BIG DATA: UNA PEQUEÑA INTRODUCCIÓN

AIDAN HOGAN

Profesor Asistente Departamento de Ciencias de la Computación, Universidad de Chile. Ph.D. Computer Science, National University of Ireland, Galway; Licenciado en Ingeniería Electrónica, National University of Ireland, Galway. Líneas de Investigación: Web Semántica, Procesamiento de Datos a Gran Escala, Integración de Datos, Sistemas Distribuidos, Minería de Datos.

ahogan@dcc.uchile.cl

EL VALOR DE LOS DATOS

Soho, Londres, agosto 1854: siete años antes del descubrimiento de los gérmenes por Louis Pasteur, la gente moría por centenares de una enfermedad misteriosa llamada cólera. La sabiduría de la época decía que el cólera era causado por el miasma: algo malo en el aire, una niebla espesa que hacía que la enfermedad naturalmente se acumulara en zonas densamente pobladas. Pero a John Snow, un médico que trabajaba en Londres, no le parecía que esta teoría fuera creíble, por lo que se dispuso a encontrar una mejor.

Snow comenzó encuestando a los aquejados por el cólera en el área, marcando en un diario sus nombres, género, edad, ubicación, fecha de inicio de la enfermedad, fecha de muerte, los hábitos cotidianos, y así sucesivamente, aplicando diversas técnicas estadísticas y analíticas sobre los datos. A partir de su estudio, trazó todos los casos de cólera del Soho, como se muestra en la **Figura 1**. Cada rectángulo oscuro indica un caso de cólera en ese lugar y cada pila representa un hogar o un lugar de trabajo. Usando un diagrama de Voronoi, se hizo evidente que los casos de cólera se agrupaban alrededor de la bomba de agua en la intersección de Broad Street y Cambridge Street; las personas que vivían cerca de otra bomba no se hallaban afectadas. Snow convenció a las autoridades de quitar el mango de la bomba. Los nuevos casos de cólera cesaron.

A pesar de que los microscopios de la época no podían ver la causa física de la enfermedad nadando en el agua, 616 muertes y 8 días más tarde, los datos de Snow habían encontrado la causa: un pozo de agua abierto cavado cerca de un pozo ciego de aguas residuales. Éste fue un hallazgo revolucionario: el cólera no era algo en el aire, sino más bien algo en el agua.

Como científicos de la Computación, a veces olvidamos el valor de los datos. Al igual que un cerrajero podría considerar que las llaves son pedazos de metal que tienen que ser cortados y vendidos, tendemos a considerar los datos como algo que tiene que ser almacenado o analizado, algo que se mueve desde la entrada hasta la salida: compuesto de bytes en una máquina, o terminales en una gramática. Es sólo en el contexto de la importancia de los datos para otros campos –y la capacidad para producir conocimiento desde ellos– que el reciente e inusual ruido alrededor del término Big Data se puede entender.

John Snow comprendió el valor de los datos. Su trabajo en Soho, 1854, estableció el precedente para el campo de la epidemiología: el análisis de los datos para extraer patrones de causalidad sobre enfermedades en zonas pobladas, que abarca los controles de salud pública, los ensayos clínicos, la causa y la propagación de enfermedades infecciosas, la investigación y simulación de brotes, y así sucesivamente. Los casos de éxito de la epidemiología incluyen, entre otros, la erradicación de la viruela, el aislamiento de la poliomielitis a zonas localizadas, y una marcada reducción de los casos de malaria y cólera.

→

Por supuesto, el valor de los datos no se aprecia sólo en el campo de la epidemiología o la genética, o la medicina, o la astronomía observacional, o la física experimental, o la climatología, o la oceanografía, o la geología, o la ecología, o la sociología, o incluso la ciencia o la empresa. Los datos son los protagonistas en muchos de los aspectos científicos, comerciales y sociales de la vida. Al igual que en el trabajo de Snow, existen metodologías comunes a todos los campos que laboran con datos: la recolección de ellos, su curación, la generación de hipótesis, las pruebas estadísticas, la visualización, etc. A diferencia de los tiempos en que Snow trabajaba, los computadores permiten hoy en día recoger, gestionar y procesar los datos con niveles de escalabilidad y eficiencia que Snow no podría haber imaginado.

Pero aún así, parece que a medida que la capacidad de la sociedad para capturar más y más datos sobre el mundo que nos rodea continúa creciendo, las técnicas computacionales convencionales no son suficientes para darle sentido al resultado.

BIG DATA

Big Data, en su esencia, es una idea interdisciplinaria: tener más datos de lo que es posible para dar sentido. Big Data es un llamado a nosotros, los científicos de la Computación, para ofrecer una vez más aún mejores métodos para digerir datos aún más diversos, más complejos, más dinámicos, más granulares y más grandes.

No es difícil entender por qué tantos científicos de la Computación se sienten (en voz baja) desdeñosos del zumbido del término "Big Data". En nuestro campo, los temas fundamentales – como Bases de Datos, Lógica, la Web, Ingeniería de Software, *Machine Learning*, etc. – se fundan en tecnologías con un rico *pedigree*. Por el contrario, "Big Data" es una idea difícil de definir. Sin embargo, no es fácil permanecer totalmente indiferente una vez que uno ve titulares como los U$200 millones de inversión por parte del Gobierno de Estados Unidos en una serie de proyectos nacionales llamados "Big Data Initiative", o la inversión de 30 millones de libras –por parte del gobierno del Reino Unido y un filántropo privado– para crear in "Big Data Institute" en Oxford en temas de epidemiología y descubrimiento de drogas, u otras historias similares en las noticias.

Entonces, ¿qué es Big Data? La definición más canónica (pero aún bastante inescrutable) de Big Data hace referencia a cualquier escenario de uso intensivo de datos, donde el volumen, la velocidad y/o la variedad de los datos dificulta la utilización de "técnicas tradicionales de gestión". Este desafío es conocido como el de "las tres V's", y la mayor parte del énfasis hasta ahora se ha concentrado en el tema del volumen de los datos y (en menor medida) en su velocidad.

FIGURA 1.
PARTE DEL MAPA DE SNOW DE CASOS DE CÓLERA EN SOHO, 1854.

NUEVAS RAZAS DE BASES DE DATOS

La respuesta tradicional a trabajar con grandes cantidades de datos estructurados siempre ha sido simple: el uso de una base de datos relacional. Si el volumen o la velocidad de los datos impedían el uso de una base de datos relacional, entonces usted era (i) una empresa como Facebook, donde su equipo de altamente remunerados ingenieros le permitiría encontrar una solución personalizada, o (ii) un tipo con mala suerte. Si usted se enfrenta a un problema similar estos días, entonces tiene lo que se llama un problema de "Big Data".

La comprensión de que el sistema de bases de datos relacionales (RDBMS) no es –en palabras de Stonebraker [10]— "una solución de talla única", tomó muchos años, pero el espacio de las bases de datos ha sido ahora desmonopolizado bajo la bandera de "Big Data". La **Figura 2** proporciona una amplia perspectiva de este espacio, donde se ha hecho una selección de

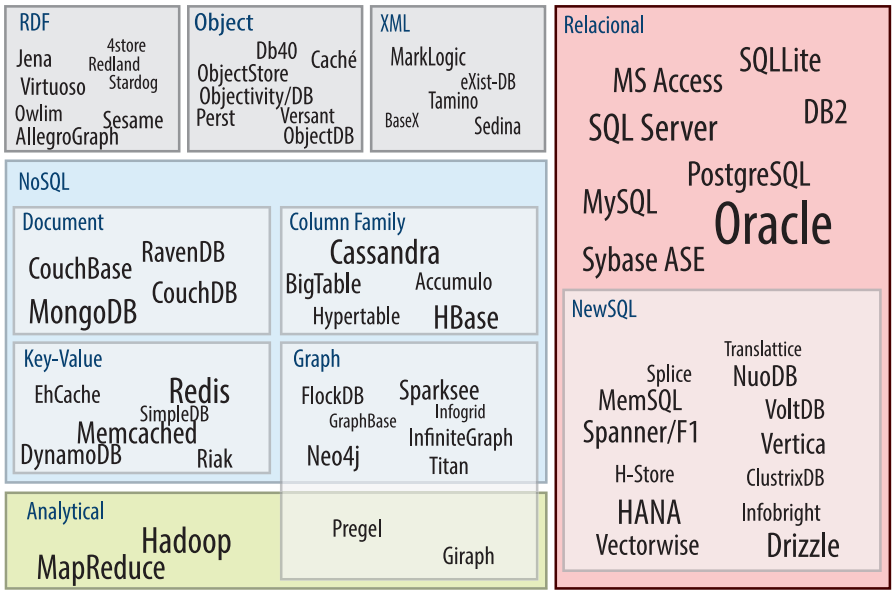

FIGURA 2.
PERSPECTIVA MODERNA DE BASES DE DATOS.

los sistemas principales de cada familia: los sistemas resaltados con letra de mayor tamaño son los que han sido mayormente adoptados[1].

En la esquina superior derecha, vemos a los sistemas de bases de datos relacionales tradicionales presentados en fuentes de gran tamaño. Estos siguen siendo la tecnología más prevalente, ofreciendo la seguridad de las garantías transaccionales (ACID) y un lenguaje de consulta poderoso (SQL). Sin embargo, estas características tienen un costo…

Tomando una base de datos moderna de código abierto en memoria (Shore) y un *benchmark* estándar (TPC-C), Harizopoulos [5] mostró que en su configuración original el sistema podía realizar 640 transacciones por segundo, pero que al apagar todas las funciones relativas a transacciones y persistencia como el *logging, latching, locking* y la gestión del *buffer*, el mismo sistema podía realizar 12.700 transacciones por segundo. Los autores estimaron que por lo tanto la base de datos sólo utilizaba un 6,8% del tiempo para realizar trabajo "útil" [5].

Por supuesto, este sistema ya no podía ofrecer garantías ACID, y el kilometraje para otros sistemas podría variar, pero en escenarios en los que tales requisitos pueden ser relajados, el mensaje central de este experimento anecdótico es que se pueden obtener enormes beneficios de rendimiento y escalabilidad reduciendo al mínimo el trabajo administrativo para las transacciones.

Siguiendo este razonamiento, y el antiguo refrán "La necesidad es la madre de todos los inventos", una nueva ola de bases de datos NoSQL (*Not Only SQL*) surgió de la era "Web 2.0", donde empresas como Facebook, Google, Amazon, Twitter, etc., se enfrentaron a un volumen y velocidad sin precedentes producidas por los datos aportados por los usuarios. Muchos de estos sistemas NoSQL se inspiraron en los Libros Blancos de compañías emergentes como Google y Amazon [1, 4] que describen las alternativas (a menudo ligeras) a las bases de datos relacionales que habían desarrollado para satisfacer estos nuevos desafíos.

NOSQL: NOT ONLY SQL

El objetivo de los repositorios NoSQL es permitir altos niveles de escalabilidad horizontal (a través de múltiples máquinas) y alto rendimiento al simplificar el modelo de base de datos, el lenguaje de consulta y las garantías de seguridad ofrecidas. Al relajar los requerimientos de ACID y SQL, y explotar la distribución, estos sistemas dicen permitir niveles de escalabilidad y rendimiento inalcanzables para las bases de datos relacionales tradicionales.

Con respecto a los modelos de datos, de acuerdo con la **Figura 2**, cuatro categorías principales de sistema NoSQL han aparecido:

- **Key-value:**
  Repositorios basados en un arreglo/map simple y asociativo. Tales sistemas permiten la búsqueda en una sola llave, devolviendo un valor que sigue un patrón. Los valores son a menudo objeto de versiones en vez de sobreescritura. Se pone énfasis en la distribución y replicación, normalmente siguiendo esquemas de hash compatibles [6]. Muchos de los sistemas en esta familia se hallan inspirados en los detalles del sistema *Dynamo* de Amazon [4], que se publicaron en 2007.

- **Document:**
  Repositorios basadas en un esquema *key-value*, pero donde los valores se refieren a "documentos" complejos sobre los cuales ciertas funciones *built-in* pueden ser ejecutadas. Una de los repositorios principales en esta familia es MongoDB, donde los valores son similares a documentos JSON.

1 Específicamente, para esto dependemos en gran medida en http://db-engines.com/en/ranking, para juzgar la popularidad de los sistemas, ranking que a su vez agrega menciones a los sistemas en páginas web, búsquedas en Google, perfiles de Linkedin, ofertas de trabajo en Google, sitios de preguntas y respuestas técnicas, Twitter, y así sucesivamente. En orden de mayor a menor, los tamaños de fuente en la Figura se refieren a las posiciones 1, 2-20, 21-50, 51-100, 101-150, 150+, resp. en la lista.

→

- **Column-family:**
  Repositorios que implementan una forma perezosa del modelo relacional a través de una abstracción *key-value*, donde las llaves son similares a las llaves primarias y los valores son multidimensionales y se ajustan a un esquema tabular flexible. *Key-values* con dimensiones similares se organizan en columna-*families*, similares a tablas relacionales. Las versiones se aplican típicamente a un nivel "celular". Así mismo las tablas están normalmente ordenadas por llave, permitiendo *range-queries* en los índices de prefijos. Estos repositorios están normalmente motivados por el diseño del sistema de BigTable de Google [1], cuyos detalles se publicaron en 2008.

- **Grafos:**
  Los repositorios que implementan adyacencias en sus índices de tal forma que atravesar de un dato/nodo a otro no requiere otra búsqueda en el índice, sino más bien un recorrido de los punteros. Lenguajes de consulta sobre caminos basados en expresiones regulares, y otros similares, permiten la navegación transitiva sobre los datos/nodos. Uno de los sistemas más importantes de esta familia es Neo4J.

Cada repositorio normalmente permite un lenguaje de consulta personalizado y ligero, que va desde búsquedas *key-version* para repositorios *key-value,* a las llaves con expresiones JSON/XML integradas para repositorios del tipo *document*, a una forma muy limitada y por tanto eficiente de SQL para los repositorios columna-*family*, a lenguajes con expresiones de camino para los *graphs*. Como un *trade-off* por la pérdida de SQL, los desarrolladores a menudo deben implementar *joins*, agregaciones y transacciones en el código de la aplicación.

En cuanto a las garantías que una base de datos distribuida puede ofrecer, el Teorema CAP establece que un sistema no puede garantizar la consistencia (acuerdo global sobre el estado/datos), disponibilidad (cada petición es atendida) y tolerancia a la partición (funcionalidad incluso si se pierden mensajes), todo al mismo tiempo. En un entorno NoSQL, la tolerancia a la partición es un objetivo fundamental ya que los datos pueden residir en cientos o miles de máquinas, aumentando la probabilidad de fallas.

A partir de entonces, los sistemas experimentan diferentes *trade-offs* entre la disponibilidad y la consistencia: una noción clave es consistencia eventual, en el cual las máquinas pueden converger hacia un acuerdo global solo después que una petición ha sido reconocida, traduciéndose en una mayor disponibilidad y mensajes reducidos, pero a costa de la consistencia. Por lo tanto si usted fuera Amazon, por ejemplo, en virtud de la consistencia eventual sus usuarios podrían ver los datos que tienen un día de antiguedad sobre las clasificaciones de productos, pero el sistema no va a rechazar nuevas calificaciones debido a mensajes de falla, y, eventualmente, todas las máquinas operativas verán estas nuevas clasificaciones. En el otro extremo del espectro, los protocolos de consenso —como los *commits* de dos o tres fases, o el algoritmo PAXOS de Lamport [9]— incurren en altos costos de comunicación, menos escalabilidad de escrituras a través de distintas máquinas, y pueden implicar tiempos de parada más frecuentes (menor disponibilidad), pero pueden garantizar nociones fuertes de consistencia.

El resultado de estos *trade-offs* que mejoran el rendimiento es el uso extendido de repositorios NoSQL en escenarios de uso intensivo de datos, de forma más prominente en empresas Web tales como Google, Facebook, Twitter, etc., pero también en muchas otras áreas: por ejemplo, repositorios centrados en documentos tales como MongoDB y CouchDB han sido utilizados en el CERN para manejar la gran cantidad de datos agregados producidos por los detectores de partículas en el Gran Colisionador de Hadrones [7].

NEWSQL: NO SOLO NOSQL

Los repositorios NoSQL han sido objeto de numerosas críticas por considerarse sobreutilizados y sobrepublicitados. Aunque este tipo de repositorios tienen, sin duda, casos de uso válidos, no todo el mundo se enfrenta a los mismos desafíos de uso intensivo de datos que Facebook o CERN. Su uso ingenuo puede implicar un riesgo innecesario de pérdida de datos o inconsistencia. La debilitación de las garantías de seguridad y los lenguajes de consulta de más bajo nivel incrementan la responsabilidad del desarrollador de la aplicación al tener que asegurarse que la base de datos se mantiene estable y que el rendimiento de las consultas más complejas es aceptable. Citando un reciente libro blanco de Google sobre la base de datos "Spanner":

> *"Creemos que es mejor que los programadores de aplicaciones se ocupen de los problemas de rendimiento debido al uso excesivo de las transacciones a medida que surgen cuellos de botella, en vez de estar siempre codificando en torno la falta de transacciones [2]".*

Aunque las características de las bases de datos relacionales de las cuales prescinden los repositorios NoSQL son computacionalmente caras, son importantes para muchas (aunque tal vez no todas) aplicaciones: fueron originalmente implementadas y añadidas a las bases de datos por una buena razón.

La familia de bases de datos NewSQL (representada en la **Figura 2**) ha surgido recientemente para lograr un compromiso entre las características de las bases de datos relacionales tradicionales y el desempeño de los repositorios NoSQL. Los sistemas NewSQL tienen como objetivo entregarle soporte a SQL y defender

ACID, pero a niveles mayores de rendimiento y escala que las bases de datos tradicionales (al menos en algunos escenarios fijos). El enfoque principal de NewSQL es diseñar bases de datos a partir de cero que exploten las arquitecturas de computadores modernas, haciendo use *cores* múltiples, y de grandes capacidades de memoria principal, GPU o clusters del tipo *shared-nothing*. Del mismo modo, muchos repositorios NewSQL optan por esquemas de indexación orientados a columnas que permiten una más eficiente agregación y filtrado sobre los valores de columnas completas que los repositorios tradicionales orientados por filas.

Sin embargo, las bases de datos no están diseñadas para el análisis de datos a gran escala, sino más bien para la ejecución de consultas en directo. Del mismo modo, muchas formas de *offline analytics* no requieren el gasto de construir y mantener los índices persistentes.

FRAMEWORKS ANALÍTICOS DISTRIBUIDOS

En la **Figura 2**, se incluye la categoría Analítica para *frameworks* de procesamiento distribuidos de datos: aunque no son, estrictamente hablando, bases de datos, ofrecen una alternativa a las bases de datos para realizar análisis.

Google propuso el *framework* MapReduce en 2004 [3] como una abstracción para la ejecución de tareas de procesamiento por lotes a gran escala sobre datos planos (no indexados/*raw* files) en un entorno distribuido. Muchos tipos diferentes de tareas de procesamiento distribuido (como las que utiliza Google, por ejemplo) tienen elementos en común: la partición de los datos a través de las máquinas, el apoyo a mecanismos de seguridad en caso de fallas de la máquina, ejecutar el procesamiento en cada máquina, mezclar los resultados de cada máquina en un resultado final, etc. Por lo tanto, el objetivo de MapReduce es ofrecer una interfaz que abstraiga de estos elementos comunes y permita el desarrollo a nivel superior de código de procesamiento distribuido.

Dado que MapReduce es considerado una de las principales tecnologías de Big Data, nos tomaremos un momento para obtener la esencia de cómo opera con el ejemplo ilustrado en la **Figura 3**. Los datos de entrada están ordenados por autor, e incluyen artículos y citaciones (note la repetición de los títulos de los artículos y las citas de cada autor). Podemos considerar esta tabla como una tabla plana (por ejemplo, un archivo CSV o TSV), además de ser muy grande: "frillones" de filas. Ahora nos gustaría saber quiénes son las parejas más productivas de coautores en nuestra tabla: queremos ver cuántas citas tiene cada par de coautores contando sólo aquellos documentos que han coescrito juntos. ¡Y tenemos un montón de máquinas para hacer esto!

MapReduce realiza agregación/*joins* en lotes mediante la ordenación de los datos. Tomando un ejemplo trivial, ya que los datos sin procesar de la entrada están ordenados por nombre de autor, podemos fácilmente obtener el recuento total de citas para cada autor, teniendo que almacenar en la memoria sólo al autor actual y su conteo actual, entregando este par de salida cuando el autor cambia. Si consideramos un entorno distribuido, aparte de la clasificación, tenemos que asegurarnos que todos los artículos de un autor terminan en la misma máquina.

Volviendo a nuestra no trivial tarea original de determinar los pares más productivos de coautores, MapReduce consta de dos fases principales: **Map** y **Reduce**. Lo primero que necesitamos figurarnos son los pares de coautores. Para hacer esto, tenemos que realizar un *join* distribuido sobre los artículos en términos de título/citas.

> `Map`$_1$: Para cada tupla A_i = (Author$_i$, Title$_i$, Citation$_i$) en la entrada, crear pares llave-valor (*key-value*) de la siguiente forma: (Title$_i$, <Author$_i$, Citations$_i$>), donde la llave es Title$_i$ y el valor es <Author$_i$, Citations$_i$>.

MapReduce asignará pares llave-valor con la misma llave a la misma máquina (por ejemplo, utilizando una función de hash en la llave). Esa máquina va a ordenar los pares de acuerdo a su llave utilizando la función `Map`. La partición y el ordenamiento son usualmente solucionados por el *framework*, lo que significa que el desarrollador no tiene que preocuparse acerca de qué datos van a cuál máquina física. Sin embargo, el valor por defecto de la partición y el ordenamiento pueden ser invalidados si es necesario.

Ahora, cada máquina tiene su propio *bag* de pares llave-valor ordenado por `Title`, con todos los autores de cada artículo disponible localmente. La siguiente es la fase de reducir:

> `Reduce`$_1$: En preparación para la reducción, la fase de ordenación agrupará los pares llave-valor (emitidos en `Map`$_1$) por título: (Title$_i$,{ <Author$_{i,1}$,Citations$_i$>; … ; <Author$_{i,n}$,Citations$_i$>}). Para cada uno de esos grupos, Reduce entregará como salida un conjunto de tuplas que represente cada par de coautores: {(<Author$_{i,j}$, Author$_{i,k}$>, Citations$_i$)| $1 \leq j < k \leq n$}.[2]

El resultado de la fase `Reduce` es un *bag* de pares únicos de coautores y sus citas para cada artículo (ya no necesitamos el nombre del artículo). Nuestra tarea final es sumar las citas totales por cada par de coautores. En la etapa anterior, los datos fueron ordenados/*joined* con respecto al título del artículo, por lo que ahora tenemos que realizar otra fase de `Map/Reduce`.

2 Asumimos Author$_x$ < Author$_y$ si y solo si x < y, etc., para mantener pares consistentes entre distintos artículos. →

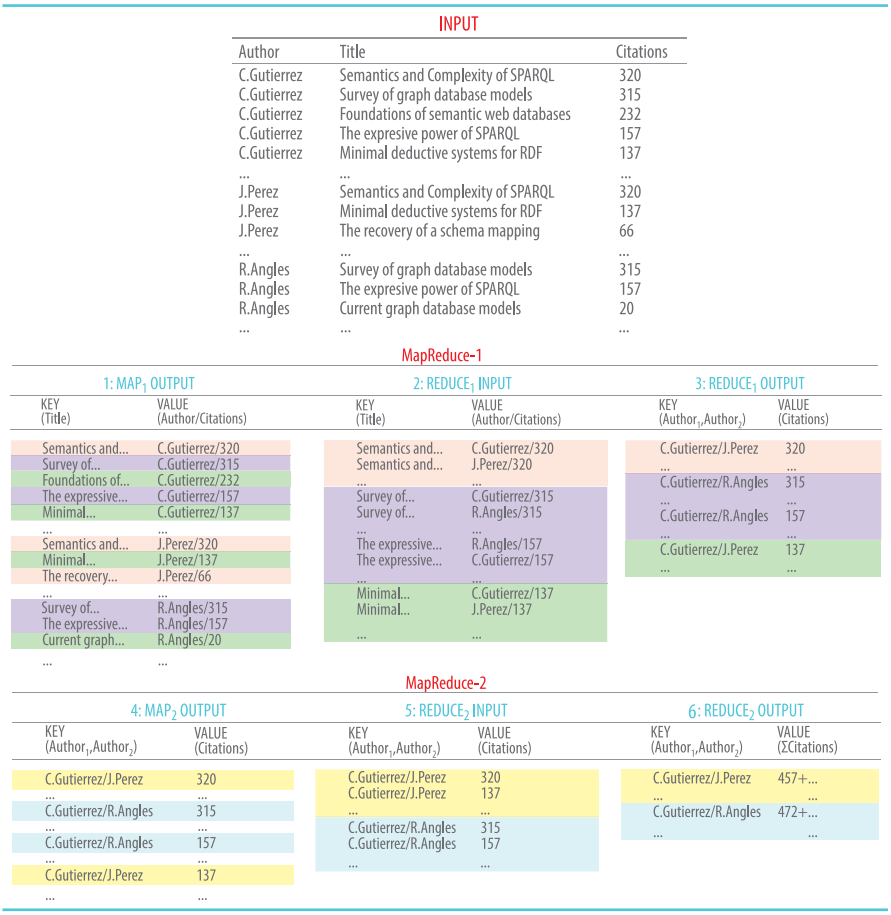

FIGURA 3.
UN EJEMPLO DE MAPREDUCE: ENCONTRAR LAS CITAS PARA CADA PAR DE COAUTORES (CUENTAN SOLO LOS ARTÍCULOS QUE HAN ESCRITO JUNTOS). LOS TONOS DIFERENTES DE LAS FILAS INDICAN UNA MÁQUINA DIFERENTE.

> `Map2`: Por cada tupla (<$Author_x$, $Author_y$>, $Citations_z$) creada en el paso intermedio, mapee el mismo par llave-valor.

Este mapa no transforma los valores, sino que simplemente se asegura que los mismos pares de coautores terminan ordenados en la misma máquina para la fase Reduce final.

> `Reduce2`: Con grupos (<$Author_x$, $Author_y$>,{$Citations_1$,..., $Citations_n$}), sume las citas en cada bag y produzca un solo par de salida: (<$Author_x$, $Author_y$>, $\sum_{z=1}^{n} Citations_z$).

Ahora tenemos nuestro resultado final. Si quisiéramos extraer resultados del tipo top-k, podríamos pasar a MapReduce un orden descendente y mapear el total de citas como llave.

En cuanto al esfuerzo de codificación, el desarrollador debe implementar las etapas de mapeo y reducción. Por su parte, MapReduce se hará cargo de balancear la carga, la tolerancia a fallas, y así sucesivamente. Con un *cluster* de máquinas que ejecuten el marco MapReduce, las tareas pueden ser ejecutadas por múltiples usuarios en paralelo y el marco tratará de hacer el mejor uso de los recursos disponibles. Del mismo modo, las tareas MapReduce son portátiles: los trabajos se pueden ejecutar en cualquier *cluster* de máquinas, siempre que el marco esté instalado (y los datos disponibles).

MapReduce permite la escalabilidad, pero, por supuesto, no la garantiza: las tareas intratables lo seguirán siendo en varias máquinas. En nuestro ejemplo, `Reduce1` produce un número cuadrático de pares de coautores por artículo ($\frac{n(n-1)}{2}$) , pero como las listas de autores son generalmente cortas, podemos esperar que `Reduce1` opere casi linealmente. Bajo la suposición de que las tareas se pueden descomponer en fases del tipo `Map-Reduce`, y que estas fases no son computacionalmente costosas o aumentan demasiado el volumen de los datos, MapReduce es una abstracción conveniente y de gran alcance para los que deseen trabajar con grandes volúmenes de datos.

Desafortunadamente, el sistema MapReduce es en sí propietario y se mantiene cerrado por Google. Sin embargo, el proyecto de código abierto Apache Hadoop ofrece una aplicación madura del marco y es de uso generalizado.

En la **Figura 2**, se puede observar otros dos sistemas en la intersección de **Graph** y **Analytical**: Pregel y Giraph. Ambos son marcos de estilo MapReduce diseñados específicamente para el procesamiento de grafos. Pregel fue introducido por Google en 2009 [8] para realizar computación distribuida sobre grafos de gran tamaño, y Apache Giraph es una implementación de código abierto de las mismas ideas (construidas sobre Hadoop). En resumen, el núcleo del modelo computacional de Pregel/Giraph es un sistema de intercambio de mensajes entre los vértices de un grafo. La computación está organizada en iteraciones. Un vértice puede leer los mensajes que le fueron enviados en la iteración anterior, puede cambiar su estado (el que puede ser continuo), y puede reenviar mensajes a otros vértices para la siguiente iteración. Los mensajes pueden ser enviados a cualquier vértice con un id localmente conocido, pero por lo general estos son los vértices que están enlazados. Estos marcos computacionales ofrecen una abstracción intuitiva para la aplicación de análisis basado en grafos –para ejecutar tareas tales como caminos más cortos, componentes conexas, *clustering*, medidas de centralidad como *PageRank*, etc.– mientras transparentemente distribuyen la computación: manejo del balance de carga, eficiente loteo (*batching*) de mensajes y tolerancia a fallas.

OTROS MODELOS: OBJETOS, XML, RDF

Con respecto a la **Figura 2**, nos quedan por tanto tres familias (relativamente tradicionales) de bases de datos en la esquina superior izquierda: `Object`, `XML` y `RDF`.

Las bases de datos de objetos (`Object`) permiten almacenar la información en la forma de objetos (de los que ocurren en el software orientado a objetos). La motivación principal de estas bases de datos es ofrecer la opción de persistencia para los objetos del software en un formato nativo. Típicamente, las bases de datos de objeto están hechas a la medida de un conjunto fijo de lenguajes de programación orientado a objetos, ofreciendo la posibilidad para una estrecha integración entre el ambiente de ejecución y la base de datos. Estas bases de datos vienen equipadas normalmente con un lenguaje de consulta que permite buscar objetos con ciertos campos o valores; de la misma forma, utilizan punteros entre los objetos para evitar el uso de *joins*. Los sistemas podrían ofrecer variadas otras funcionalidades, como versionamiento, restricciones de integridad, *triggers*, etc.

Las bases de datos XML son típicamente nativas, donde los datos son almacenados en una estructura de datos del tipo XML. Dada la relativa popularidad de XML como modelo para el intercambio de información y su uso en la Web (e.g., XHTML, RSS y ATOM *feeds*, mensajes SOAP, etc.), las bases de datos XML almacenan tales datos en un formato nativo que les permite ser consultadas directamente a través de lenguajes como XQuery, XPath y XSLT.

Las bases de datos RDF se enfocan en proveer funcionalidad de consulta sobre datos representados en el modelo básico de datos de la Web semántica. Tales repositorios implementan típicamente esquemas optimizados de almacenamiento para RDF y ofrecen funcionalidad de consulta usando el lenguaje SPARQL recomendado por la W3C. Además, estos motores podrían implementar funcionalidades relacionadas al razonamiento, comúnmente con respecto a los estándares RDFS y/o OWL.

Aunque estas tres familias de bases de datos han recibido atención significativa a nivel de investigación en la década pasada, como puede observarse en la **Figura 2**, permanecen siendo mayormente una tecnología de nicho cuando se consideran en el contexto más amplio de las tecnologías de bases de datos.

¿VARIEDAD?

Hemos hablado mucho acerca del volumen, e.g., escalabilidad a través de múltiples máquinas e.g., aumento en el rendimiento de escritura mediante la relajación de las garantías de consistencia. Sin embargo, no hemos hablado mucho sobre el tercer reto del Big Data: variedad. Mientras los problemas de volumen y velocidad se enlazan con el rendimiento y pueden ser abordados desde una perspectiva ingenieril mediante la composición de técnicas existentes tales como la partición horizontal, la replicación, ordenamientos distribuidos, *hashing* consistentes, compresión, *batching*, filtros de Bloom, árboles de Merkle, indexamiento orientado a columnas, etc. en el diseño de un sistema maduro, el problema de la variedad plantea cuestiones de carácter más conceptual.

Podría decirse que la familia de bases de datos NoSQL ofrece algunas ventajas sobre las bases de datos relacionales tradicionales cuando es necesario procesar conjuntos de datos diversos: al relajar el modelo relacional, los datos pueden ser almacenados de una forma no-normalizada "rápida y sucia". Aunque esto podría ahorrar tiempo para el manejo de esquemas, y es más flexible para trabajar con datos incompletos o irregulares, emplear modelos más simples nuevamente sobrecarga la capa de aplicación, en la cual los desarrolladores deben asegurarse que los datos permanecen consistentes y que los índices se hallan presentes para cubrir eficientemente la carga de trabajo esperada de las consultas.

De la misma forma, la variedad puede ser parcialmente solucionada por el rango de bases de datos disponibles hoy. Por ejemplo, uno podría usar una base de datos XML para almacenar datos XML, una base de datos relacional para los datos relacional, una base de datos de grafos para los datos estructurados en forma de grafo, y así sucesivamente. Múltiples bases de datos podrían entonces ser compuestas para atacar el problema de la variedad: sin embargo, la carga nuevamente está puesta sobre los desarrolladores de la aplicación que deben conectar las bases de datos y asegurarse que los datos permanecen consistentes a través de los distintos sistemas.

En general, el problema de la variedad es mucho más profundo que la forma o la incompletitud de los datos. Además, la variedad está lejos de ser un problema nuevo: varias subáreas de la Ciencia de la Computación han enfrentado el problema de la "integración de datos", ya sea combinando múltiples bases de datos en una, o combinando documentos desde millones de fuentes en la Web, o combinando repositorios de código fuente desde varios proyectos, o …

LOS DATOS ESTÁN EVOLUCIONANDO

Pareciera que el único enfoque práctico para resolver el cuello de botella en Big Data es repensar lo que queremos decir por "datos". Desde los días de papel y tinta de Snow, los datos han evolucionado a través del código Morse, tarjetas agujereadas, formatos binarios, ASCII, anotaciones, y así sucesivamente. La evolución de los datos ha ido siempre en la dirección de mejorar la lectura de las máquinas. Así como podría ser considerado imposible contar todas las

ocurrencias de la palabra "brócoli" en los libros almacenados en una biblioteca bien equipada (y trivial de hacerse sobre un archivo de texto electrónico), existen muchas tareas intensivas en datos que podríamos considerar imposible de realizarse hoy en día simplemente porque nuestra noción de "datos" no lo permite.

Actualmente, tenemos una plétora de modelos de datos y sintaxis de datos disponibles para permitir que la información sea estructurada y analizada sintácticamente. Sin embargo, en su mayoría, las máquinas necesitan código especializado en datos para interpretar los datos antes mencionados. Por ejemplo, los buscadores han sido construidos específicamente para interpretar datos relacionados con HTML.

En términos de la evolución de los datos, parece que la única forma de avanzar es hacer explícita la semántica de los datos que están siendo entendidos, de tal forma que las máquinas puedan, con profundidad creciente, procesar el significado de los datos. Las máquinas aún no pueden aprender bien del lenguaje natural ni adaptarse bien a nuevos problemas: por eso la semántica necesita hacerse explícita como parte de los datos para ayudar a vencer la verdadera variedad en los datos estructurados; i.e., la capacidad de incorporar datos imprevistos.

En términos de hacer explícita la semántica, podemos comenzar simplemente por usar identificadores globales para los objetos descritos en los datos de tal forma que una máquina pueda conocer, por ejemplo, que el "Boston" descrito en un conjunto de datos es el mismo que el "Boston" descrito en otro conjunto de datos (e.g., utilizando URIs como identificadores); ahora la máquina puede correr *joins* sobre los dos conjuntos. Aún mejor, si la máquina puede saber que el "Boston" en cuestión es la banda de música, no la ciudad de Estados Unidos (e.g., usando un sistema de clasificación) ahora la máquina puede entregar resultados acerca de "Boston" para preguntas sobre bandas de música. Aún mejor incluso si la máquina puede saber que las bandas de música tienen miembros, y típicamente lanzan álbumes y… (e.g., a través de una ontología del dominio).

BIG DATA SEMÁNTICO

Muchos de estos principios han sido explorados por la comunidad de la Web Semántica; en la opinión sesgada de este autor, las técnicas de la Web Semántica podrían aún cumplir un rol importante en términos de Big Data, particularmente para hacer frente a la diversidad. La Web Semántica ha sido largamente mirada en menos por mucho como un ejercicio académico estéril, condenada al fracaso debido a una serie de preocupaciones. Y hay un grado de verdad en esas preocupaciones. Muchas de las técnicas propuestas en el área de las ontologías y métodos de razonamiento deductivo no son adecuadas en escenarios con montones de datos, y ciertamente no para escenarios con montones de datos (Web) desordenados. Sin embargo, los desafíos no son irremontables: un poco de semántica puede llevarnos lejos.

Recientemente, compañías como Google, Facebook, Yahoo!, Microsoft, etc., están comenzando a usar partes de las tecnologías de la Web Semántica para potenciar nuevas aplicaciones. Por ejemplo, en junio de 2011, Bing, Google y Yahoo! anunciaron la ontología *schema.org* para que los *webmasters* puedan dejar datos estructurados disponibles en su sitio. Google ha usado bases de conocimiento semánticas en la construcción de su aplicación *Knowledge-graph*. El protocolo *Open Graph* de Facebook –que trata de crear un red descentralizada de datos– utiliza RDFa: un standard de la Web Semántica. Más recientemente, Google anunció soporte para anotaciones semánticas en Gmail usando la sintaxis JSON-LD. En tales casos, estas grandes compañías han virado hacia las tecnologías de la Web Semántica para construir aplicaciones sobre gigantescos conjuntos de fuentes diversas provenientes de millones de contribuyentes arbitrarios. Y esta tendencia de utilizar la semántica para hacerle frente a la diversidad parece que va a continuar.

CONCLUSIONES

BIG DATA ES UN LLAMADO A LOS CIENTÍFICOS DE LA COMPUTACIÓN A INVESTIGAR MÉTODOS PARA TRATAR DATOS CON MÁS VOLUMEN, MÁS VELOCIDAD Y MÁS VARIEDAD. AUNQUE LAS BASES DE DATOS RELACIONALES HAN SERVIDO COMO CABALLO DE BATALLA SEGURO POR MUCHOS AÑOS, LA GENTE ESTÁ COMENZANDO A DARSE CUENTA QUE SE NECESITAN NUEVAS ALTERNATIVAS PARA ENFRENTAR LOS DESAFÍOS VENIDEROS PLANTEADOS POR EL EMERGENTE DILUVIO DE DATOS.

LAS PRINCIPALES TECNOLOGÍAS BIG DATA QUE HAN APARECIDO HASTA EL MOMENTO SE CENTRAN PRINCIPALMENTE EN LOS DESAFÍOS DE VOLUMEN Y VELOCIDAD. LOS REPOSITORIOS NOSQL OFRECEN NUEVOS NIVELES DE ESCALABILIDAD MEDIANTE LA DISTRIBUCIÓN DEL MANEJO DE LOS DATOS EN MÚLTIPLES MÁQUINAS, LA RELAJACIÓN LAS GARANTÍAS ACID Y UTILIZANDO LENGUAJES DE CONSULTA MÁS LIVIANOS QUE SQL. NEWSQL BUSCA ENCONTRAR UN BALANCE OFRECIENDO ACID/SQL COMO LAS BASES DE DATOS RELACIONALES TRADICIONALES, AL MISMO TIEMPO QUE PRETENDE COMPETIR CON EL RENDIMIENTO Y ESCALA DE LOS SISTEMAS NOSQL. ASÍ MISMO, MARCOS DE PROCESAMIENTO DISTRIBUIDO COMO MAPREDUCE OFRECEN UNA ABSTRACCIÓN CONVENIENTE PARA CLIENTES QUE DESEEN REALIZAR TAREAS ANALÍTICAS DE GRAN ESCALA.

POR OTRO LADO, LA VARIEDAD SE MANTIENE COMO PROBLEMA ABIERTO. EN PARTICULAR, EL OBJETIVO DE SER CAPAZ DE CONSTRUIR APLICACIONES QUE PUEDAN ROBUSTAMENTE DESCUBRIR E INCORPORAR NUEVAS FUENTES DE DATOS SIGUE ELUDIÉNDONOS. PARA SOLUCIONAR ESTE PROBLEMA, QUIZÁ NECESITAMOS REPENSAR NUESTRA CONCEPTUALIZACIÓN DE LOS DATOS. COMPARADO CON LOS TIEMPOS DE JOHN SNOW, HOY EN DÍA APRECIAMOS LA CONVENIENCIA CON DATOS SERIALIZADOS EN UN FORMATO ELECTRÓNICO QUE PUEDE SER LEÍDO POR UNA MÁQUINA. DE LA MISMA FORMA, TENER DATOS CON UNA SEMÁNTICA EXPLÍCITA PODRÍA CONVERTIRSE EN LA NORMA EN LAS SIGUIENTES DÉCADAS. ■

BIBLIOGRAFÍA

[1] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A Distributed Storage System for Structured Data. ACM Trans. Comput. Syst., 26(2), 2008.

[2] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. C. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford. Spanner: Google's Globally-Distributed Database. In OSDI, pages 261-264, 2012.

[3] J. Dean and S. Ghemawat. MapReduce: Simplifed Data Processing on Large Clusters. In OSDI, pages 137-150, 2004.

[4] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. In SOSP, pages 205-220, 2007.

[5] S. Harizopoulos, D. J. Abadi, S. Madden, and M. Stonebraker. OLTP through the looking glass, and what we found there. In SIGMOD Conference, pages 981-992, 2008.

[6] D. R. Karger, E. Lehman, F. T. Leighton, R. Panigrahy, M. S. Levine, and D. Lewin. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In STOC, pages 654-663, 1997.

[7] V. Kuznetsov, D. Evans, and S. Metson. The CMS data aggregation system. In ICCS, number 1, pages 1535-1543, 2010.

[8] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In SIGMOD Conference, pages 135-146, 2010.

[9] M. C. Pease, R. E. Shostak, and L. Lamport. Reaching agreement in the presence of faults. J. ACM, 27(2):228-234, 1980.

[10] M. Stonebraker. One size fits all: an idea whose time has come and gone. Commun. ACM, 51(12):76, 2008.