Big Data: tecnologías y benchmarks

Resumen

El término Big Data se ha vuelto muy popular, pudiendo encontrarlo a menudo en múltiples noticias de la prensa tecnológica. Estas noticias destacan principalmente dos aspectos de Big Data: la gran cantidad de datos a analizar y las posibles aplicaciones. Sin embargo, las tecnologías subyacentes son menos conocidas.

En este trabajo presentamos una introducción a Big Data destacando el papel que ejerce en este ámbito la ingeniería de servidores. En primer lugar, motivamos y definimos el concepto de Big Data, mostrando la necesidad de desarrollar nuevas tecnologías que permitan abordar problemas de este campo. El desarrollo de estas tecnologías se encuentra ligado a la ingeniería de servidores y a la computación de altas prestaciones. Posteriormente, explicamos map reduce, un nuevo paradigma de programación distribuida, y varias de las tecnologías que lo utilizan más importantes del estado del arte (Hadoop, Spark y Flink). Por último, introducimos TPCx-HS, un benchmark para máquinas que usen Hadoop, y destacamos el trabajo que queda por hacer en este ámbito.

Palabras clave— Big Data, map reduce, benchmarks, ingeniería de servidores

1. Introducción

Desde hace miles de años el ser humano ha investigado la manera de almacenar y recopilar información. Durante muchos siglos la escritura y la pintura eran los únicos mecanismos existentes. Posteriormente surgió la fotografía, los discos de vinilo... Sin embargo, poca información seguía ocupando mucho volumen físico. Gracias a los avances tecnológicos de las últimos décadas, hoy en día disponemos dispositivos electrónicos para el almacenamiento de datos binarios. Además, la evolución de estos dispositivos ha sido frenética. IBM comercializó el primer disco duro en 1956. Este constaba solamente de 5 mega bytes de capacidad [10] mientras que actualmente podemos utilizar discos duros con más de 1 tera byte.

La capacidad de cómputo y procesamiento de los computadores también ha crecido de forma exponencial. El primer ordenador comercial se presentó en 1951 y se conoce como UNIVAC 1 [18]. Este computador tenía una frecuencia de $2{,}25$ mega hercios y 1000 palabras de memoria principal o RAM [25]. Actualmente utilizamos procesadores con más de 2 giga hercios de frecuencia de reloj. Además, es habitual encontrar computadores con 8 o más giga bytes de memoria principal, lo que permite trabajar con bastante información de forma eficiente.

Estas nuevas tecnologías han posibilitado que el almacenamiento de datos sea mucho más sencillo. Podemos guardar multitud de archivos en un dispositivo de unos centímetros y compartirlos con cualquier usuario. Además, el procesamiento de estos archivos a nivel de usuario es eficiente gracias a la velocidad de los computadores.

El mayor flujo de datos es producido en Internet. Aunque es relativamente joven, se hizo público en 1993, actualmente existen más de mil millones de páginas webs [24]. Además, multitud de dispositivos electrónicos se conectan e interaccionan con Internet (lo que se denomina Internet de las cosas [26]). Los usuarios de estos dispositivos utilizan aplicaciones web y redes sociales, publicando textos y archivos multimedia.

Todo este cúmulo de tecnologías y actividades ha dado lugar a que hoy en día haya más de 10 zeta bytes de información almacenados (1 ZB = 10^{12} GB). La Figura 1 muestra la evolución histórica de la cantidad de información acumulada por el ser humano. Podemos observar que cada año se generan varios zeta bytes de información, el crecimiento es exponencial. Hasta 2003 se habían almacenado en

total 5 exa bytes de información (1 EB = 10^9 GB). Actualmente, generamos esta cantidad de datos en dos días [21]. Podemos decir que vivimos en una sociedad digital o sociedad de la información.

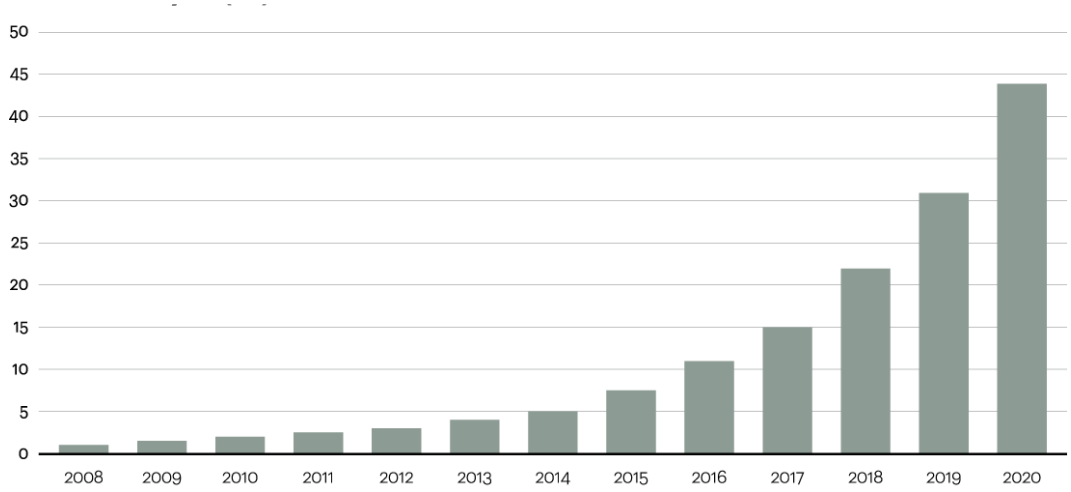

Figura 1: Evolución histórica del número de Zeta Bytes de información almacenados y predicción para los próximos años [9].

En resumen, a pesar de la evolución de los computadores en todas sus facetas, la cantidad de datos e información a procesar y almacenar crece incluso a mayor velocidad. Por ejemplo, encontramos empresas como Facebook y Google cuyos usuarios generan una gran cantidad de datos diariamente. Estos datos deben ser tratados en tiempo real para poder mantener los sistemas de recomendaciones asociados. En estos casos se requieren servidores de altas prestaciones para poder tratar tal cantidad de datos en un tiempo razonable. Estos conjuntos de datos adquieren el calificativo “masivos”. El desarrollo de nuevas tecnologías, algoritmos y herramientas para tratar de forma eficiente conjuntos de datos masivos es lo que se conoce como Big Data [14].

En este trabajo presentamos una introducción a Big Data y a las diferentes herramientas software existentes, destacando el papel de la ingeniería de servidores en este contexto. En particular, incidimos en la importancia del desarrollo de nuevos benchmarks que permitan evaluar las tecnologías de Big Data de forma clara y objetiva.

El resto del texto se organiza como sigue. La Sección 2 contiene una descripción del concepto de Big Data. En la Sección 3 introducimos el paradigma de programación map reduce y la tecnología libre que lo implementa, denominada Hadoop. En la Sección 4 describimos las nuevas tecnologías que han surgido para cohesionar la filosofía map reduce con el procesamiento iterativo. Estas se denominan Spark y Flink. En la Sección 5 explicamos uno de los benchmarks existentes para las tecnologías Big Data, denominado TPCx-HS. Por último, en la Sección 6 presentamos las conclusiones obtenidas.

2. Big Data

En esta sección profundizamos en el concepto de big data y en su relación con la ingeniería de servidores. En primer lugar, en la Sección 2.1 describimos qué es Big Data y qué tipos de problemas involucra. Además, destacamos la importancia de Big Data en el ámbito empresarial y científico. Por último, en la Sección 2.2 mostramos que la ingeniería de servidores es esencial en Big Data debido al uso de los servidores de altas prestaciones para el tratamiento de conjuntos de datos masivos.

2.1. ¿Qué es Big Data?

En la Sección 1 hemos incidido en la rápida evolución que han sufrido los computadores. Sin embargo, en muchas aplicaciones encontramos conjuntos de datos que ponen a prueba a los ordenadores más pontentes. Estos conjuntos de datos se denominan masivos. El tratamiento de datos masivos tiene dos problemas claros. El primero de ellos es la memoria. Necesitamos computadores con suficiente memoria principal para procesar de forma eficiente un gran flujo de datos. El segundo problema es la capacidad de cómputo necesaria para aplicar algoritmos sobre estos conjuntos de datos.

Big Data engloba el tratamiento de datos masivos desde el punto de vista tecnológico y algorítmico. En palabras de Michael J. Franklin, profesor de informática en la universidad de Berkley [27]:

"Un problema sobre datos entra en el ámbito de Big Data cuando la aplicación de las actuales tecnologías no permite al usuario obtener soluciones rápidas, efectivas en costo y de calidad"

Por tanto, Big Data es un concepto relativo a la situación tecnológica de la sociedad. Lo que hoy es Big Data puede no serlo dentro de varios años. Esto asegura que Big Data será un tema recurrente de la literatura especializada. La resolución de problemas de Big Data no cierra su desarrollo sino que abre nuevos retos, siempre habrá un conjunto de datos lo suficientemente grande como para poner a prueba los últimos logros del estado del arte.

Los problemas que entran en el ámbito de Big Data surgen de manera natural en el mundo actual. En palabras de Francisco Herrera triguero, investigador de la Universidad de Granada [14]:

"Vivimos en la era de la información. El progreso y la innovación no se ve obstaculizado por la capacidad de recopilar datos sino por la capacidad de gestionar, analizar, sintetizar y descubrir el conocimiento subyacente en dichos datos. Este es el reto de las tecnologías de Big Data."

A veces se utiliza el término Big Data para referirse meramente a conjuntos de datos masivos. Sin embargo, los problemas de Big Data constan de múltiples características que los hacen todavía más complejos. En la literatura especializada estas características se denominan las 3 V's de Big Data [21]:

- **Volumen.** El tamaño de los conjuntos de datos a procesar es cada vez mayor, por ejemplo, facebook procesa cada día 500 TB de información. Este volumen de datos requiere tecnologías específicas para que los servidores de altas prestaciones puedan manejar la información con éxito.
- **Velocidad.** Necesitamos herramientas que permitan procesar y analizar conjuntos de datos masivos en poco tiempo. Además, es habitual que el procesamiento de los datos deba ser incluso en tiempo real, esto es, los datos llegan al sistema de forma continua y este debe agregar la información de los mismos.
- **Variedad.** Los datos a tratar provienen de una gran variedad de fuentes. Por tanto, las herramientas Big Data deben permitir procesar a la vez datos de diferentes características y tamaños. Es más, habitualmente encontramos datos de tres tipos: estructurados, semi estructurados y sin estructurar. Los datos estructurados son sencillos de clasificar. Sin embargo, los datos sin estructurar son aleatorios y difíciles de analizar. Por su parte, los datos semi estructurados requieren técnicas avanzadas para poder clasificarlos correctamente.

Algunos autores han extendido estas características hasta utilizar un total de 9 V's: veracidad, valor, viabilidad y visualización entre otras [34]. De esta forma se destacan diferentes aspectos del tratamiento de conjuntos de datos masivos como mantener y analizar la veracidad de los datos y poner en valor el conocimiento subyacente.

El término Big Data ha tomado peso en el ámbito empresarial. Actualmente forma parte del área de conocimiento que se denomina inteligencia de negocio (business intelligence) [2]. La inteligencia de negocio cubre aquellos problemas relacionados con datos que se resuelven en organizaciones empresariales. Las empresas almacenan información de sus clientes y de toda la actividad realizada.

Esta información contiene conocimiento que es valioso para determinar nuevas estrategias de negocio. Podemos decir que el análisis de datos es crucial para que las decisiones tomadas en la empresa sean efectivas.

Big Data también se considera parte del área denominada ciencia de datos [22]. Esta es una temática emergente que aglutina todas las áreas científicas que se encargan del tratamiento de los datos y de la extracción de conocimiento de los mismos. En este contexto, algunos autores denominan Big Data Analytics a la aplicación de Big Data al análisis de datos [15]. La ciencia de datos y la inteligencia de negocio están muy relacionadas, siendo el último término más popular en el ámbito empresarial. La importancia de estas áreas de conocimiento ha dado lugar a una nueva profesión, el científico de datos, que es considerada una de las profesiones más valoradas del siglo XXI [3].

Cuando uno se enfrenta a un problema de Big Data necesita estudiar y desarrollar tecnologías que permitan el procesamiento de gran cantidades de datos de forma eficiente. Además, debe adaptar los algoritmos que pretenda utilizar a este nuevo ámbito, reduciendo en medida de lo posible complejidad algorítmica de los mismos. La dificultad de este proceso resolutivo es lo que hace que los expertos en Big Data sean muy valorados en el ámbito científico y empresarial. En el resto del documento presentaremos varias tecnologías que se utilizan de forma habitual en Big Data (Hadoop, Spark y Flink) así como un benchmark para Hadoop, denominado TPCx-HS.

2.2. Ingeniería de servidores en Big Data: computación de altas prestaciones

En la Sección 2.1 destacábamos dos problemas del tratamiento de conjuntos de datos masivos: el conjunto de datos puede no caber en memoria principal y se necesita una gran capacidad de cómputo para ejecutar algoritmos sobre este. La solución a este problema suele ser la aplicación de la computación de altas prestaciones [23].

El término computación de altas prestaciones se refiere a la agregación de varios computadores con el fin de obtener una mayor potencia y rendimiento. Cada uno de los computadores utilizados se denomina nodo mientras que al conjunto de todos los nodos se le llama clúster o servidor de altas prestaciones. La agregación de la capacidad de cómputo de los nodos se consigue gracias a la paralelización y programación distribuida. Todos los nodos realizan tareas simultáneamente que ayudan a la resolución de un problema. El conjunto de todas las tareas a realizar conforma el algoritmo que se ejecuta en el servidor. De esta forma el trabajo a realizar se reparte entre los nodos, obteniendo un menor tiempo de ejecución.

Sin embargo, el uso de la computación de altas prestaciones no soluciona de forma directa el problema. Supóngase que una empresa compra un servidor en el cual ejecuta un algoritmo distribuido con eficiencia $\theta(n^2)$ sobre conjuntos de datos masivos. En el momento en el que se compra el hardware consiguen ejecutar el algoritmo en 8 horas, obteniendo resultados en el mismo día. Sin embargo, siguiendo la tendencia mostrada en la Figura 1, es probable que el conjunto de datos a utilizar se duplique en tamaño dentro de un año. En tal caso el servidor puede tener problemas desde el punto de vista de la memoria y el flujo de dato debido a los cuellos de botella que pueda mostrar el hardware. Por otro lado, aunque no hubiese problemas con la memoria, el algoritmo tardará como mínimo 4 veces más en ejecutarse al tener complejidad cuadrática. Por tanto, la ejecución pasará a durar 32 horas, no pudiendo obtener resultados en un mismo día.

El ejemplo anterior muestra un problema de escalabilidad inherente a Big Data. Aunque está claro que el procesamiento distribuido en servidores de altas prestaciones es la solución, el crecimiento de los conjuntos de datos hace que el hardware se quede obsoleto en poco tiempo. Una solución puede ser actualizarlo cada año. Sin embargo, una empresa no puede permitirse renovar el hardware anualmente pues esto supondría un aumento de costes, consumo eléctrico, mantenimiento y, además, requiere una gran cantidad de espacio. Consecuentemente, el buen aprovechamiento de los recursos del servidor es necesario para obtener resultados satisfactorios en Big Data. Necesitamos comprar

desde un inicio aquel servidor que proporcione el mayor rendimiento para el manejo de datos de manera que la inversión sea rentable a largo plazo. La ingeniería de servidores es clave en esta cuestión.

Por último, hay que destacar que los servidores a utilizar contendrán ingentes cantidades de información de vital importancia. Por tanto, se necesita almacenar los datos en un sistema de archivos tolerante a errores. Esto es, si un nodo o disco duro del servidor se estropea, no se pueden perder datos ni provocar una caída del servidor. En esta situación se encontró Google a principios de este siglo cuando desarrolló Google File System, GFS, que fue presentado oficialmente en 2003 [11]. Este es un sistema de archivos escalable y distribuido que utilizan para aplicaciones distribuidas que requieran conjuntos de datos masivos. El sistema proporciona tolerancia ante fallos de forma automática y administra de forma eficiente grandes conjuntos de datos en el servidor, no necesitando que este sea especialmente caro. Además, proporciona escalabilidad en el sentido de que se pueden añadir o eliminar nodos o discos duros sin problemas. De esta forma, se consigue un correcto almacenamiento de los datos en el servidor.

Google File System es un software privado. Basándose en las ideas de este, la fundación Apache ha desarrollado HDFS (Hadoop Distributed File System) [1]. Incidiremos en este en la Sección 3.2, pues una herramienta esencial dentro de Hadoop.

3. Map Reduce. Hadoop

En esta sección introducimos Map Reduce, un nuevo paradigma de programación distribuida que es básico en el ámbito de Big Data. En la Sección 3.1 explicamos el paradigma mientras que en la Sección 3.2 destacamos uno de los softwares libres que lo implementan.

3.1. Map Reduce

En la Sección 2.2 destacábamos el papel central de los servidores de altas prestaciones y la programación distribuida en Big Data. Por tanto, la implementación de algoritmos para Big Data requiere herramientas y librerías que permitan realizar implementaciones distribuidas. Las librerías OPEN MP y MPI son muy conocidas [12, 13]. OPEN MP permite paralelizar cálculos vectoriales. Por su parte, MPI se basa en el paso de mensajes entre hebras como mecanismo para realizar implementaciones distribuidas. Estas librerías operan a bajo nivel y funcionan principalmente con lenguajes de programación como c y c++.

La implementación de un algoritmo distribuido utilizando estas librerías es muy costosa. El programador debe indicar expresamente el tratamiento distribuido de los datos y cómo se paralelizan los cómputos. Además, tiene que crear las hebras asociadas y encargarse de que el sistema funcione correctamente. Nótese que si durante la ejecución del algoritmo un nodo se estropea, entonces la implementación puede dejar de funcionar a no ser que el programador le haya dedicado mucho tiempo para conseguir que sea tolerante ante fallos. Es más, si uno implementa varios algoritmos distribuidos esencialmente se está implementando el mismo tipo de paralelización una y otra vez.

Recordemos en este punto el problema de escalabilidad de Big Data, destacado en la Sección 2.2. Es claro que una solución a este problema consiste en reformular los algoritmos a utilizar, mejorando la eficiencia y escalabilidad. Sin embargo, esto implica un gran trabajo de implementación dada la dificultad de la programación distribuida. Ante esta situación en Google se dieron cuenta de la necesidad de desarrollar un nuevo paradigma de programación distribuida. Este debía proporcionar una interfaz que permitiese al programador implementar algoritmos distribuidos de forma simple y eficiente sin tener que centrarse en cuestiones de bajo nivel. La distribución de los datos en los nodos y la tolerancia ante los fallos serían llevadas a cabo por el propio lenguaje de programación de manera

que el programador solamente tiene que desarrollar el algoritmo en cuestión. Gracias a esta idea surge el paradigma de programación Map Reduce, que Google hizo público en 2004 [5, 6, 4].

Map Reduce se basa en un proceso de abstracción a la hora de implementar algoritmos distribuidos. Los creadores de este paradigma se inspiraron en las funciones map y reduce que suelen estar presentes como primitivas en múltiples lenguajes de programación funcional. En primer lugar, la entrada y la salida de un algoritmo se conciben como dos listas de parejas (clave,valor), que pueden tener tipos distintos. El programador debe proporcionar el código de las funciones map y reduce. Estas funciones se ejecutarán en paralelo en los diferentes nodos del clúster.

- La función **map** toma como entrada una pareja (clave, valor) y produce un nuevo conjunto de parejas (clave, valor), denominado conjunto intermedio. La función map se aplica en paralelo a cada una de las parejas que conforman la entrada del algoritmo. Tras la ejecución de todos los maps, el sistema ordena y filtra los conjuntos intermedios de parejas de manera que a cada posible clave se le asigna una lista con todos los valores que tenía asociados como pareja.
- La función **reduce** toma una clave y una lista de valores asociados a la clave. Se aplica a cada una de las salidas que se han obtenido tras la ordenación y el filtrado de los resultados de los maps. Los valores de cada clave se mezclan mediante el procedimiento indicado por el programador. Este debe devolver al final cero o más parejas formadas por la clave y cada uno de los valores obtenidos tras la mezcla. La lista de valores que se proporciona como argumento a reduce se representa en la implementación mediante un iterador. De esta forma se puede trabajar con listas de valores tan grandes que no quepan en memoria principal.

Para ejemplificar el funcionamiento de Map Reduce mostraremos un algoritmo distribuido que cuenta el número de ocurrencias de cada una de las palabras de un documento. El Algoritmo 1 contiene el pseudo-código de este. La función map toma como parámetros el nombre del documento y una string con parte del texto asociado. Por cada palabra que encuentra emite una pareja con la palabra y un 1. La función reduce suma todas las ocurrencias encontradas de la palabra, devolviendo la pareja (palabra, número total de ocurrencias).

Algoritmo 1 Obtención del número de ocurrencias de cada una de las palabras de un texto mediante Map Reduce.

```
key: Nombre del documento
value: Texto del documento
function MAP(String key, String value)
    for each word w in value do
        EmitIntermediate(w, "1" )
    end for
end function
```

```
key: Palabra
values: Conjunto con las ocurrencias de la palabra
function REDUCE(String key, Iterator values)
    result = 0
    for each value v in values do
        result += Int(v);
    end for
    return key, String(result);
end function
```

La paralelización la lleva cabo el sistema de forma automática. Para ello la entrada debe particionarse en M bloques en tiempo de ejecución. Cada uno de estos bloques contiene una lista de parejas (clave, valor). Map Reduce repartirá los bloques en distintos nodos del servidor de forma que cada uno de estos nodos ejecuta la función map sobre cada una de las parejas de su bloque. Las hebras que realizan esta acción se denomina mappers. El programador debe indicar tanto el número de bloques a utilizar (M) como definir la función utilizada para el particionamiento de la entrada. Habitualmente se recomienda que cada uno de los M bloques contenga entre 16 y 64 MB.

Conforme los maps terminan de ejecutarse se va realizando la fase de ordenación y filtrado de los resultados. En este ejemplo para cada palabra se obtiene una lista de unos con tantos elementos como el número de veces que se haya encontrado la palabra. Una vez han finalizado los mappers, se pueden ejecutar las hebras que aplican la función reduce. Estas se denominan reducers. Para ello el conjunto de claves intermedias se particiona en R bloques. Cada uno de esto R bloques se asigna a un nodo con un reducer, que ejecutará la función reduce sobre las parejas (clave, iterador) que le

correspondan. El programador debe decidir el valor R para obtener el mejor rendimiento posible del sistema. Tras la ejecución de los reduce se obtiene la lista de las palabras encontradas y el número de ocurrencias de las mismas. La Figura 2 muestra de forma gráfica este proceso.

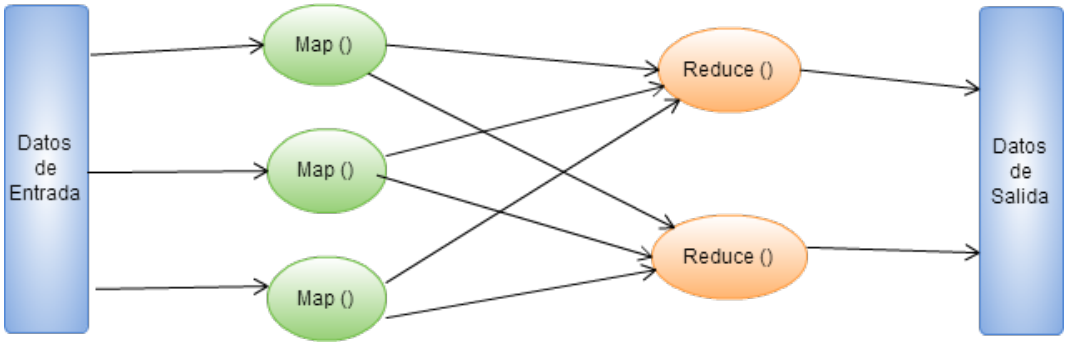

Figura 2: Aplicación de Map Reduce sobre un conjunto de datos.

La implementación de Map Reduce se lleva a cabo siguiendo el paradigma maestro - esclavo. La Figura 4 (véase el anexo) muestra el proceso que se realiza de forma interna en una ejecución de Map Reduce con $M = 3$ y $R = 2$. El primer paso es el particionamiento de la entrada. Tras realizarse este se crea una hebra denominada maestro que se encarga de controlar toda la ejecución. Además, se lanzan tantas hebras como tareas haya que realizar, en total $M + R$. Estas hebras se llaman esclavos y son de dos tipos: mappers y reducers. Los esclavos se almacenan en una cola en la que tienen prioridad los mappers.

Las hebras de la cola se van asignando a los nodos del sistema y los mappers comienzan a ejecutarse. Cada mapper genera parejas intermedias que se almacenan en un buffer de la memoria. Periódicamente, el buffer se vacía, escribiendo las parejas en un disco duro asociado al mapper. En cada disco duro se utilizan R regiones, numeradas desde el 0 al $R - 1$. Las parejas se reparten entre estas regiones utilizando una función hash sobre las claves. De esta manera parejas con la misma clave irán a la misma región disco duro.

Como consecuencia, si los datos generados no están uniformemente distribuidos en todas las claves puede suceder que una región contenga mucho más trabajo que otra. A cambio, se obtiene un proceso eficiente para conseguir que todas las parejas con la misma clave vayan a la misma región.

La localización de estas regiones en los discos es enviada al maestro. Cuando un reducer pasa a ejecutarse, el maestro le notifica de la localización de las regiones de los discos que les corresponden (aquellas que tienen el mismo índice que el reducer). Posteriormente, el esclavo llama a los discos para obtener las regiones correspondientes. Nótese que de esta forma todos las parejas con igual clave van al mismo reducer. Tras la obtención de los datos, el reducer los ordena por clave y agrupa aquellos valores que pertenezcan a la misma clave en conjuntos. Si el tamaño de los datos es demasiado grande, entonces una ordenación externa es utilizada.

Cuando todos los mappers se han ejecutado, los reducers comienzan a aplicar la función reduce a sus correspondientes parejas. Cada hebra produce un archivo de salida con los resultados obtenidos. Tras finalizar la ejecución de estas, el control vuelve al usuario. Normalmente se deberían unir los R archivos generados en uno nuevo, para lo cual se requeriría una nueva llamada a MapReduce u otra aplicación distribuida que permita realizar este proceso.

En este contexto se implementa la tolerancia a errores como sigue. Periódicamente el maestro realiza un ping a cada uno de los esclavos. Si algunos de ellos no le responde, entonces se asume que el nodo ha fallado y se envía su correspondiente trabajo al primer nodo que se encuentre disponible. En caso de que falle un mapper, los reducers son informados de que deben leer los datos de este mapper desde la nueva localización en disco a no ser que ya hayan completado la lectura de los datos de la ejecución errada.

En definitiva, Map Reduce permite a los programadores con poca experiencia en sistemas distribuidos llevar a cabo implementaciones eficientes para grandes cantidades de datos. De esta manera se ahorra trabajo de implementación, pudiendo dedicarse el programador a mejorar la eficiencia de los algoritmos. Es más, una implementación típica de Map Reduce puede procesar varios tera bytes de datos en servidores con miles de nodos.

3.2. Hadoop

La implementación de Google del paradigma de programación Map Reduce es privada. Sin embargo, las publicaciones que hicieron los trabajadores de Google sobre este paradigma han inspirado implementaciones libres de Map Reduce. La más importante sin duda es Hadoop [33].

Hadoop comenzó a desarrollarse poco después de la publicación de GFS y Map Reduce. Inicialmente el proyecto en cuestión era Apache Nutch [16] y tenía como objetivo desarrollar un buscador web libre. El proyecto estaba siendo desarrollado por Doug Cutting entre otros. Sin embargo, el software tenía muchos problemas cuando se ejecutaba sobre un clúster de computadores. Con el fin de solucionar este problema comenzaron a implementar un sistema de archivos distribuidos basado en GFS y una versión de Map Reduce. Este nuevo proyecto pasó a llamarse Hadoop en 2006. Como curiosidad, Hadoop es el nombre del elefante de peluche del hijo de Dough Cutting, que fue director del proyecto.

Hadoop está implementado principalmente en Java, aunque utiliza de forma minoritaria C y utilidades de la línea de comandos. Se encuentra disponible para Windows y los sistemas Unix, aunque solo incluye soporte para Linux. Hadoop incluye 4 módulos básicos [7]:

1. **Hadoop Common.** Librería con las funciones y utilidades comunes al resto de módulos.
2. **Hadoop Distributed File System (HDFS).** Sistema de archivos distribuidos que imita las ideas utilizadas por Google en GFS. Es el núcleo de todas las aplicaciones de Hadoop.
3. **Hadoop YARN.** Framework para la asignación de tareas a los nodos y mantenimiento de los recursos un clúster.
4. **Hadoop MapReduce.** Implementación del paradigma Map Reduce en el ecosistema Hadoop. Utiliza YARN para procesar en paralelo de grandes conjuntos de datos. Funciona de forma similar a la explicación dada en la Sección 3.1. El programador debe proporcionar la implementación de las funciones map y reduce y del módulo de particionamiento de la entrada. Aunque Hadoop está implementado principalmente en Java, permite utilizar Java, Python, Ruby y C++ para programar los algoritmos. Además, Hadoop MapReduce posibilita una alta configuración, como modificar las operaciones de ordenación y filtrado de los datos.

Además de estos módulos básicos, Hadoop incluye otros subproyectos como Hive, que utiliza Map Reduce para realizar búsquedas SQL sobre los datos almacenados con HDFS, o Spark, que estudiaremos en la Sección 4.

4. Spark y Flink

El paradigma de programación distribuida Map Reduce presenta varios problemas. En primer lugar, no todos los algoritmos pueden formularse de forma eficiente utilizando las funciones map y reduce. Por ejemplo, encontramos algoritmos iterativos de machine learning como boosting cuyas iteraciones dependen de los resultados anteriores. En cada iteración habría que ejecutar de nuevo Map Reduce, teniendo que distribuir el conjunto de datos de nuevo en memoria, lo que supone un gran tiempo perdido.

Apache Spark [7] started as a research project at UC Berkeley in the AMPLab, was started with a goal to design a programming model that supports a much wider class of applications than MapReduce, while maintaining its automatic fault tolerance. Spark offers an abstraction called Resilient distributed Datasets (RDDs) [8] to support these applications efficiently. RDDs can be stored in memory between queries without requiring replication. Instead, they rebuild lost data on failure using lineage: each RDD remembers how it was built from other datasets (by transformations like map, join or groupBy) to rebuild itself. RDDs allow Spark to outperform existing models by up to 100x in multi-pass analytics. RDDs can support a wide variety of iterative algorithms, as well as interactive data mining and a highly efficient SQL engine Shark [9].

Spark es un sistema de computación de código abierto enfocado a la rapidez y facilidad de uso. Es ideal si se tiene una gran cantidad de datos para los que se requiere un procesamiento de baja latencia, lo cual un programa de Map Reduce no provee. Por ello, Spark es una alternativa que trabaja cien veces más rápido de lo que lo hace Map Reduce para, por ejemplo, algoritmos iterativos. Spark aporta velocidad en la computación con clusters en memoria(no sé si esta bien traducido esta última frase)[32].

Spark combina SQL, continuos y complejos análisis de la misma aplicación para manejar un amplio rango de los escenarios que surgen al procesar los datos.

Flink es una herramienta de procesamiento para Big Data y es conocida por procesar grades cantidades de datos rápidamente con baja latencia y con una alta tolerancia de fallos en sistemas distribuidos de larga escala. La característica que lo define es la habilidad de procesar flujos de datos en tiempo real. Flink surgió como un proyecto académico de código abierto conocido como Stratosphere [19].

4.1. Comparación de Spark y Flink

En cuanto al procesamiento de datos, Spark realiza un procesamiento por lotes mientras que Flink procesa los datos en tiempo real. Spark procesa bloques de data mientras que Flink puede procesar filas tras filas de datos en tiempo real, por lo que hay un mínimo de latencia con Spark, lo que no ocurre con Flink.

En cuanto a las iteraciones, Spark permite iteraciones de datos en lotes, pero Flink puede iterar sobre los datos usando su arquitectura en tiempo real.

5. Benchmarks: TPCx-HS

La variedad de computadores es bastante heterogénea. Cada uno realiza de forma eficiente un determinado conjunto de operaciones a consta de presentar peores resultados en otros factores. Por tanto, la comparación entre diferentes modelos de computadores es compleja. Consecuentemente, se han creado benchmarks con el objetivo de aportar elementos de juicio con los que se discernir entre el uso de un computador u otro para una determinada aplicación. Habitualmente el benchmarking se define como la obtención de información útil mediante pruebas empíricas que ayude a una organización a mejorar sus procesos [8]. Sin embargo, el benchmarking de computadores es un proceso costoso computacionalmente. Gasta tanto energía como mucho tiempo de cómputo. Por tanto, se ha de reservar para cuestiones sean importantes y no para evaluar tareas simples [20].

Una técnica utilizada para ver qué sistema nos da más prestaciones son los benchmarks, uno de ellos es TPC. TPC es una organización sin ánimo de lucro que estudia el proceso de transacción y desarrolla benchmarks para bases de datos. El término transacción se suele atribuir a aspectos bancarios. Sin embargo, en este contexto una transacción es un conjunto de operaciones de un computador que incluyen lectura y escritura en disco, llamadas a funciones del sistema operativo, o

cualquier forma de transferencia de datos de un sistema a otro. Los benchmarks que produce TPC miden el proceso de transacción y el rendimiento de las bases de datos en términos del número transacciones que se pueden hacer por unidad de tiempo [28].

TPCx-HS (Transaction Processing Performance Council Express Hadoop System) se desarrolló para proveer de una medida del rendimiento fiable, rendimiento relación precio, disponibilidad y, opcionalmente, datos de consumo de energía de los sistemas de Big Data que usen Hadoop. TPCx-HS fue el primer benchmark objetivo que permitía medir tanto hardware como software, como el Hadoop Runtime. [31]

5.1. Carga de trabajo de TPCx-HS

La carga de trabajo de TPCx-HS consiste en los siguientes módulos [29]:

- HSGen: es un programa que genera los datos según factor de escala, que suele estar entre un 1TB y 10000TB (se denota TB como terabytes).
- HSDataCheck: es un programa que comprueba el cumplimiento del conjunto de datos.
- HSSort: es un programa que ordena los datos según un orden total.
- HSValidate: es un programa que valida la salida, es decir, los resultados obtenidos.

5.2. Fases de ejecución del benchmark

Una ejecución válida consiste en cinco fases separadas que corren secuencialmente. Estas fases no se solapan en su ejecución, es decir, el comienzo de la fase 2 no puede darse hasta que la fase 1 esté completa. Para que comience cada fase se necesita de un script llamado $< TPCx - HS - master >$ que es el que inicia cada fase y que puede ser ejecutado desde cualquier nodo del sistema que se está bajo test [29].

Una ejecución válida consiste en cinco fases separadas que corren secuencialmente. Estas fases no se solapan en su ejecución, es decir, el comienzo de la fase 2 no puede darse hasta que la fase 1 esté completa. Para que comience cada fase se necesita de un script llamado ¡TPCx-HS-master¿ que es el que inicia cada fase y que puede ser ejecutado desde cualquier nodo del sistema que esté bajo el test [29].

Las fases de ejecución son las siguientes:

- Fase 1: se generan los datos de entrada usando HSGen. Se han de copiar en un soporte duradero y hacer la copia replicada en tres discos (llamado *"3-ways replication"*), que suelen ser el principal, el secundario y uno de respaldo [17].
- Fase 2: se verifica la validez del conjunto de datos usando HSDataCheck. El programa sirve para verificar la cardinalidad, tamaño y el factor de réplica de los datos generados. Si el programa reporta un fallo, entonces la ejecución no es válida y se deberá volver a empezar.
- Fase 3: se lanza el programa HSSort con el que se ordena los datos de entrada. Esta fase muestra los datos de entrada y los datos de salida (los datos ya ordenados). Al igual que en la fase 1, se han de copiar los datos en un soportde duradero y hacer el *"3-ways replication"*.
- Fase 4: se comprueba la viabilidad del conjunto de datos usando HSDataCheck. El programa comprueba la cardinalidad de los datos, tamaño y el factor de replicación de los datos ordenados. Si el programa reporta un fallo, entonces la ejecución no es válida y se deberá volver a empezar.
- Fase 5: una vez que en la fase 4 se ha comprobado la cardinalidad de los datos antes y después de ordenarlos, se procede a la validación de los datos con HSValidate. Como su nombre indica, HSValidate comprueba que los datos de salida sean correctos, es decir, se realiza la verificación

de la replicación de los datos. Si reporta el fallo consisten en que el HSSort no generó el correcto orden de salida, la ejecución se considerará inválida.

El benchmark consiste en dos ejecuciones (ver la Figura 3) y cada vez que se ejecuta una fase se ha de especificar el tiempo que ha llevado la ejecución. Entre la primera ejecución de las cinco fases y la segunda, hay una fase intermedia que sirve para limpiar el sistema, conocida como *"file system cleanup"* y, lógicamente, no se permite ninguna actividad durante dicha fase. Una vez realizadas las dos ejecuciones se obtiene el tiempo total de ejecución que servirá para calcular datos en la fase de medición.

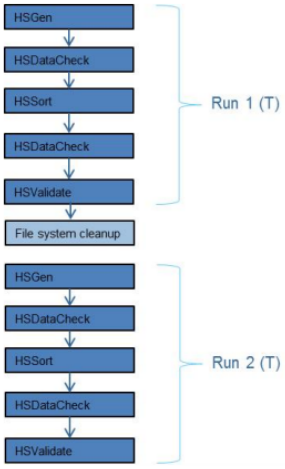

Figura 3: Gráfico de las ejecuciones de TPCx-HS

Como consideraciones finales en la ejecución, el sistema bajo test no puede ser reconfigurado o cambiado durante o entre cualquiera de las fases de la ejecución ni tampoco entre la primera y la segunda ejecución. Cualquier cambio que se haga en el sistema se deberá realizar antes del comienzo de la fase 1 de la primera ejecución. El factor de escala usado en el conjunto de datos del test debe ser escogido del conjunto de factores de escala definido como sigue: 1TB, 3TB, 10TB, 30TB, 100TB, 300TB, 1000TB, 3000TB, 10000TB.

5.3. Medida del rendimiento

Para hacer el análisis de la medida del rendimiento, TPCx-HS define las siguientes medidas:

- HSph@SF: refleja la medida del rendimiento de TPCx-HS.
- $/HSph@SF: es la medida del rendimiento-precio.
- Si se escoge la opción de *"TPC-energy"*, la medida de la energía que hace TPCx-HS informa de la potencia por rendimiento.

La medida del rendimiento se representa con HSph@SF, que se mide con la siguiente expresión:

$$HSph@SF = \frac{SF}{T/3600}$$

donde:

- SF es el factor de escala escogido.

- T es el tiempo total que se obtuvo al sumar el tiempo de las dos ejecuciones.

La medida del rendimiento-precio se puede calcular con la siguiente fórmula. Dicha medida usa una lógica negativa ya que a menor valor se tiene un mejor rendimiento-precio.

$$\$/HSph@SF = \frac{P}{HSph@SF}$$

donde P es el costo del sistema en el que se han hecho las ejecuciones.

5.4. Comparación de la medida

Un resultado dado por la ejecución de TPCx-HS sólo puede ser comparado con otro resultado que provenga de la ejecución de este benchmark y que tenga el mismo factor de escala. Otras consideraciones a tener en cuenta son:

- Los resultados producidos por test que tienen diferente factor de escala no son comparables, debido a los *retos* computacionales encontrados en volúmenes de datos de distinto tamaño.
- Si los resultados medidos con diferentes factores de escalas aparecen impresos o en algún documento electrónico, entonces cada referencia que se haga a uno de estos resultados se ha de especificar claramente el factor de escala que se usó para obtener esos resultados. Si los resultados aparecen de forma gráfica, el factor de escala en el que se basó dicha medición se deberá poder discernir, por ejemplo usando una etiqueta en uno de los ejes.

La figura 5 (véase el anexo) presenta una tabla con los resultados para algunas compañías donde se puede ver el factor de escala usado, el sistema y los resultados obtenidos tras la ejecución del benchmark.

6. Conclusión

Podemos decir que Big Data es un área que aúna tanto la ingeniería del software como la ingeniería de servidores y la ciencia de datos.

Referencias

[1] D. Borthakur. *HDFS architecture guide*. 2008. URL: archive.cloudera.com/cdh/3/hadoop-0.20.2-cdh3u6/hdfs_design.pdf.

[2] H. Chen, R. H. Chiang y V. C. Storey. «Business Intelligence and Analytics: From Big Data to Big Impact». En: *MIS Quarterly* (2012), págs. 1165-1188.

[3] Thomas H. Davenport y D. J. Patil. «Data Scientist: The Sexiest Job of the 21st Century». En: *Harvard Business Review* (2012). URL: https://hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century/.

[4] J. Dean y S. Ghemawat. «MapReduce: a flexible data processing tool». En: *Communications of the ACM* (2010), págs. 72-77.

[5] J. Dean y S. Ghemawat. «MapReduce: Simplified data processing on large clusters». En: *Proceedings of Operating Systems Design and Implementation (OSDI)* (2004), págs. 137-150.

[6] J. Dean y S. Ghemawat. «MapReduce: simplified data processing on large clusters». En: *Communications of the ACM* (2008), págs. 107-113.

[7] *Documentación oficial de Hadoop*. URL: http://hadoop.apache.org/.

[8] Confederación Granadina de Empresarios. *¿Qué es el Benchmarking*. URL: http://www.cge.es/portalcge/tecnologia/innovacion/4111benchmarking.aspx.
[9] Hugo Evans y et. al. *Big Data and the Creative Destruction of Today's Business Models*. URL: http://www.atkearney.es/paper/-/asset_publisher/dVxv4Hz2h8bS/content/big-data-and-the-creative-destruction-of-today-s-business-models/10192.
[10] Rex Farrance. «Timeline: 50 Years of Hard Drives». En: *PCWorld* (2016). URL: http://www.pcworld.com/article/127105/article.html.
[11] S. Ghemawat, H. Gobioff y S. T. Leung. «The Google file system». En: *ACM SIGOPS operating systems review* (2013), págs. 29-43.
[12] Pawan Ghildiyal. *MPI vs OpenMP: A Short Introduction Plus Comparison (Parallel Computing)*. URL: http://pawangh.blogspot.com.es/2014/05/mpi-vs-openmp.html.
[13] William Gropp, Ewing Lusk y Anthony Skjellum. *Using MPI: portable parallel programming with the message-passing interface*. MIT Press, 1999.
[14] F. Herrera. *Inteligencia artificial, inteligencia computacional y Big Data*. Servicio de Publicaciones, Universidad de Jaén, 2014.
[15] Karthik Kambatla y col. «Trends in big data analytics». En: *Journal of Parallel and Distributed Computing* (2014), págs. 2561-2573.
[16] R. Khare y col. «Nutch: A flexible and scalable open-source web search engine». En: *Oregon State University* (2004), págs. 32-42.
[17] Linbit. *Three-way replication*. URL: https://www.drbd.org/en/doc/users-guide-83/s-three-way-repl.
[18] Computer History Museum. *Timeline of Computer History*. URL: http://www.computerhistory.org/timeline/1951/.
[19] KDNuggets news. *Fast Big Data: Apache Flink vs Apache Spark for Streaming Data*. URL: http://www.kdnuggets.com/2015/11/fast-big-data-apache-flink-spark-streaming.html.
[20] Universidad Técnica José Peralta. *Ventajas, desventajas y causas posibles de fracasos del benchmarking*. URL: http://www.buenastareas.com/ensayos/Ventajas-Desventajas-y-Causas-Posibles-De/3531988.html.
[21] Seref Sagiroglu y Duygu Sinanc. «Big data: A review». En: *International Conference on. IEEE* (2013), págs. 42-47.
[22] R. Schutt y C. O'Neil. *Doing data science: Straight talk from the frontline*. O'Reilly Media, Inc., 2013.
[23] Charles Severance y Kevin Dowd. *High Performance Computing*. O'Reilly Media, 1998.
[24] Internet live stats. *Total number of Websites*. URL: http://www.internetlivestats.com/total-number-of-websites/.
[25] N.B Stern. *From Eniac to UNIVAC: An Appraisal of the Eckert-Mauchy Computers. Butterworth-Heinemann.* Butterworth-Heinemann, 1981.
[26] Mario Tascón y Arantza Coullaut. *Big data y el Internet de las cosas*. Catarata, 2016.
[27] Steve Todd. *AMPed At UC Berkeley*. URL: http://stevetodd.typepad.com/my_weblog/2011/08/amped-at-uc-berkeley.html.
[28] TPC. *About TPC*. URL: http://www.tpc.org/information/about/abouttpc.asp.
[29] TPC. *TPC EXPRESS BENCHMARK ™ HS, Standard Specification Version 1.3.0*.
[30] TPC. *TPCx-HS - Ten Most Recently Published Results*. URL: http://www.tpc.org/tpcx-hs/results/tpcxhs_last_ten_results.asp.
[31] TPC. *Transaction Processing Performance Council (TPC) Launches TPCx-HS, the First Vendor-Neutral, Industry Standard Big Data Benchmark*. URL: http://www.tpc.org/information/other/tpcx-hs_press_release_final.pdf.
[32] Big Data University. *Spark Fundamentals I*. URL: http://bigdatauniversity.com/courses/spark-fundamentals/.
[33] T. White. *Hadoop: The definitive guide.* O'Reilly Media, Inc, 2012.
[34] P. Zikopoulos y C. Eaton. *Understanding big data: Analytics for enterprise class hadoop and streaming data.* McGraw-Hill Osborne Media, 2011.

Anexo

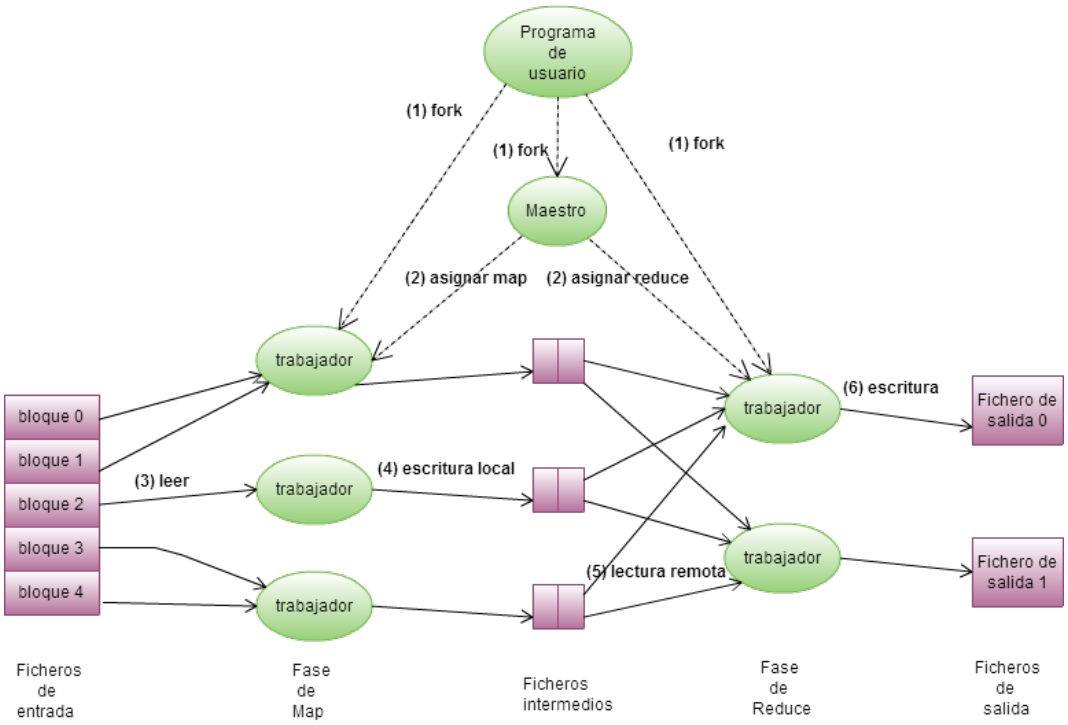

Figura 4: Funcionamiento interno de Map Reduce.

Date Submitted	Scale Factor	Company	System	HSph	Price/HSph	Watts/KHSph	System Availability	Apache Hadoop Compatible Software	Operating System	Cluster
03/30/16	1 TB	CISCO	Cisco UCS Integrated Infrastructure for Big Data	10.12	38,168.98 USD	NR	03/31/16	MapR Converged Community Edition Version 5.0	Red Hat Enterprise Linux Server 6.7	Y
03/30/16	10 TB	CISCO	Cisco UCS Integrated Infrastructure for Big Data	12.02	32,135.61 USD	NR	03/31/16	MapR Converged Community Edition Version 5.0	Red Hat Enterprise Linux Server 6.7	Y
03/22/16	10 TB	CISCO	Cisco UCS Integrated Infrastructure for Big Data	11.56	37,066.53 USD	NR	03/23/16	MapR Converged Community Edition Version 5.0	Red Hat Enterprise Linux Server 6.5	Y
10/23/15	30 TB	CISCO	Cisco UCS Integrated Infrastructure for Big Data	23.42	36,800.52 USD	NR	10/26/15	Cloudera Distribution for Apache Hadoop (CDH) 5.3.2	Red Hat Enterprise Linux Server 6.5	Y
10/23/15	100 TB	CISCO	Cisco UCS Integrated Infrastructure for Big Data	21.99	39,193.64 USD	NR	10/26/15	Cloudera Distribution for Apache Hadoop (CDH) 5.3.2	Red Hat Enterprise Linux Server 6.5	Y
10/16/15	1 TB	DELL	Dell PowerEdge 730/730xd	7.39	46,762.93 USD	NR	10/19/15	Cloudera Distribution for Apache Hadoop (CDH) 5.4.2	Red Hat Enterprise Linux Server 6.5	Y
10/16/15	3 TB	DELL	Dell PowerEdge 730/730xd	8.25	41,888.25 USD	NR	10/19/15	Cloudera Distribution for Apache Hadoop (CDH) 5.4.2	Red Hat Enterprise Linux Server 6.5	Y
10/16/15	10 TB	DELL	Dell PowerEdge 730/730xd	9.07	38,101.22 USD	NR	10/19/15	Cloudera Distribution for Apache Hadoop (CDH) 5.4.2	Red Hat Enterprise Linux Server 6.5	Y
10/16/15	30 TB	DELL	Dell PowerEdge 730/730xd	8.38	41,238.43 USD	NR	10/19/15	Cloudera Distribution for Apache Hadoop (CDH) 5.4.2	Red Hat Enterprise Linux Server 6.5	Y
09/24/15	3 TB	CISCO	Cisco UCS Integrated Infrastructure for Big Data	11.76	44,052.98 USD	NR	09/25/15	Cloudera CDH 5.3.0, HDFS API ver 2, Map Reduce API ver 1	Red Hat Enterprise Linux Server 6.5	Y

Figura 5: Tabla de resultados obtenidos con TPCx-HS [30].