



Ingeniería de Servidores

Big Data y TPCx-HS

Resumen

En este texto puedes incluir un resumen del documento. Este informa al lector sobre el contenido del texto, indicando el objetivo del mismo y qué se puede aprender de él.

Índice

1. Introducción	2
2. Big Data	3
3. Map Reduce. Hadoop	4
3.1. Servidores de altas prestaciones. Sistemas de archivos distribuidos	4
3.2. Map Reduce	5
3.3. Hadoop	8
4. Spark y Flink	8
5. Benchmarks: TPCx-HS	8
5.1. Carga de trabajo de TPCx-HS	8
5.2. Fases de ejecución del benchmark	8
5.3. Medida del rendimiento	9
5.4. Comparación de la medida	10
6. Conclusión	11

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 últimas 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 [9] 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 [14]. Este computador tenía una frecuencia de 2,25 mega hercios y 1000 palabras de memoria principal o RAM [19]. 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 [18]. Además, multitud de dispositivos electrónicos se conectan e interaccionan con Internet (lo que se denomina Internet de las cosas [20]). 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 ($1 \text{ ZB} = 10^{12} \text{ GB}$). 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 ($1 \text{ EB} = 10^9 \text{ GB}$). Actualmente, generamos esta cantidad de datos en dos días [16]. Podemos decir que vivimos en una sociedad digital o sociedad de la información.

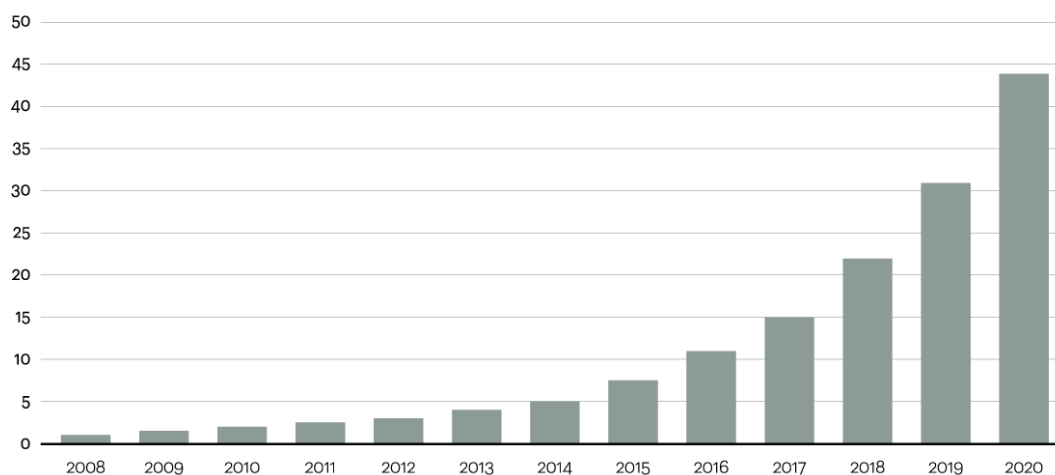


Figura 1: Evolución histórica del número de Zeta Bytes de información almacenados y predicción para los próximos años [8].

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 [11].

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 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 potentes. 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 [21]:

“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 [11]:

“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 [16]:

- **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 la definición hasta utilizar un total de 9 V's: veracidad, valor, viabilidad y visualización entre otras [26]. 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 [17]. 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 [12]. 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].

3. Map Reduce. Hadoop

3.1. Servidores de altas prestaciones. Sistemas de archivos distribuidos

En la Sección 2 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 utilizar servidores de altas prestaciones con procesamiento en paralelo y distribuido.

Los servidores de altas prestaciones (alguna descripción).

Los servidores a utilizar contendrán ingentes cantidades de información de vital importancia. Por tanto, se necesita almacenar la información 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 [10]. 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 3.3, pues una herramienta esencial dentro de Hadoop.

3.2. Map Reduce

Además de un sistema de archivos distribuido que asegure un correcto almacenamiento de la información en el servidor, se necesitan herramientas y librerías que permitan programar algoritmos paralelos y distribuidos. Las librerías OPEN MP y MPI (referencias) son muy conocidas. 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 el 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.

Ante esta situación en Google se dieron cuenta de la necesidad de desarrollar un nuevo paradigma de programación distribuida. Este debía proporcionar una interfaz para 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 hace 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 conjunto intermedio de parejas. 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 obtenidos 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 map. 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 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 para contar el número de ocurrencias de cada una de las palabras de un texto. El Algoritmo 1 muestra

la implementación de este. La función map toma como parámetros el nombre del documento y parte del texto asociado. Por cada palabra que encuentra emite un 1. La función reduce suma todas las ocurrencias encontradas de la palabra, devolviendo la pareja palabra / número total de ocurrencias.

La paralelización se lleva a cabo de forma automática por el sistema. La entrada se particiona en tiempo de ejecución, dando lugar a distintos bloques de datos sobre los que se ejecuta en paralelo la función map. Conforme los maps terminan de ejecutarse se va realizando la fase de filtrado u ordenación de los resultados, obteniendo para cada palabra el conjunto con todos los valores que se han emitido junto a esta. Una vez han finalizado los maps, se pueden ejecutar los reduce también en paralelo, obteniendo como resultado la lista de palabras y el número de ocurrencias de las mismas.

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

<pre> key: Nombre del documento value: Texto del documento function MAP(String key, String value) for each word <i>w</i> in <i>value</i> do EmitIntermediate(<i>w</i>, "1") end for end function </pre>	<pre> key: Palabra values: Conjunto con las ocurrencias de cada palabra function REDUCE(String key, Iterator values) result = 0 for each value <i>v</i> in <i>values</i> do result += Int(<i>v</i>); end for return key, String(result); end function </pre>
--	--

Las llamadas a la función map se distribuyen entre los diferentes nodos del sistema. Para ello la entrada debe particionarse en M bloques de manera que estos bloques se procesen en paralelo en diferentes máquinas. La función map es aplicada de forma secuencial a cada una de las parejas que conforman el bloque. El programador debe indicar tanto el número de bloques como definir la función utilizada en el particionamiento. Habitualmente se recomienda que cada uno de los M bloques contenga entre 16 y 64 MB.

Por su parte, las llamadas a reduce también se distribuyen en el sistema. Para ello, el conjunto de claves intermedias sobre las que se aplican se particiona en R bloques. Cada uno de estos bloques se procesa en un nodo de forma secuencial. El programador debe decidir el valor R para obtener el mejor rendimiento posible del sistema.

1. En primer lugar los archivos de entrada se particionan en los M bloques. Por cada bloque se lanza una copia del programa en el servidor. Una de estas copias se denomina maestro mientras que el resto son esclavos a los que el maestro asigna trabajo. Hay en total M bloques asociados a map y R bloques asociados a reduce que se tienen que asignar.
2. Un trabajador al que se le asigna un bloque de map lee cada una de las parejas clave / valor que lo componen. A cada una de ellas le aplica la función map. Las parejas intermedias que se producen son almacenadas en buffers de la memoria.
3. Periódicamente, las parejas intermedias almacenadas en buffers se escriben en los discos duros. En concreto, cada pareja se asigna a uno de los R bloques según corresponda. La localización de estos bloques en el disco es pasada al maestro, quien se encarga de informar a los esclavos que realizan los reduce de dónde se encuentran almacenados los bloques.
4. Cuando un esclavo es notificado por el maestro de la localización de los bloques del reduce, llama al correspondiente disco local para obtener los datos. Tras esto, ordena los datos 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.
5. El esclavo reduce itera sobre los datos ordenados y por cada clave encontrada aplica el reduce a su correspondiente conjunto de valores intermedios. La salida de cada reduce se añade a un

archivo asociado a este reduce.

6. Cuando todos las las tareas maps y reduce se han completado, el maestro finaliza la ejecución y el control vuelve al usuario.

La salida del algoritmo se encuentra disponible en los R archivos creados por los esclavos que hicieron los reduce. Normalmente se deben unir estos archivos en un nuevo archivo, para lo cual se requeriría una nueva llamada a MapReduce u otra aplicación distribuida que permita realizar este proceso.

La tolerancia a errores se implementa 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 la hebra ha fallado y se envía su correspondiente trabajo al primer nodo que se encuentre disponible. En caso de que falle un esclavo que tenía que ejecutar un map, los nodos que realicen el reduce son informados de que deben leer los datos de la nueva ejecución a no ser que ya hayan obtenido completamente los datos de la ejecución errada.

Por tanto, se requieren tecnologías que manejen grandes conjuntos de datos de forma eficiente y a alto nivel, permitiendo que la implementación sea independiente del hardware a utilizar.

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 al año 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 datos. Es habitual que el hardware tenga cuellos de botella que provoquen que determinadas operaciones relacionadas con la memoria lleven mucho tiempo. Por otro lado, el algoritmo tardará de todas formas 4 veces más en ejecutarse al ser cuadrático. 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 esta claro que el procesamiento distribuido 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. Otra solución consiste en reformular el algoritmo en cuestión, mejorando su eficiencia y escalabilidad.

En cualquier caso, la solución que se tome implicará un gran trabajo para adaptar el software existente al nuevo hardware o para implementar nuevos algoritmos distribuidos. Cada nueva implementación conlleva multitud de líneas de código para manejar el conjunto de datos y realizar los cálculos distribuidos. Además, la implementación debe conseguir que el sistema sea tolerante ante fallos, esto es, si un nodo se estropea, entonces el sistema debe no perder la información y reasignar el trabajo del nodo a otro.

Map Reduce permite a los programadores con poca experiencia en sistemas distribuidos llevar a cabo implementaciones eficientes para grandes cantidades de datos. Una implementación típica de MapReduce puede procesar varios tera bytes de datos en servidores con miles de nodos.

3.3. Hadoop

4. Spark y Flink

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 [7]. 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 [15].

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 los benchmarks para las bases de datos. El término transacción se suele atribuir a aspectos bancarios, pero si pensamos en este concepto como una función que tienen los ordenadores, una transacción puede ser un conjunto de operaciones 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. Este es el ámbito en el que se mueve TPC, que produce benchmarks que miden el proceso de transacción y el rendimiento de las bases de datos en términos de: dados un sistema y una base de datos, ¿cuántas transacciones pueden hacer por unidad de tiempo?[22]

TPCx-HS se desarrolló para proveer de un rendimiento fiable, rendimiento relación precio, disponibilidad y, opcionalmente, datos de consumo de energía de los sistemas de Big Data. TPCx-HS fue el primer benchmark objetivo que permitía medir tanto hardware como software, como el Hadoop Runtime. [25]

5.1. Carga de trabajo de TPCx-HS

La carga de trabajo de TPCx-HS consiste en los siguientes módulos [23]:

- 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 [23].

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[13].
- 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 soporte 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: validación de los datos con HSValidate. Como su nombre indica, HSValidate comprueba que los datos de salida sean correctos y 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 2) 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.

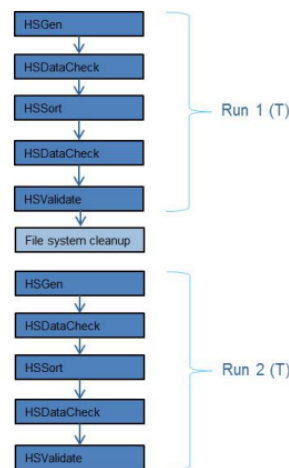


Figura 2: Gráfico de las ejecuciones de TPCx-HS

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:

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

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.

Se presenta a continuación 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.











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	38,168.96 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 3: Tabla de resultados obtenidos con TPCx-HS [24].

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] Confederación Granadina de Empresarios. *¿Qué es el Benchmarking*. URL: <http://www.cge.es/portalcge/tecnologia/innovacion/4111benchmarking.aspx>.
- [8] 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.

- [9] Rex Farrance. «Timeline: 50 Years of Hard Drives». En: *PCWorld* (2016). URL: <http://www.pcworld.com/article/127105/article.html>.
- [10] S. Ghemawat, H. Gobioff y S. T. Leung. «The Google file system». En: *ACM SIGOPS operating systems review* (2013), págs. 29-43.
- [11] F. Herrera. *Inteligencia artificial, inteligencia computacional y Big Data*. Servicio de Publicaciones, Universidad de Jaén, 2014.
- [12] Karthik Kambhata y col. «Trends in big data analytics». En: *Journal of Parallel and Distributed Computing* (2014), págs. 2561-2573.
- [13] Linbit. *Three-way replication*. URL: <https://www.drbd.org/en/doc/users-guide-83/s-three-way-repl>.
- [14] Computer History Museum. *Timeline of Computer History*. URL: <http://www.computerhistory.org/timeline/1951/>.
- [15] 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>.
- [16] Seref Sagiroglu y Duygu Sinanc. «Big data: A review». En: *International Conference on. IEEE* (2013), págs. 42-47.
- [17] R. Schutt y C. O'Neil. *Doing data science: Straight talk from the frontline*. O'Reilly Media, Inc., 2013.
- [18] Internet live stats. *Total number of Websites*. URL: <http://www.internetlivestats.com/total-number-of-websites/>.
- [19] N.B Stern. *Stern, N. B. (1981). From Eniac to UNIVAC: An Appraisal of the Eckert-Mauchy Computers*. Butterworth-Heinemann. Butterworth-Heinemann, 1981.
- [20] Mario Tascón y Arantza Coullaut. *Big data y el Internet de las cosas*. Catarata, 2016.
- [21] Steve Todd. *AMPed At UC Berkeley*. URL: http://stevetodd.typepad.com/my_weblog/2011/08/amped-at-uc-berkeley.html.
- [22] TPC. *About TPC*. URL: <http://www.tpc.org/information/about/abouttpc.asp>.
- [23] TPC. *TPC EXPRESS BENCHMARK™ HS, Standard Specification Version 1.3.0*.
- [24] TPC. *TPCx-HS - Ten Most Recently Published Results*. URL: http://www.tpc.org/tpcx-hs/results/tpcxhs_last_ten_results.asp.
- [25] 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.
- [26] P. Zikopoulos y C. Eaton. *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media, 2011.