instalación y configuración. Hortonworks proporciona diferentes servicios de soporte de pago, pero es algo totalmente opcional.

Por el contrario, Cloudera (CDH) ofrece diferentes distribuciones. La versión *Community* es completamente gratuita, pero tiene varias restricciones, como por ejemplo la administración de varios *clusters* de Hadoop, que con esta versión no es posible realizarlo, además de algunas restricciones en cuanto a la seguridad. La versión *Enterprise* si incluye la administración y sincronización de diferentes *clusters* además de la integración con varios sistemas de seguridad.

Nuestra arquitectura debe asegurar la continuidad del servicio en el caso de que nuestra instalación tenga algún tipo de problema que le impida proporcionar el servicio o incluso que el *data center* donde se encuentre desplegada caiga por completo.

Nuestra empresa de ejemplo tiene dos *data centers*, uno principal y otro de respaldo (*disaster recovery*). En este caso nuestra arquitectura deberá contemplar dos instalaciones diferentes, una que va a ser el despliegue activo y otra que será desplegada en el *data center* de respaldo en modo *stand-by*.

En este punto, la versión *Community* de Cloudera no cumpliría nuestros requisitos, y para lograr que los cumpliera, deberemos realizar desarrollos propios para mantener en sincronización ambos clusters (sobre todo en lo que a la sincronización de los datos residentes en HDFS se requiere).

Si por el contrario optamos por utilizar Hortonworks, este problema lo tendríamos resuelto con alguno de los productos que incluye la distribución (Apache Falcon [30]). Desde el punto de vista de administración, ambas distribuciones ofrecen una consola de gestión del *cluster*.

En el caso de Cloudera, disponemos de Cloudera Manager, que es un producto muy maduro y que proporciona toda la funcionalidad que podemos necesitar para la administración y configuración del *cluster*. En el caso de Hortonworks, Apache Ambari (31) ha sido el producto incluido en la distribución para llevar a cabo estas tareas. Si comparamos Cloudera Manager con Apache Ambari, Cloudera Manager sería el vencedor por su madurez y su facilidad de uso.

Utilizando cualquiera de las dos distribuciones, podemos tener básicamente dos problemas:

1. Si nos decidimos Apache Spark, tendremos que ceñirnos a las versiones que proporcionan las distribuciones. Podríamos incluir versiones superiores que ya existen en el mercado, pero no tendría demasiado sentido empezar a reemplazar componentes con el riego que conlleva en futuras actualizaciones de la distribución.

2. Si por el contrario nos decidimos por Apache Flink, tenemos un problema mayor. Ninguna de las distribuciones incluye este *framework*. Podríamos realizar la integración, pero en este caso no nos estaríamos aprovechando de las facilidades de administración y configuración que nos proporcionan las distribuciones.

3. La introducción de cualquiera de las dos distribuciones significa que en dicha plataforma solamente vamos a poder desplegar y ejecutar aplicaciones *Big Data*. Esto significa que los administradores tendrán que operar con un nuevo sistema en la compañía.

Los tres problemas anteriores pueden solventarse utilizando otro tipo de plataforma, **Apache Mesos**.

Apache Mesos nos proporciona la suficiente flexibilidad como para poder desplegar sobre el casi cualquier componente. Digo casi, porque por ejemplo la actual versión de Apache Flink no dispone de soporte para poder ejecutarse sobre Mesos. En el *roadmap* de Apache Flink ya está contemplado el soporte sobre Mesos [32], pero si tuviéramos que poner hoy un sistema basado en Apache Flink sobre un cluster de Apache Mesos con soporte nativo, dicho despliegue se convertiría en una ardua tarea.

Por el contrario, Apache Spark es uno de los *frameworks* que prácticamente desde el principio están incluidos en los que soporta Mesos. Aunque todavía no hemos hablado del resto de las capas de nuestra arquitectura Lambda (*speed layer* y *serving layer*), tenemos que empezar a pensar en ambas, ya que, si disponemos de una arquitectura base que soporte todos los componentes que ambas capas van a necesitar, sería un punto a su favor, sobre todo porque desplegaremos en un único entorno, lo que facilitará enormemente la forma en la que se despliegan nuestros componentes y tendremos que monitorizar y administrar un único entorno.

Otra de las ventajas que ofrece Mesos, es ser soporte nativo a contenedores Docker. Docker es un producto que permite crear imágenes de *software* muy ligeras y portables que pueden ser ejecutadas en cualquier sistema operativo con Docker instalado, independientemente del sistema operativo que nuestras máquinas tengan por debajo, facilitando enormemente los despliegues de los mismos. Otra ventaja de Docker es que los contenedores son autosuficientes. Esto quiere decir que el contenedor contiene todo el *software* necesario que necesita nuestra aplicación, que puede ir desde el propio sistema operativo hasta por ejemplo la versión específica de Java que necesitemos.

Esto significa que si por ejemplo decidimos utilizar Apache Spark 2.1 (última versión de Spark) porque incluye mejoras significativas desde el punto de vista del rendimiento y en los algoritmos de *machine learning* que proporciona su módulo MLLib con respecto a la versión 1.6.x, con Docker sería posible, ya que desplegaríamos los contenedores Docker que fuera necesarios sobre Mesos todos ellos con la versión de Spark que necesitamos.

Otro punto a favor sobre la utilización de Mesos es la escalabilidad. Cuando desplegamos una aplicación sobre Mesos, tenemos la facilidad de asignar de forma dinámica los recursos que nuestro *software* necesita independientemente del *hardware* subyacente. Mesos nos abstrae del *hardware* sobre el que nuestro *software* se ejecuta. Desde el punto de vista de nuestras aplicaciones solamente tenemos que indicar cuánta memoria, cuánta CPU y disco requiere nuestra aplicación, olvidándonos de si por debajo hay cinco o cincuenta servidores físicos. Mesos se va a encargar de proporcionarnos los recursos que estamos solicitando, y en el caso de que el *cluster* no los tenga disponibles, nos dejará en una cola esperando que los recursos solicitados sean liberados para que los podamos ejecutar.

Otra de las ventajas de Mesos sobre una distribución de Hadoop, es la ejecución de otro tipo que aplicaciones que no son *Big Data*. Sobre Mesos podemos ejecutar bases de datos NoSQL (Apache Cassandra por ejemplo), arquitecturas basadas en microservicios que se basan en AKKA (33) o en Spring Boot (34) o por ejemplo un cluster de Elasticsearch (35).

Esto significa que podemos tener una plataforma única para la ejecución, despliegue y monitorización de la mayor parte de las aplicaciones y sistemas de nues-

tra empresa. Esto se traduce directamente en el ahorro de costes, uniformidad en el despliegue, empaquetado de nuestras aplicaciones y monitorización de un único sistema.

Una vez que hemos diseñado nuestra capa *batch* y hemos seleccionado Mesos como plataforma de ejecución sobre la que instalar la solución y ejecutar las aplicaciones, tenemos que comenzar el diseño de la capa de procesado de información en tiempo real (también denominada capa de *streaming*).

Cuando hablamos de «tiempo real», tenemos que tener cuidado con lo que significa este término. Por ejemplo, si la base tecnológica de esta capa de la arquitectura la basamos en Apache Spark, sería más adecuado hablar de «*near real time*», ya que Apache Spark no proporciona verdadero tiempo real debido a la manera en que gestiona la consumición de eventos o información en «tiempo real», pues lo que realmente hace Apache Spark es ejecutar procesos denominados *micro batch*. Estos procesos *micro batch* están constantemente en ejecución y leen información de la fuente cada un periodo de tiempo establecido (en el caso de Apache Spark, un periodo mínimo de 0,5 segundos). Si nuestros requerimientos son de verdad *real time*, tendremos que buscar otro tipo de *framework* que nos proporcione la consumición de información en *streaming*, sin demora en el consumo de la información, incluso siendo esta demora de tan solo medio segundo. Para estos casos, tendríamos que utilizar un *framework* como Apache Flink, Apache Storm o Apache Apex, todos ellos bajo el paraguas de Apache al igual que Spark. La principal diferencia desde el punto de vista de la pura consumición de información en *streming* entre estos *frameworks* y Apache Spark, es que la información se consume de manera constante, sin interrupciones como en el caso de Apache Spark. Pero si medio segundo es un valor aceptable en la consumición de la información que vamos a procesar, Apache Spark es hoy por hoy el *framework* de referencia para utilizar en esta capa de la arquitectura.

Hemos hablado de la consumición de información en tiempo real, pero esa información tiene que estar almacenada en algún sitio para poder ser consumida con baja latencia. Es lo que conocemos como *bus* de datos o *brokers* de mensajería.

Un *broker* de mensajería es un sistema de almacenamiento que es muy rápido, tanto en escrituras como en lecturas, y permite la comunicación entre diferentes sistemas manteniendo un flujo de datos continuado entre los sistemas que escriben (denominados publicadores) y los sistemas que leen y procesa dichos datos (denominados subscriptores). Hay un producto en el mercado por encima de todos los demás cuando hablamos de este tipo de tecnologías: Apache Kafka.

Apache Kafka es un sistema de almacenamiento que sigue el patrón publicador/subscriptor que fue desarrollado originalmente por LinkedIn y posteriormente donado a la comunidad convirtiéndose en *open source*. Como principales características podemos destacar la baja latencia en la consumición de los datos por parte de los consumidores de los mismos, lo que en gran medida nos facilita el procesado de información en tiempo real. La escalabilidad de Apache Kafka permite a empresas como Netflix procesar 700 billones de mensajes diarios (36) llegando a perder solamente un 0,01% de dichos mensajes, lo cual es algo realmente impresionante y nos proporciona una visión de la escalabilidad y rendimiento que podemos llegar a alcanzar utilizando este tipo de tecnología.

Una vez que hemos seleccionado nuestros *frameworks* de referencia para la capa *batch* y la capa de *streaming* o *real time*, debemos definir lo que se denomina «*Serving Layer*». Esta capa de la arquitectura nos va a proporcionar los repositorios donde vamos a almacenar tanto los datos que procesamos desde la capa *batch* como la información que procesamos en tiempo real. Dependiendo del tipo de casos de uso que estemos implementando, la capa de real time podría no necesitar un repositorio de información donde almacenar el resultado del proceso, escribiendo directamente en un *broker* de mensajería el resultado de dicho proceso para que fuera consumido por otro sistema de nuestra compañía.

Desde el punto de vista de la capa *batch*, tenemos disponibles en el mercado diferentes productos *open source* donde almacenaremos el resultado de nuestros procesos *batch*. Desde el punto de vista lógico es lo que denominamos «*batch views*». Estas «*batch views*» contendrán vistas con información procesada, lista para ser consumida desde sistemas externos a nuestra arquitectura o desde la propia capa de tiempo real para enriquecer la información que estamos consumiendo en *streaming*.

El catálogo de productos *open source* que podemos utilizar dentro de esta capa de nuestra arquitectura es enorme. Productos como Apache HBase (37), Apache Cassandra o MongoDB (38) son solamente alguno de los productos *open source* que existen en el mercado y que podemos utilizar dependiendo del caso de uso o incluso podemos utilizar la combinación de varios de ellos dentro de esta capa.

CONCLUSIONES: EL MERCADO ESPAÑOL Y LOS ENTORNOS DE EJECUCIÓN PARA APLICACIONES *BIG DATA*

La estrategia empresarial (39) alrededor del dato por parte de las compañías españolas está tomando impulso en los últimos años.

Sectores como el financiero, teleco o los seguros llevan ventaja en España, pero para todos los sectores, aún la gestión y análisis de grandes volúmenes de datos sigue suponiendo un gran reto.

Para integrar estas tecnologías, las compañías españolas deben introducir cambios a dos niveles:

- Organizativo: creando nuevos departamentos y unidades de negocio.

- Técnico: Introducción de nuevas tecnologías en la empresa, lo que implica la contratación o formación de personal especializado.

Una compañía que decida comenzar el diseño y la construcción de una plataforma que sea capaz de procesar en tiempo real información para realizar comunicaciones relevantes a sus clientes, necesitaría:

– Arquitectos Big Data, personal que sea capaz de instalar la plataforma, la explotación de la misma.

– Programadores que sean capaces de construirla, actualmente un programador en España que tenga experiencia real con Apache Spark y que sepa programar en Scala, es un recurso muy escaso.

– Profesionales con la formación y experiencia práctica en la dirección y gestión de proyectos con tecnologías *Big Data*.

Todo este personal antes mencionado es escaso en el mercado español y de encontrarlo, tiene un coste muy elevado.

Todo lo anterior son factores a tener muy en cuenta cuando la organización decide que el tratamiento de grandes cantidades de información va a formar parte de su estrategia empresarial.

En apartados anteriores se ha descrito desde un punto de vista técnico y a alto nivel, los desafíos a los que un arquitecto de datos se enfrenta a la hora de diseñar una arquitectura *Big Data* y se ha intentado, a través de un ejemplo práctico, trasladar los retos que plantea la rápida evolución del mercado con las múltiples opciones en cuanto a la tecnología sobre la que vamos a basar nuestro diseño.

Como conclusión, resultado de la experiencia práctica, sólo cabe sugerir prestar especial atención a la hora de seleccionar los componentes y la integración entre los mismos cuando se acometa nuestro diseño, ya que una mala decisión de arquitectura puede llevarnos a que nuestra plataforma no sea capaz de escalar y evolucionar al ritmo que nuestra compañía nos exija.

NOTAS

[1] Apache Hadoop http://hadoop.apache.org/
[2] Google MapReduce paper https://static.googleusercontent.com/media/research.google.com/es/archive/mapreduce-osdi04.pdf
[3] Facebook Hadoop https://www.facebook.com/notes/facebook-engineering/hadoop/16121578919
[4] Apache Hive https://hive.apache.org/
[5] Apache Pig https://pig.apache.org/
[6] Apache Zookepeer https://zookeeper.apache.org/
[7] Apache Sqoop http://sqoop.apache.org/
[8] Apache Oozie http://oozie.apache.org/
[9] Apache Spark http://spark.apache.org/
[10] Apache Cassandra http://cassandra.apache.org/
[11] Cloudera https://www.cloudera.com/
[12] Hortonworks https://es.hortonworks.com/
[13] Apache Storm http://storm.apache.org/
[10] Apache Flink https://flink.apache.org/
[15] Apache Apex https://apex.apache.org/
[16] Lambda Architecture http://lambda-architecture.net/
[17] Kappa Architecture http://milinda.pathirage.org/kappa-architecture.com/
[18] Apache Mesos http://mesos.apache.org/
[19] Apache Mesos Paper https://people.eecs.berkeley.edu/~alig/papers/mesos.pdf
[20] Docker https://www.docker.com/
[21] Doopex Big Data as a Service https://doopex.com/
[22] Qubole Cloud Data Platform https://www.qubole.com/
[23] Cazena http://www.cazena.com/
[24] Big Data Landscape 2016 http://mattturck.com/wp-content/uploads/2016/01/matt_turck_big_data_landscape_full.png
[25] Apache Impala https://impala.incubator.apache.org/
[26] Mapr Converged Data Platform https://www.mapr.com/
[27] Apache Kafka https://kafka.apache.org/
[28] Cloudera 5.9.x Packaging and Tarball Information https://www.cloudera.com/documentation/enterprise/release-notes/topics/cdh_vd_cdh_package_tarball_59.html#topic_3
[29] Hortonworks 2.5.3 Components version https://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.5.3/bk_release-notes/content/ch_relnotes_v253.html
[30] Apache Falcon https://falcon.apache.org/index.html
[31] Apache Ambari https://ambari.apache.org/
[32] Flink and Apache mesos support https://ci.apache.org/projects/flink/flink-docs-release-1.2/setup/mesos.html
[33] Akka http://akka.io/
[34] Spring Boot https://projects.spring.io/spring-boot/
[35] Elasticsearch https://www.elastic.co/
[36] Netflix Kafka Inside Keystone Pipeline http://techblog.netflix.com/2016/04/kafka-inside-keystone-pipeline.html
[37] Apache HBase https://hbase.apache.org/
[38] MongoDB https://www.mongodb.com/
[39] Big Data, asignatura obligada en la estrategia empresarial http://www.datacentermarket.es/tendencias-tic/noticias/1094612032809/big-data-asignatura-obligada-en-la-estrategia-empresarial.1.html