Ciencia e Ingeniería Neogranadina
ISSN: 0124-8170
revistaing@unimilitar.edu.co
Universidad Militar Nueva Granada
Colombia

Moreno Arboleda, Francisco Javier; Quintero Rendón, Juan Esteban; Rueda Vásquez, Robinson

UNA COMPARACIÓN DE RENDIMIENTO ENTRE ORACLE Y MONGODB
Ciencia e Ingeniería Neogranadina, vol. 26, núm. 1, enero-junio, 2016, pp. 109-129
Universidad Militar Nueva Granada
Bogotá, Colombia

Disponible en: http://www.redalyc.org/articulo.oa?id=91145342002

UNA COMPARACIÓN DE RENDIMIENTO ENTRE ORACLE Y MONGODB

A PERFORMANCE COMPARISON BETWEEN ORACLE AND MONGODB

Francisco Javier Moreno Arboleda[1], Juan Esteban Quintero Rendón[2], Robinson Rueda Vásquez[3]

Fecha de recepción: 24 de septiembre de 2015
Fecha de aprobación: 15 de marzo de 2016

Referencia: F. J. Moreno Arboleda, J. E. Quintero Rendón, R. Rueda Vásquez. (2016). Una comparación de rendimiento entre Oracle y MongoDB. Ciencia e Ingeniería Neogranadina, 26 (1), pp. 109-129, DOI: http://dx.doi.org/10.18359/rcin.1669

RESUMEN

La creciente y enorme cantidad de datos, del orden de exabytes, generados por las aplicaciones empresariales actuales han originado conjuntos masivos de estos. Los sistemas de gestión de bases de datos (SGBD) NoSQL han surgido como una alternativa a los SGBD relacionales para la gestión de estos conjuntos. Entre los principales SGBD NoSQL está MongoDB. En este artículo se compara el rendimiento entre MongoDB y Oracle (uno de los principales SGBD que soporta bases de datos relacionales). La comparación se basa en las operaciones de inserción, consulta, actualización y borrado (CRUD, por sus siglas en inglés). Aunque se requieren experimentos más exhaustivos y muchos otros tipos de pruebas, los resultados ofrecen un punto de partida para el análisis de rendimiento en estos SGBD.

Palabras clave: bases de datos, bases de datos NoSQL, MongoDB, Oracle, rendimiento.

ABSTRACT

The growing and huge amount of data, of the order of exabytes, generated by current enterprise applications have originated massive datasets. Non-relational database management systems

1. Ing. Sistemas, Dr. en Ingeniería de Sistemas, profesor asociado, Universidad Nacional de Colombia, Facultad de Minas, Medellín, Colombia, fjmoreno@unal.edu.co
2. Ing. Sistemas, Universidad Nacional de Colombia, Facultad de Minas, Medellín, Colombia, juequinterore@unal.edu.co
3. Ing. Sistemas, Universidad Nacional de Colombia, Facultad de Minas, Medellín, Colombia, rvrobinson@unal.edu.co

(DBMS) have emerged as an alternative to relational DBMS to manage these sets. Among the main non-relational DBMS is MongoDB. In this paper, we compare the performance between MongoDB and Oracle (one of the main DBMS that supports relational databases). The comparison is based on the CRUD operations (create, read, update, and delete). Although more exhaustive experiments and many other types of tests are required, the results give a starting point for the performance analysis of these DBMS.

Keywords: databases, NoSQL databases, MongoDB, Oracle, performance.

1. INTRODUCCIÓN

El término bases de datos (BD) no-relacionales se refiere a las BD que no satisfacen el modelo relacional. Esta denominación puede llevar a malinterpretaciones, ya que sugiere que son BD (o para ser más precisos, sistemas de gestión de BD, SGBD) que carecen de las características disponibles en un SGBD relacional (cuyo lenguaje de gestión de datos es usualmente SQL); por ello se prefiere el nombre SGBD Not Only SQL (SGBD NoSQL) [1] para enfatizar que aunque estos SGBD carecen de algunas características propias de los SGBD relacionales, también suelen incluir características no disponibles en estos últimos.

Por ejemplo, la operación relacional reunión (*join*), disponible en el lenguaje SQL, suele no estarlo en la mayoría de los lenguajes de los SGBD NoSQL. Por otro lado, MapReduce (ver sección 4) es un paradigma de programación que suele estar presente en algunos SGBD NoSQL y que no es propio del lenguaje SQL. Otras diferencias se indican adelante en esta sección.

Aunque actualmente este tipo de BD está adquiriendo popularidad, el término no es nuevo. En 1998 Strozzi [2] empleó el término BD NoSQL para referirse a aquellas BD (SGBD) que no usaban el lenguaje SQL para la gestión de datos. En 2009 se retoma este término para referirse a las BD que comenzaron a surgir como una alternativa a las BD relacionales [3]. Las BD NoSQL han tomado relevancia recientemente debido al flujo creciente de datos en internet. Este crecimiento ha originado el término *Big Data* [4]. Aunque existen varias definiciones, *Big Data* se refiere a enormes volúmenes de datos (estructurados, semiestructurados y no-estructurados) del orden de exabytes (10^{18} bytes) cuyo almacenamiento y análisis (*e.g.*, análisis textual [5] de mensajes de correo, *tweets*, blogs) se puede hacer mediante BD especializadas, entre estas las BD NoSQL. Por ejemplo, hay sistemas que han generado exabytes de datos provenientes de sensores, con cientos de millones de datos espaciales, temporales y sociales [6].

Las características principales de estas BD son: a) el esquema (estructura) de los datos no está necesariamente predefinido, b) se permite que el valor de un atributo sea multivaluado (arreglo de valores), y c) suele haber redundancia de datos (debido a la poca normalización). Esto puede conllevar a que operaciones como la reunión sean innecesarias.

Las características a) y b) son las que diferencian este tipo de base de datos de las relacionales. Una relación tiene dos componentes: un esquema predefinido, *i.e.*,

un conjunto de parejas (nombre_atributo, tipo de datos) y un conjunto de filas que se apegan a dicho esquema. De esta forma, la característica a) viola este principio, ya que las filas de una misma colección (el término equivalente en algunas BD NoSQL a una relación) no se apegan necesariamente a un esquema predefinido. Por otro lado, una propiedad del modelo relacional establece que el valor de un atributo debe ser atómico [7]; así la característica b) viola este principio (sin embargo, algunos autores [8] sugieren que la noción de atomicidad es ambigua y elusiva y aceptan atributos multivaluados en el modelo relacional). Nótese que la característica c) no es un elemento diferenciador porque una BD relacional también podría tener redundancias (*e.g.*, relaciones que solo están en primera forma normal), y aun así seguir siendo una BD relacional.

Una manera de clasificar los SGBD NoSQL es la forma en que almacenan los datos [9].

- **Clave/valor:** el almacenamiento se hace mediante estructuras con dos componentes: una clave y un valor. Ejemplo: {"nombre": "Juan"}, donde la clave es "nombre" y su valor es "Juan". Entre los SGBD NoSQL de este tipo están Redis, Dynamite! y Voldemort DB.
- **Columna:** el almacenamiento se hace por columnas. Es decir, una fila de datos corresponde a un grupo de valores para un mismo atributo, e.g., como se muestra en la Tabla 1. Por otro lado, en una BD relacional una fila de datos corresponde a un grupo de valores, uno para cada atributo del esquema, e.g., como se muestra en la Tabla 2. Entre los SGBD NoSQL de este tipo están Apache Hbase, Cassandra y Hypertable.
- **Grafo:** el almacenamiento se hace mediante grafos. En los nodos se almacenan entidades, las cuales se relacionan por medio de aristas. Ejemplo: ver Figura 1. Entre los SGBD NoSQL de este tipo están Neo4j, OrientDB e InfiniteGraph.

Tabla 1. Ejemplo de filas de empleados en una BD NoSQL orientada a columnas

1	2	3	→ Una fila de **ids.**
Juan	Pablo	Ana	→ Una fila de **Nombres.**
23	25	21	→ Una fila de **Edades.**
8	10	6	→ Una fila de **Grados.**
10	10	20	→ Una fila de **Deps.**

Tabla 2. Ejemplo de filas de empleados en una BD relacional orientada a filas

ID	Nombre	Edad	Grado	Dep	
1	Juan	23	8	10	→ Un empleado.
2	Pablo	25	10	10	→ Un empleado.
3	Ana	21	6	20	→ Un empleado.

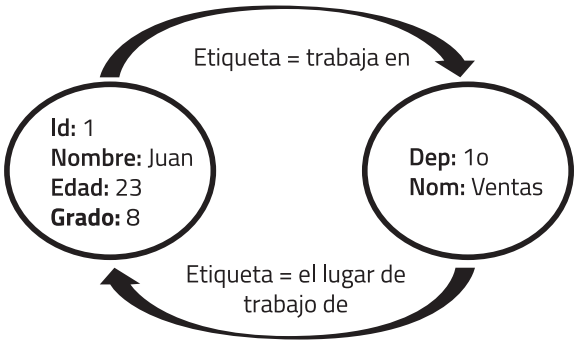

Figura 1. Representación de datos en una BD NoSQL de tipo grafo

- **Documento**: el almacenamiento se hace en colecciones de documentos, *e.g.*, los

documentos JSON (JavaScript Object Notation) los cuales se explican en la Sección 2. Entre los SGBD NoSQL de este tipo están MongoDB y Apache CouchDB.

En este artículo se compara el rendimiento entre MongoDB (un SGBD NoSQL) y Oracle (un SGBD que soporta BD relacionales) para las operaciones CRUD [10] (*create, read, update y delete*) inserción, consulta, actualización y borrado sobre una sola relación/colección y para una operación de consulta binaria, *i.e.* que involucra dos relaciones/colecciones, como la reunión. Se eligió Oracle porque es uno de los tres principales SGBD relacionales en el mercado, junto con SQL Server y DB2 [11] (véase también la clasificación que ofrece http://db-engines.com/en). Por otro lado, MongoDB es uno de los tres principales SGBD NoSQL junto con Cassandra y Hbase [12] (de nuevo, véase http://db-engines.com/en). Además, se eligieron las operaciones CRUD porque son las principales para la gestión de datos en las aplicaciones de BD. En particular, la operación de reunión (*join*) es la operación binaria más frecuentemente usada en aplicaciones de BD [13]. Esta comparación ofrece un punto de partida para los diseñadores y desarrolladores a la hora de elegir un SGBD para una aplicación. Aunque los resultados no se pueden generalizar, estos concuerdan con las ideas presentadas en [14], donde se sugiere que entre los aspectos que inciden para esta elección es determinar si la aplicación es de lectura o de escritura intensiva (*read-heavy applications vs. write-heavy applications*). Además, se considerará si se debe dar prioridad al desempeño frente a las anomalías potenciales en los datos (en MongoDB los controles de integridad son mínimos, *e.g.*, no hay controles de integridad referencial) y al cumplimiento de las propiedades de las transacciones (propiedades ACID: atomicidad, consistencia, aislamiento [isolation] y durabilidad), igualmente mínimo en MongoDB.

El artículo está organizado así: en la Sección 2 se presentan los elementos básicos de MongoDB, de su lenguaje de gestión de datos y se compara con SQL. En la Sección 3 se explica cómo llevar a cabo la operación de reunión en MongoDB mediante MapReduce. En la Sección 4 se presentan los experimentos y resultados. En la Sección 5 se muestran trabajos relacionados. En la Sección 6 se concluye el artículo y se proponen trabajos futuros.

2. MongoDB, SU LENGUAJE Y SQL

MongoDB es un SGBD NoSQL de tipo documento, el cual usa documentos JSON [15]. JSON es un formato para el intercambio de datos similar a XML pero su estructura es más simple. Un ejemplo de un documento JSON es:

```
{
        "id": 1,
        "nombre": "Juan",
        "edad": 23,
        "grado": 8,
        "dep": 10
}
```

Un documento JSON está encerrado entre llaves {}. En su interior hay parejas "clave": valor, separadas por comas, donde valor puede ser un número, una cadena de caracteres, un documento JSON o un arreglo de valores (que incluso pueden ser documentos JSON). Si el valor es un arreglo, este se encierra entre corchetes [] y en su interior se colocan sus valores separados por comas.

En MongoDB los documentos JSON se agrupan en una colección. Una colección es el equivalente a una relación en un SGBD relacional. Los documentos JSON de una misma colección no se apegan necesariamente al mismo esquema. Un documento equivale a una fila de una relación y una clave de un documento equivale a un atributo de una relación. En la Figura 2 se establece un paralelo entre los términos de una BD relacional y una BD en MongoDB.

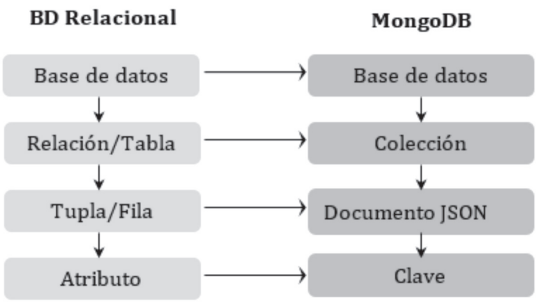

Figura 2. Comparación de términos entre una BD relacional y una BD orientada a documentos: MongoDB

En la Tabla 3 se presenta la forma general de las sentencias básicas de definición del esquema de datos en SQL y en MongoDB, y en la Tabla 4 ejemplos de estas sentencias. En particular, la función *ensureIndex*() de MongoDB recibe como parámetro una pareja "clave": valor. La clave indica el atributo que se indexará, y el valor puede ser 1 o -1. El 1 indica que el orden es ascendente, y el -1 que es descendente.

En la Tabla 5 se presenta la forma general de las sentencias de inserción, actualización y borrado de datos en SQL y en MongoDB. En MongoDB la sentencia *insert* recibe como parámetro un documento JSON. La clave id equivale a la clave primaria en el modelo relacional; si no se indica MongoDB, la asigna automáticamente.

En la Tabla 6 se presentan ejemplos de las sentencias de inserción, actualización y borrado de datos en SQL y en MongoDB. Para los ejemplos de actualización se consideran los datos de la relación/colección de empleados de las Tablas 1 y 2.

Tabla 3. Formas generales de las sentencias básicas de definición del esquema en SQL y en MongoDB

Operación	SQL	MongoDB
Creación base de datos	**CREATE DATABASE** nombreBD;	**USE** nombreBD
Creación tabla (en SQL) / Colección (en MongoDB)	**CREATE TABLE** nombreTabla (atributo1 tipo_de_dato restricciones <, atributo2...>);	**db.createCollection**("nombreColección")
Creación índice	**CREATE INDEX** nombreIndice **ON** nombreTabla (atributo1 <, atributo2... >);	**db.nombreColeccion.ensureIndex**({"atributo": 1})
Destrucción tabla (en SQL) / Colección (en MongoDB)	**DROP TABLE** nombreTabla;	**db.nombreColeccion.drop()**

Tabla 4. Ejemplos de las sentencias de definición del esquema en SQL y en MongoDB

SQL	MongoDB
CREATE DATABASE empresa;	**USE empresa**

CREATE TABLE dpto (
dep NUMBER(3) **PRIMARY KEY**,
nom VARCHAR(20) **NOT NULL**);
 	
CREATE TABLE empleado (
id NUMBER(4) **PRIMARY KEY**,
nombre VARCHAR(20) **NOT NULL**,
edad VARCHAR(20) **NOT NULL**,
grado NUMBER(3) **NOT NULL**,
dep NUMBER(3) **REFERENCES** dpto);
	db.createCollection(**"dpto"**)
	
	db.createCollection(**"empleado"**)

CREATE INDEX indiceDep **ON** empleado (dep);	db.empleado.ensureIndex({**dep: 1**}) **//**Suponiendo que empleado tiene una clave dep.
DROP TABLE **empleado;**	db.empleado.drop()

Tabla 5. Formas generales de las sentencias de inserción, actualización y borrado de datos en SQL y en MongoDB

Operación	SQL	MongoDB
Inserción	**INSERT INTO** nombreTabla **VALUES** (valorAtributo1, <, valorAtributo2...>);	db.nombreColeccion.insert({ atributo1: valorAtributo1 <, atributo2: valorAtributo2, ...>})
Actualización	**UPDATE** nombreTabla **SET** atributo = nuevoValor <**WHERE** condición>	db.nombreColeccion.update({<condición>}, {$set: {atributo: nuevoValor} }<, [true|false],[true|false]>)
Borrado	**DELETE** **FROM** nombreTabla <**WHERE** condición>	db.nombreColeccion.remove({<condición>})

Tabla 6. Ejemplos de las sentencias de inserción, actualización y borrado de datos en SQL y en MongoDB

SQL	MongoDB	Resultados en MongoDB
INSERT INTO dpto **VALUES**(10, 'Ventas'); **INSERT INTO** empleado **VALUES**(1, 'Juan', 23, 8, 10);	db.dpto.insert({_id: 10, nom: "Ventas"}) db.empleado.insert({_id: 1, nombre: "Juan", edad: 23, grado: 8, dep: 10})	Los datos son insertados.
UPDATE empleado **SET** grado = 9 **WHERE** edad = 25;	db.empleado.update({edad:25}, {$set: {grado: 9}})	{**"_id"**: 1, **"nombre"**: "Juan", **"edad"**: 23, **"grado"**: 8, **"dep"**: 10} {**"_id"**: 2, **"nombre"**: "Pablo", **"edad"**: 25, **"grado"**:9, **"dep"**: 10} {**"_id"**: 3, **"nombre"**: "Ana", **"edad"**: 11, **"grado"**: 6, **"dep"**: 20}
DELETE **FROM** empleado **WHERE** edad >= 18;	db.empleado.remove({edad: {$gt:18}})	{**"_id"**: 3, **"nombre"**: "Ana", **"edad"**: 11, **"grado"**: 6, **"dep"**: 20}

Nótese que la inserción en MongoDB de la Tabla 6 simula el comportamiento de una clave foránea. Una segunda opción es almacenar las dos piezas de datos en un único documento JSON, así (por simplicidad se muestra un solo empleado en el arreglo):

```
db.dpto.insert({
        dep: 10, nom: "Ventas",
        empleados: [{_id: 1, nombre: "Juan", edad: 23, grado: 8, dep: 10}]
})
```

La forma esencial de una consulta en SQL es:

```
SELECT <DISTINCT> [*atributo1 <, atributo2...>]
FROM relación1 <, relación2...>
WHERE condición
GROUP BY atributos
HAVING condición de grupo;
```

Y en MongoDB es:

```
db.collection.find(<condición> <, proyección>)
```

donde db es la palabra clave para hacer operaciones sobre la base de datos actual, y collection indica la colección a consultar.

La función find() recibe dos parámetros: una condición y una proyección. La condición es equivalente a la cláusula WHERE de SQL pero

aplicada a los documentos. La proyección es equivalente a la sentencia SELECT de SQL, allí se indican los atributos (claves en MongoDB) que se desean seleccionar. En la Tabla 7 se presentan ejemplos de consultas en SQL y en MongoDB.

3. LA REUNIÓN EN MongoDB MEDIANTE MapReduce

MongoDB no posee un operador propio (nativo) para reunir (join) dos colecciones, como las presentadas en la Tabla 6. Como MongoDB es un SGBD NoSQL y en este tipo de BD suele haber cierto tipo de redundancia (debido a la poca normalización) esto hace usualmente innecesaria la ejecución de operaciones como la reunión. Los arreglos, como en la segunda opción que se planteó en la Sección 2, también pueden hacer innecesarias las reuniones. Sin embargo, si se requiere hacer la reunión de dos colecciones (como las de la Tabla 6) esta se puede lograr en MongoDB mediante MapReduce.

MapReduce es un paradigma de programación paralela en arquitecturas distribuidas cuyo objetivo es la solución de problemas mediante algoritmos que exploten el paralelismo [16]. MapReduce suele ser usado en aplicaciones que manejan enormes volúmenes de datos (del orden de exabytes). MapReduce toma su nombre de dos funciones Map y Reduce. Estas funciones deben ser programadas para resolver cada problema específico.

3.1 Función Map

La función Map se aplica a un conjunto de datos y retorna una lista de parejas (clave: valor), no

Tabla 7. Ejemplos de las consultas en SQL y en MongoDB

SQL	MongoDB	Resultado en MongoDB
SELECT * **FROM** empleado **WHERE** edad < 30 **AND** grado = 8;	**db.empleado.find(**{edad: {**$lt**: 30}, grado: 8}**)**	{"_id": 1,"nombre": "Juan", "edad": 23,"grado": 8, "dep": 10}
SELECT * **FROM** empleado **WHERE** nombre **IN** ('Pablo', 'Ana');	**db.empleado.find(** {nombre: {**$in**: ["Pablo", "Ana"] } }**)**	{"_id": 2,"nombre": "Pablo", "edad": 25, "grado": 10, "dep": 10} {"_id": 3, "nombre": "Ana", "edad": 11, "grado": 6, "dep": 20}
SELECT * **FROM** empleado **WHERE** nombre **LIKE** '%blo';	**db.empleado.find(**{nombre:/.blo/} **)**	{"_id": 2, "nombre": "Pablo", "edad": 25, "grado": 10, "dep": 10}
SELECT SUM(edad**) AS** total **FROM** empleado;	**db.empleado.aggregate(**[{**$group**: {_id: null, total: {**$sum**: "$edad"} } }]**)**	[{"_id": null, "total": 69 }]
SELECT dep, **MAX(**grado**) AS** mg **FROM** empleado **GROUP BY** dep;	**db.empleado.aggregate(**{**$group**: {_id: "**$**dep", mg: {**$max**: '**$**grado'} } }**)**	[{"_id": 10,"max": 10}, {"_id": 20,"mg": 6}]

necesariamente distintas, donde clave y valor pueden ser tan complejos como se requiera. El conjunto original de datos se particiona en *n* fragmentos, cada fragmento se asigna a un procesador donde se ejecuta la función Map, como se muestra en la Figura 3. Luego, la lista de parejas generada por cada función Map se envía a un proceso controlador.

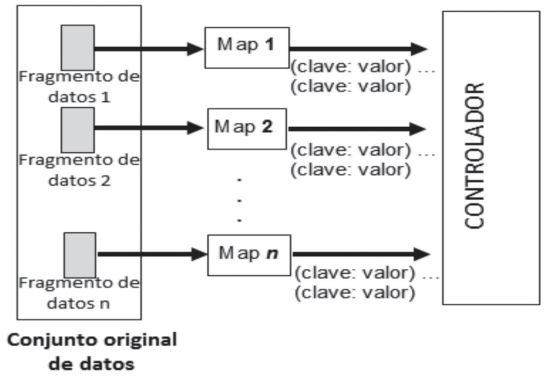

Figura 3. Etapa inicial de MapReduce

El proceso controlador une todos los valores de una misma clave en un arreglo, *i.e.*, se genera un conjunto de parejas (clave: arreglo de valores). Este conjunto de parejas se divide en m subconjuntos, cada subconjunto de parejas se asigna a un procesador donde se ejecuta la función Reduce.

3.2 Función Reduce

La función Reduce se aplica a cada pareja (clave: arreglo de valores) de cada subconjunto de parejas, y a partir de esta se genera una o varias parejas de la forma (clave: valor), véase la Figura 4. Usualmente, el valor asociado con una clave se genera a partir de una operación de agregación que se aplica al arreglo de valores, *e.g.*, la suma o el promedio de todos los valores en el arreglo. Sin embargo, el valor (y la clave) pueden ser tan complejos como se requiera.

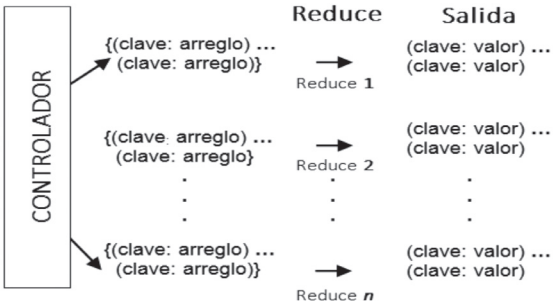

Figura 4. Etapa final de MapReduce

Ejemplo 1:
A continuación se mostrará cómo obtener la reunión (⋈) de dos relaciones mediante MapReduce. La siguiente explicación está basada en la propuesta de Leskovec [17]. Considérense las relaciones Emp y Dpto de la Figura 5, donde Id es la clave primaria de Emp y Dep la de Dpto. Además, el atributo Dep en Emp es una clave foránea con respecto al atributo Dep de Dpto.

Emp	
Id	Dep
1	10
2	10
3	20

Dpto	
Dep	Nom
10	V
20	P

Figura 5. Tablas Emp y Dpto

La función Map toma las filas de cada relación y genera la siguiente lista de parejas: para las filas de la relación Emp: (10: (E, 1)), (10: (E, 2))

y (20: (E, 3)). Es decir, se genera una lista de parejas (clave: valor), donde clave es dep y valor está compuesto por el nombre de la relación (abreviada con la letra E) y ced (la cédula). Para el caso de las filas de la relación Dpto la función Map genera: (10: (D, V)) y (20: (D, P)).

Por tanto, al controlador llegan cinco parejas, el cual las agrupa y genera las siguientes dos parejas (clave: arreglo) que son enviadas a la función Reduce: (10, [(E, 1), (E, 2), (D, V)]) y (20, [(E, 3), (D, P)]).

La función Reduce recibe estas dos parejas y a partir de cada una, genera parejas de la forma (clave: valor), donde la clave es dep y valor está compuesto por la cédula y por el nombre del departamento.

A partir de la pareja (10, [(E, 1), (E, 2), (D, V)]) se generan las parejas (10, (1, V)) y (10, (2, V)). Nótese que la función Reduce genera cada empleado de este arreglo combinado con los datos de su departamento (V), de allí la necesidad de saber cuáles parejas del arreglo provienen de la relación Emp o de la relación Dpto (por ello se incluye en cada pareja del arreglo el nombre abreviado de la relación). A partir de la pareja (20, [(E, 3), (D, P)]) se genera la pareja (20, (3, P)). Estas tres parejas conforman la reunión de las relaciones Emp y Dpto. En la Figura 6 se ilustra el proceso.

Para ejecutar MapReduce en MongoDB se procede así. Se programan las funciones Map y Reduce en JavaScript:

```
var nombreFunciónMap = function(){...}

var nombreFunciónReduce = function(){...}
```

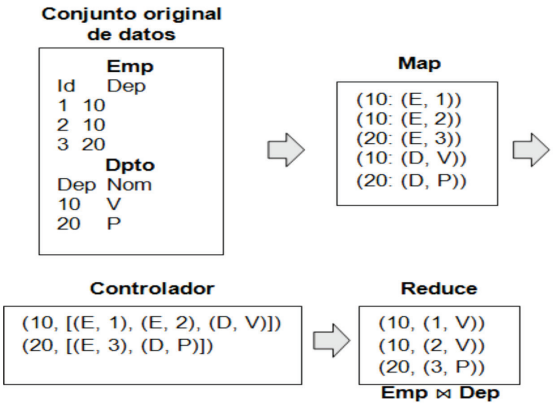

Figura 6. Ejemplo de la reunión usando MapReduce

En el Apéndice 1 se muestra el código correspondiente a las funciones Map y Reduce para hacer la reunión de las colecciones Emp y Dpto, correspondientes a las colecciones de la Tabla 6. Una vez se han programado estas funciones se usan así:

```
db.runCommand{ mapreduce: 'nomColecciónOrigen'
               map: nomFunciónMap
               reduce: nomFunciónReduce
               out: 'nomColecciónDestino' }
```

Donde nomColecciónDestino es la colección donde quedarán los resultados de MapReduce. Sin embargo, en MongoDB hay una dificultad para hacer la reunión de dos colecciones (como las presentadas en la Tabla 6). En la sintaxis se observa que MapReduce solo acepta como datos de origen una colección (nomColecciónOrigin). Para solucionar este problema se pueden unir las dos colecciones antes de invocar a MapReduce. La unión de dos colecciones en MongoDB se obtiene mediante la función copyTo(): db.nomColección1.copyTo(nomColeclón2), donde nomColeclón2 será la colección donde quedará la unión de los documentos de las dos colecciones.

Nótese que aunque las colecciones de la Tabla 6 tienen dos atributos en común (_id y Dep) para la reunión solo se considera la clave Dep como el atributo de reunión.

4. EXPERIMENTOS Y RESULTADOS

A continuación se presenta una serie de pruebas de rendimiento entre los SGBD Oracle y MongoDB. Para las pruebas se usó un equipo con procesador Intel® Core (TM) i7 de 1.8 GHz y con 4 GB de memoria RAM. Se usó la versión 11g Express Edition de Oracle y la versión 3.2.4 de MongoDB. El sistema operativo fue Windows 8. Para tratar de garantizar en lo posible igualdad de condiciones en la plataforma de pruebas, se verificó según los sitios oficiales tanto de Oracle (*www.oracle.com*) como de MongoDB (*www.mongodb.org*), que este sistema operativo es uno de los recomendados para el desempeño óptimo de estos productos. Las pruebas se hicieron para las cuatro operaciones: inserción, actualización, consulta y borrado. También se hicieron pruebas para la operación de reunión.

Cada prueba se ejecutó diez veces para cada tamaño de muestra elegido. Para la tabla/colección empleado (ver Tablas 4 y 6) los tamaños fueron 1.000, 5.000, 10.000, 50.000, 100.000 y 500.000 filas/documentos; y se tomó el promedio del tiempo de ejecución para cada tamaño dado. Se consideró, además, que cuando una misma consulta se ejecuta repetidamente en un SGBD, algunos de sus resultados pueden permanecer en la memoria caché, esto hace que el tiempo de ejecución de las últimas ejecuciones sea menor con respecto a las primeras. Para evitar que los resultados se afectasen por este aspecto, se "limpió" la memoria caché *antes* de la ejecución de cada consulta.

Para las operaciones de inserción los resultados se muestran en la Figura 7. Los resultados favorecieron a MongoDB. El mayor número de verificaciones de consistencia y de integridad (claves primarias, foráneas, checks de no nulidad, entre otras) y la gestión de transacciones podría explicar el mayor tiempo requerido por Oracle [18].

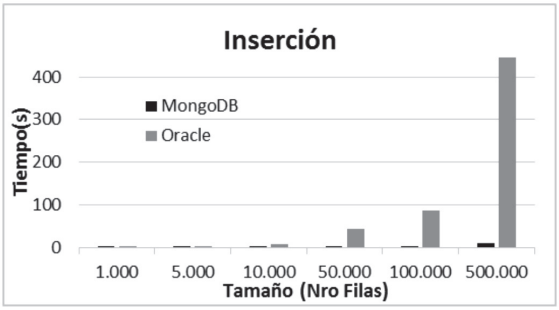

Figura 7. Resultados de las pruebas de inserción

Para la operación de actualización se evaluaron tres casos donde la operación afectaba (ver Figuras 8, 9 y 10): 1) a todas las filas/documentos, 2) a un 20% de las filas/documentos y 3) a 10 de las filas/documentos.

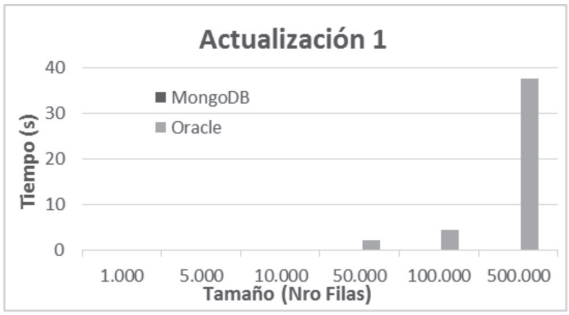

Figura 8. Resultados de las pruebas de actualización: caso 1

En la Figura 8, los resultados favorecieron de nuevo a MongoDB. De hecho, los tiempos para los seis tamaños de muestra fueron

prácticamente los mismos. El mayor tiempo registrado por Oracle posiblemente se debe a la misma razón expuesta para los resultados de la operación de inserción.

En la Figura 9 los resultados en MongoDB fueron similares a los del caso anterior. Aunque el tiempo registrado por Oracle fue mayor en todas las muestras, este mejoró con respecto al caso anterior. Esto se evidenció aún más en el siguiente caso, ver Figura 10, aunque los tiempos de MongoDB siguieron siendo mejores. Esta mejora en Oracle se puede deber al uso de índice sobre la clave primaria, el cual facilita la localización de las filas a actualizar en el segundo y tercer caso, y evita una exploración completa de la tabla (table access full) como se evidencia en los planes de ejecución obtenidos, véase la Figura 11.

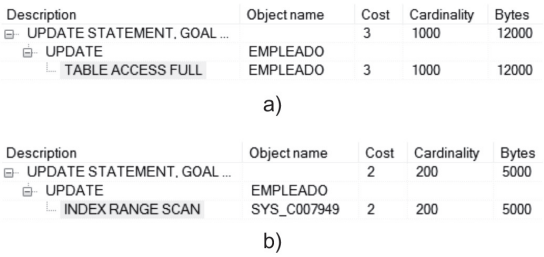

Figura 11. Planes de ejecución en Oracle para la actualización (tabla empleado con 1000 filas): a) de todas las filas y b) del 20% de las filas

Para la operación de consulta se evaluaron tres casos donde la operación recuperaba (ver Figuras 12, 13 y 14): 1) una única fila/documento, 2) 20% de las filas/documentos, y 3) 10 de las filas/documentos.

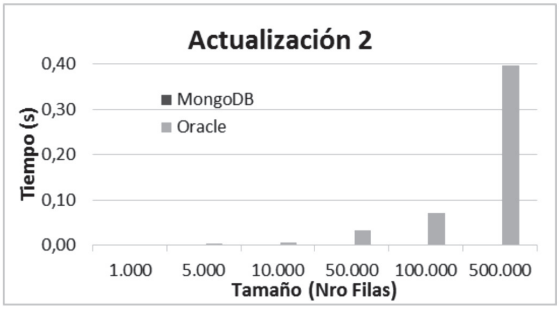

Figura 9. Resultados de las pruebas de actualización: caso 2

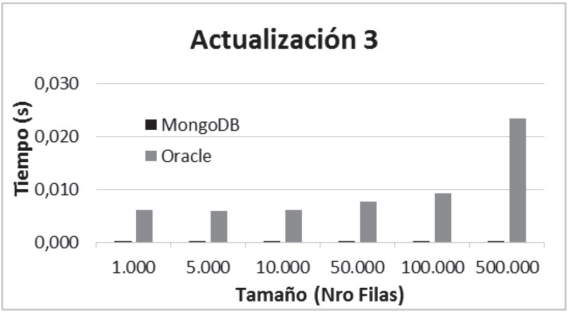

Figura 10. Resultados de las pruebas de actualización: caso 3

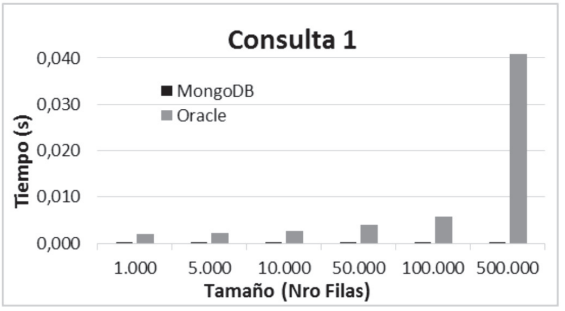

Figura 12. Resultados de las pruebas de consulta: caso 1

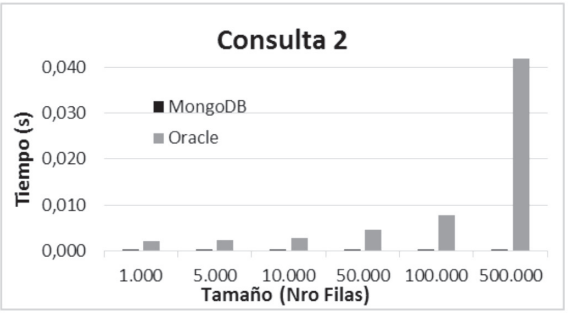

Figura 13. Resultados de las pruebas de consulta: caso 2

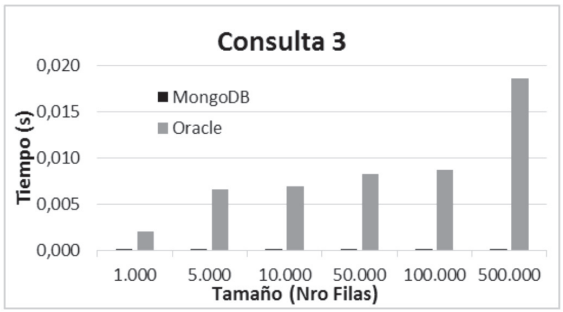

Figura 14. Resultados de las pruebas de consulta: caso 3

En los tres casos de las consultas los resultados se mantuvieron prácticamente iguales en cada SGBD. De hecho, todas las consultas requirieron *menos de un segundo*. Aunque los resultados favorecieron a MongoDB, las diferencias fueron menores en comparación con la operación de inserción y con el caso 1 de actualización.

Para la operación de borrado solo se muestran los resultados para muestras de datos mayores a 50.000 filas/documentos borrados en su totalidad (véase la Figura 15), ya que para tamaños inferiores los tiempos de ejecución en ambos SGBD fueron, como en el caso de las consultas, menores a un segundo.

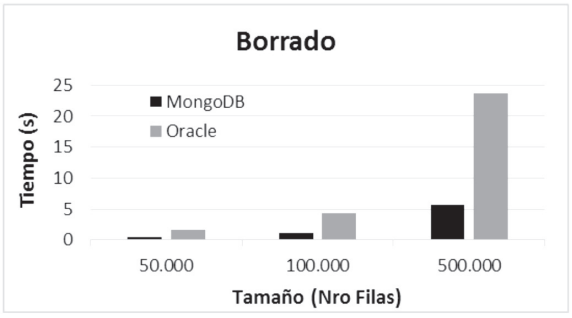

Figura 15. Resultados de las pruebas de borrado

Los resultados favorecieron a MongoDB. El mayor tiempo registrado por Oracle se puede deber a la misma razón expuesta para los resultados de la operación de inserción, pero adicionalmente porque en Oracle la operación de borrado (DELETE) ejecuta procesos adicionales, *e.g.*, puede incluir copia de los datos borrados (papelera de reciclaje del SGBD).

Para la reunión se consideraron las tablas/colecciones empleado y dpto. El tamaño de la tabla dpto en cada prueba fue a la décima parte de la tabla empleado, *i.e.*, por cada departamento hay diez empleados.

En la Figura 16 se puede observar que los resultados favorecieron esta vez a Oracle. MongoDB se ve posiblemente afectado por los costos implicados para la ejecución de la reunión por medio de MapReduce, ya que MongoDB no posee un operador propio (nativo) para ejecutar dicha operación.

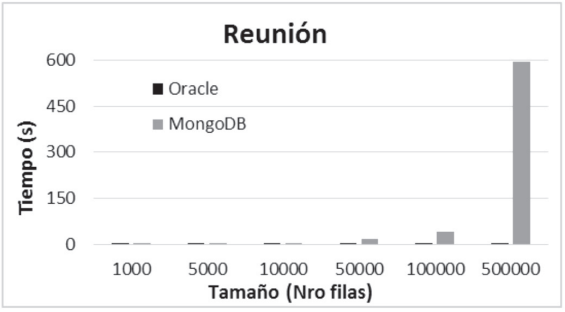

Figura 16. Resultados de las pruebas de la reunión

Adicionalmente, con el fin de aprovechar el paralelismo de MongoDB se probó la reunión en una minired de cuatro y ocho equipos (cada equipo con la configuración descrita al inicio de esta sección). En cada equipo se alojó un fragmento de cada tabla (*e.g.*, una cuarta parte de cada tabla en cada uno de los cuatro equipos). Para garantizar en lo posible igualdad

de condiciones se usó esta vez la opción paralela de Oracle (Oracle 12c Release 1). Los resultados se muestran en las Figuras 17 y 18.

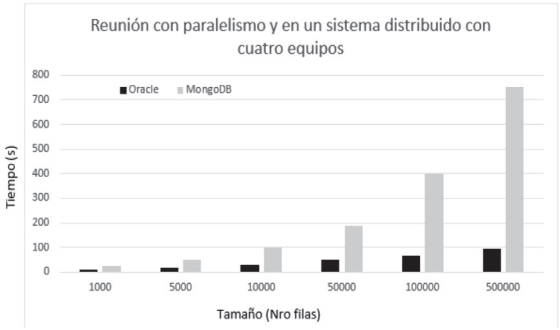

Figura 17. Resultados de las pruebas de la reunión (cuatro equipos)

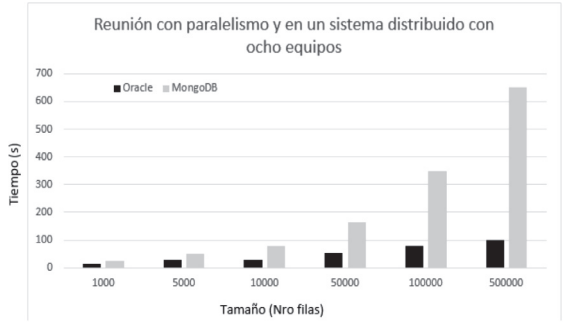

Figura 18. Resultados de las pruebas de la reunión (ocho equipos)

En un experimento más complejo se analizó el rendimiento de dos reuniones. Para ello se consideró el modelo de la Figura. 19.

Figura 19. Modelo Entidad-Relación para los experimentos con dos reuniones

Sea la consulta "*imprimir los datos de cada ciudad, junto con los datos de su estado y país*". Si se considera una relación para cada entidad, para solucionar esta consulta se requiere una reunión entre Ciudad y Estado y luego una reunión con País. En los experimentos se generaron datos aleatorios: para cada país se generaron 10 estados y para cada estado 10 ciudades. Los tamaños de las muestras fueron 500, 1.000, 5.000 y 10.000 países (*i.e.*, para este último tamaño se tuvieron 100 mil estados y un millón de ciudades). Se intentó también con una muestra de 100 mil países, pero MongoDB generó un error, ya que *el tamaño máximo por documento es de 16 MB* (no soportó la inserción de 10 millones de ciudades). Esta es una restricción que se debe considerar también cuando se va a elegir un SGBD. Los experimentos se ejecutaron en un solo equipo y también en la red de cuatro y ocho equipos análogamente a la reunión entre Empleado y Dpto (en cada equipo se alojó un fragmento de cada relación; se validó, además, que en un mismo equipo quedase un país con todos sus estados y ciudades). Similarmente, en MongoDB se crearon las tres colecciones y se hizo primero la reunión entre las colecciones de Ciudad y Estado (análogamente a la reunión entre Empleado y Dpto) y luego la reunión con la colección País. Los resultados se muestran en la Figura 20.

En un siguiente experimento en MongoDB, se consideraron solo dos colecciones, una para países y en la otra se agregó a cada estado un arreglo de ciudades. Se ejecutó entonces la reunión entre las dos colecciones. Los resultados se muestran en la Figura 21. En Oracle se mantuvieron las tres relaciones, por ello en la Figura 21 solo se muestran los resultados de MongoDB. Aunque los resultados de MongoDB mejoraron con respecto al experimento anterior (Figura 20), debido a que

solo se requirió una reunión, su rendimiento fue inferior a Oracle (Figura 20).

En un último experimento, en MongoDB se consideró una sola colección de países, a cada país se le incluyó un arreglo de estados y a cada estado un arreglo de ciudades. Los resultados se muestran en la Figura 22. Esta vez MongoDB obtuvo resultados mejores (y en particular cuando se usó un solo equipo) que Oracle (ver Figura 20), ya que los datos estaban en una misma colección (no se requirió ninguna reunión), en términos prácticos estaban *pre-reunidos*. Este es un patrón de diseño usual que se emplea en el diseño de BD de documentos tal y como se sugiere en [14] y [19] para las relaciones uno a muchos (*one-to-many*). Este mismo experimento se replicó en Oracle, *i.e.*, se colocó en una sola relación cada país con sus departamentos y con sus municipios (*i.e.*, la relación solo estaba en primera forma normal) y los resultados fueron en términos prácticos iguales que en MongoDB, lo mismo que sucedió con las consultas de las Figuras 12, 13 y 14.

A partir de las Figuras 20, 21 y 22 se concluye que el rendimiento de MongoDB se ve afectado por las reuniones y por los costos de comunicación (nótese que cuando se usó la minired con ocho equipos y las tres colecciones los resultados fueron los peores). Por otro lado, cuando se trabajó con una sola colección (Figura 22) los resultados de MongoDB fueron un poco mejores que los de Oracle, en particular

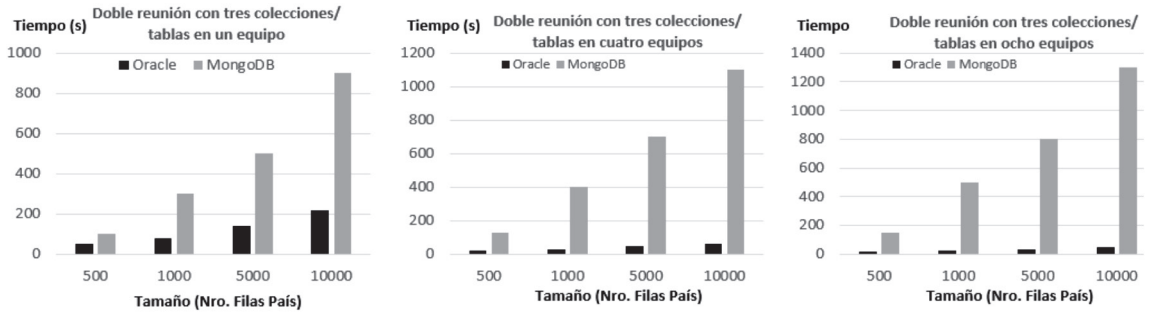

Figura 20. Resultados de las pruebas con dos reuniones (con tres colecciones)

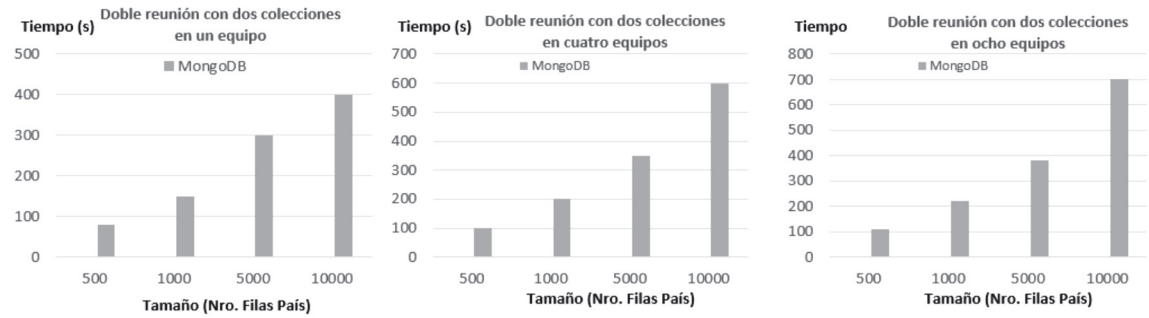

Figura 21. Resultados de las pruebas una reunión (con dos colecciones)

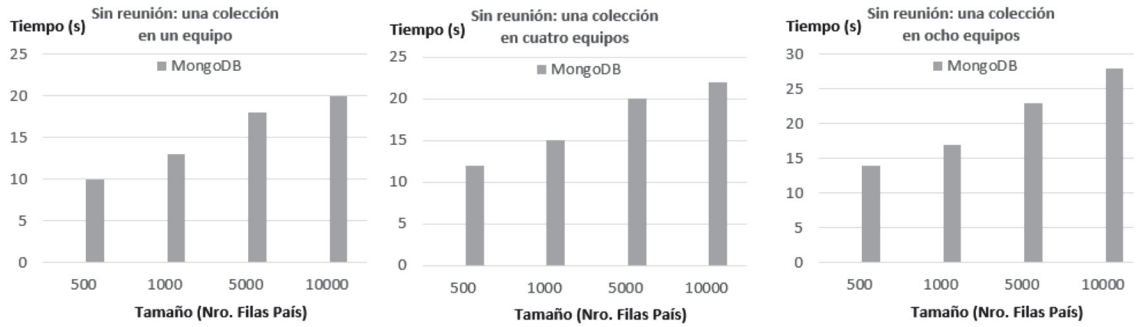

Figura 22. Resultados de las pruebas con todos los datos en una colección

cuando se usó *un solo equipo*: esto es razonable, ya que no se requirieron reuniones (todos los datos estaban en una misma colección) y no había costos de comunicación. Sin embargo, si se trabaja igualmente en Oracle con una sola relación que aloja todos los datos, se llega a resultados similares.

En general, los resultados sugieren que MongoDB es más rápido que Oracle para las operaciones de inserción, actualización y borrado de datos. Para las consultas que involucran una *sola tabla* los resultados son en términos prácticos iguales y para las consultas que involucran reuniones Oracle fue más rápido, incluso cuando se consideró la minired. Estos resultados fueron similares a los obtenidos por Pavlo [20], donde se compara, en un ambiente de paralelismo, el rendimiento de uno de los principales SGBD relacional (no se especifica cuál) contra Vertica y Hadoop (ver más detalles en la siguiente sección).

5. TRABAJOS RELACIONADOS

En [21] se muestra a través de una serie de comparaciones, entre el SGBD MySQL y algunos SGBD NoSQL, que MySQL en inserciones por lotes de entre 100.000 y 70 millones de registros y en las consultas fue el más veloz. Por ejemplo, al comparar con MongoDB se obtuvo una diferencia de factor 100 en las inserciones y las consultas. En [22] también se comparan los tiempos de inserción y de consulta de MongoDB y MySQL y los resultados favorecen a MongoDB. Se afirma que aunque los SGBD relacionales aún son necesarios, no se acomodan a las necesidades de las aplicaciones actuales que requieren manipular grandes volúmenes de datos (*Big Data*). Las diferencias de los resultados en estos trabajos posiblemente se debe a la forma de inserción de los registros, ya que en [10] se insertan registros "simples" (sin claves foráneas ni documentos anidados) mientras que en [22] se insertan registros más complejos (en varias tablas/colecciones, con claves foráneas en MySQL y documentos anidados en MongoDB) lo que al parecer favorece a MongoDB.

En [23] se hace una serie de comparaciones entre MongoDB y PostgreSQL, entre estas inserción por lotes y consultas geoespaciales. Se concluye que MongoDB se debe usar en aquellas aplicaciones en las que las

operaciones sean principalmente de inserción, actualización y borrado, en tanto que si se requieren principalmente operaciones de consulta se recomienda PostgreSQL.

Orend [24] compara y propone una clasificación para diferentes SGBD NoSQL. Se resalta la variedad de este tipo de SGBD, y como cada uno de estos es apropiado para determinadas aplicaciones: es tarea del desarrollador determinar cuál es el más apropiado según sus necesidades.

Arora y Aggarwal [25] discuten sobre la incapacidad de las BD relacionales para adaptarse a las aplicaciones actuales debido a su esquema predeterminado. A raíz de esto sugieren migrar los datos de un SGBD relacional a uno NoSQL mediante un algoritmo que transforma los datos relacionales en documentos JSON.

En [26] se compara el rendimiento entre algunos SGBD NoSQL y SGBD relacionales. Se destaca el rendimiento de las operaciones de inserción, actualización, borrado y consulta en MongoDB y Couchbase, SGBD NoSQL que obtuvieron los mejores resultados.

En [27] se presentan los antecedentes de las BD NoSQL y las limitaciones de las BD relacionales. Se analizan las ventajas y desventajas de los SGBD NoSQL y se propone una clasificación que guíe al usuario en la elección del SGBD NoSQL más apropiado según sus necesidades.

En [28] se discute el término NoSQL y se clasifican las BD que abarcan este término. Analizan cuatro SGBD: Cassandra, HBase, Sherpa y MySQL y hacen una serie de pruebas de rendimiento, donde Cassandra obtiene los mejores resultados. Concluyen que el uso de un producto depende de la necesidad o problema y que ninguno reemplaza al otro totalmente.

En [29] se compara el rendimiento de MongoDB y MySQL para las operaciones CRUD. Los autores concluyen que a) para conjuntos de datos de diez filas y dos columnas, MySQL tuvo un mejor desempeño para las consultas, b) para conjuntos de datos de dos mil filas y veinte columnas, las diferencias fueron irrelevantes, c) el tiempo de inserción fue menor en todas las pruebas en MongoDB, y d) para consultas que involucraron múltiples reuniones, MongoDB ejecutó mejor que MySQL.

Pavlo [20] analiza el rendimiento de Hadoop (uno de los sistemas más populares que soporta MapReduce), Vertica (un SGBD que soporta paralelismo y grandes bodegas de datos) y uno de los principales SGBD relacional también con soporte para paralelismo (no se especifica cuál, en dicho artículo se denomina DBMS-X). Los resultados y conclusiones concuerdan en general con los obtenidos en la sección 4. Pavlo concluye que: "*En general, el SGBD relacional fue significativamente más rápido y requirió menos código, pero su afinamiento y la carga de datos requirió más tiempo*". La reunión (join) en Vertica y en DBMS-X fue en promedio 50 veces más rápida que en Hadoop.

6. CONCLUSIONES Y TRABAJOS FUTUROS

En este artículo se comparó el rendimiento entre dos SGBD, uno relacional (Oracle en su versión Express 11g) y otro NoSQL (MongoDB) orientado a colecciones de documentos JSON. Aunque se requieren experimentos más exhaustivos y muchos otros tipos de

pruebas, los tiempos registrados para las operaciones de inserción, actualización, borrado y consultas sobre una sola relación/colección favorecieron a MongoDB. Esto se debe posiblemente al mayor número de verificaciones de consistencia y de integridad, y a la gestión de transacciones realizadas por Oracle. Sin embargo, para operaciones de consulta binarias, *i.e.*, que involucran dos relaciones/colecciones como la reunión, el tiempo de respuesta favoreció a Oracle. Esto se debe posiblemente a los costos implicados para la ejecución de la reunión por medio de la programación adicional requerida (MapReduce) y *los costos de comunicación entre los nodos (equipos) de la red.*

Debido a que el rendimiento de la programación en paralelo, como en el caso de MapReduce, depende del número de procesadores y de los costos de comunicación entre estos, un trabajo futuro es determinar si con un mayor número de procesadores los resultados de la operación de reunión siguen favoreciendo a Oracle. Otro trabajo es comparar el rendimiento de la operación reunión mediante MapReduce en diferentes SGBD NoSQL y el de otras operaciones binarias como la unión y la intersección.

Apéndice

MapReduce en MongoDB para la reunión entre una colección de empleados y de departamentos.

Función Map:

```
function(){
if (this.id != null) //Si tiene id, es EMP
   emit (this.dep, {id: this._id, tabla: 'Emp'});
   else // Si no tiene id, es DEP
   emit (this._id, {nom: this.nom, tabla: 'Depto'});
}
```

Función Reduce:

```
function (key, values){
   var departamento = '';
   for (var i = 0; i < values.length; i++){
   //Se recorre cada uno de los valores buscando el nombre del departamento
      value = values[i];
      if (value != null){ //Se encontró
         if (value.nom != null){
            departamento = value.nom;
            break;
         }
      }
}
values.forEach(function(value){
//Se recorren de nuevo los valores
   var toPrint = '';
   if(value != null){
   //Se genera la cadena con los datos del empleado junto con el nombre de departamento
```

```
    if(value.id != null){
      toPrint += 'Depto: ' + key + ' idEmp: ' + value.id + ' idEmp: ' + value.id
                  + ' nombreDepto: ' + departamento;
    }
   }
});
}
```

BIBLIOGRAFÍA

[1] NoSQL (2009). *Your Ultimate Guide to the Non-Relational Universe!* En: http://nosql-database.org.

[2] Strozzi C., (2015). *NoSQL Relational Database Management System: Home Page.* En: http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page. (22 de septiembre de 2015).

[3] Duda, J. (2012). Business intelligence and NoSQL databases. *Information Systems in Management*, 1(1), pp. 25-37.

[4] Joyanes, L. (2014). *Big data: análisis de grandes volúmenes de datos en organizaciones*. Barcelona: Marcombo Ediciones Técnicas.

[5] Moreno, A., Redondo, T. (2016). Text Analytics: the convergence of Big Data and Artificial Intelligence. En *International Journal of Interactive Multimedia and Artificial Inteligence*, 3(6).

[6] Cheng, Z., Caverlee, J., Lee, K. & Sui, D. Z. (2011). Exploring Millions of Footprints in Location Sharing Services. En *Fifth International AAAI Conference on Weblogs and Social Media*, Barcelona, España, pp. 81-88.

[7] Elmasri, R., Navathe, S. B. (2014). *Fundamentals of database systems*. Boston: Pearson/Addison Wesley.

[8] Date, C. J. (2012). What First Normal Form Really Means. *Date on Database: Writings 2000-2006*. Apress.

[9] Scofield, B. (2009). NoSQL: Death to Relational Databases. *Online RubyConference*. En: http://es.slideshare.net/bscofield/nosql-death-to-relational-databases

[10] Heller, M. (2007). REST and CRUD: the impedance mismatch. *InfoWorld-Developer World*, IDG News Service.

[11] Bassil, Y. (2012). A comparative study on the performance of the Top DBMS systems. *Journal of Computer Science & Research*, 1(1), pp. 20-31.

[12] Asay M. (2014). MongoDB, Cassandra, and HBase the three NoSQL databases to watch. *InfoWorld-Developer World*, IDG News Service.

[13] Shukla, D., Deepak, D., Rakesh A., Pandey, K. & Agrawal, K. K. (2011). An Efficient Approach of Block Nested Loop Algorithm based on Rate of Block Transfer. En *Int. Journal of Comp. Applications*, 21(3), pp. 24-30.

[14] Sullivan, D. (2015). *NoSQL for Mere Mortals*. Ann Arbor, MI, Addison-Wesley Professional, 1sr edition.

[15] MongoDB CRUD (2015). Introduction - MongoDB Manual 3.0.1. En: http://docs.mongodb.org/manual/core/crud-introduction. (22 de septiembre de 2015).

[16] Dean, J., Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1).

[17] Leskovec, J., Rajaraman, A., Ullman, J. D. (2012). *Mining of massive datasets*. New York, Cambridge: Cambridge University Press.

[18] Niemiec, R. (2012). *Oracle Database 11g Release 2 Performance Tuning Tips & Techniques*, McGraw-Hill.

[19] Copelan, R. (2013). M*ongoDB Applied Design Patterns*. Sebastopol, CA, O'Reilly Media, 1ra edition.

[20] Pavlo, A., Paulson, E., Rasin, A., Abadi, D.J., DeWitt, S., Madden, S., Stonebraker, M. (2009). A Comparison of Approaches to Large-Scale Data Analysis. En *ACM SIGMOD International Conference on Management of data Providence*, USA.

[21] Lith, A., Mattsson, J. (2010). *Investigating storage solutions for large data*. (Tesis de maestría). Chalmers University of Technology.

[22] Khan, S., Mane, V. (2013). SQL Support over MongoDB using Metadata. *International Journal of Scientific and Research Publications*, 3(10).

[23] Suter, R. (2012). *MongoDB An introduction and performance analysis*. Informe Técnico, HSR Hochschule für Technik Rapperswil, Universidad de Ciencias Aplicadas de Rapperswil. En: http://wiki.hsr.ch/Datenbanken/files/MongoDB.pdf. (22 de septiembre de 2015).

[24] Orend, K. (2010). *Analysis and Classification of NoSQL Databases and Evaluation of their Ability to Replace an Object-relational Persistence Layer*. (Tesis de maestría). Technische Universitåt München.

[25] Arora, R., Aggarwal, R. R. (2013). An Algorithm for Transformation of Data from MySQL to NoSQL (MongoDB). *International Journal of Advanced Studies in Computer Science and Engineering*, 2(1).

[26] Li, Y., Manoharan, S. (2013). A performance comparison of SQL and NoSQL databases. *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM),* Victoria, Canada, pp. 15-19.

[27] Han, J., Haihong, E., Guan, L., Du, J. (2011). Survey on NoSQL database. En *6th IEEE international conference on Pervasive computing and applications*, pp. 363-366.

[28] Tudorica, B. G., Bucur, C. (2011). A comparison between several NoSQL databases with comments and notes. En *10th Roedunet International Conference (RoEduNet)*, Lași, Rumania, pp. 1-5.

[29] Aghi, R., Mehta, S., Chauhan, R., Chaudhary, S. & Bohra, N. (2015). A comprehensive comparison of SQL and

MongoDB A comprehensive comparison of SQL and MongoDB. En *International Journal of Scientific and Research Publications*, 5(2).