Tecnura
ISSN: 0123-921X
tecnura@udistrital.edu.co
Universidad Distrital Francisco José de Caldas
Colombia

Moreno Arboleda, Francisco Javier; Castrillón Charari, Nataly; Taborda Zuluaga, Camilo
Procesamiento en paralelo y distribuido en dos SGBDS: un caso de estudio
Tecnura, vol. 21, núm. 52, abril-junio, 2017, pp. 111-129
Universidad Distrital Francisco José de Caldas
.png, Colombia

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Tecnura

http://revistas.udistrital.edu.co/ojs/index.php/Tecnura/issue/view/819

DOI: http://dx.doi.org/10.14483/udistrital.jour.tecnura.2017.2.a09

Estudio de caso

Procesamiento en paralelo y distribuido en dos SGBDS: un caso de estudio

Parallel and distributed processing in two SGBDS: A case study

Francisco Javier Moreno Arboleda[1], Nataly Castrillón Charari[2], Camilo Taborda Zuluaga[3]

Fecha de recepción: 8 de agosto de 2016 **Fecha de aceptación:** 15 de febrero de 2017

Cómo citar: Moreno A., F.J.; Castrillón C., N. y Taborda Z., C. (2017). Procesamiento en paralelo y distribuido en dos SGBDS: un caso de estudio. *Revista Tecnura, 21*(52), 111-129. doi: 10.14483/udistrital.jour.tecnura.2017.2.a09

Resumen

Contexto: Una de las estrategias para la gestión de grandes volúmenes de datos es la computación distribuida y en paralelo. Entre las herramientas que permiten aplicar estas características se encuentran algunos sistemas de gestión de bases de datos (SGBD), como Oracle, SQL Sever y DB2.

Método: En este artículo se presenta un caso de estudio donde se evalúa el rendimiento de una consulta SQL en dos de estos SGBD. La evaluación se hace mediante diversas formas de distribución de los datos en una red computadores y con diferentes grados de paralelismo.

Resultados: Aunque son necesarias pruebas más exhaustivas y con mayor variedad de consultas, se evidenciaron las diferencias de rendimiento entre los dos SGBD analizados.

Conclusiones: Las diferencias en rendimiento de los dos SGBDs analizados muestran que a la hora de evaluar este aspecto, se deben considerar las particularidades de cada SGBD y el grado de paralelismo de las consultas.

Palabras clave: Bases de datos, paralelismo, computación distribuida.

Abstract

Context: One of the strategies for managing large volumes of data is distributed and parallel computing. Among the tools that allow applying these characteristics are some Data Base Management Systems (DBMS), such as Oracle, DB2, and SQL Server.

Method: In this paper we present a case study where we evaluate the performance of an SQL query in two of these DBMS. The evaluation is done through various forms of data distribution in a computer network with different degrees of parallelism.

Results: The tests of the SQL query evidenced the performance differences between the two DBMS analyzed. However, more thorough testing and a wider variety of queries are needed.

Conclusions: The differences in performance between the two DBMSs analyzed show that when evaluating this aspect, it is necessary to consider the particularities of each DBMS and the degree of parallelism of the queries.

Keywords: Databases, Parallelism, Distributed computing.

1 Ingeniero de sistemas, doctor en Ingeniería de Sistemas. Docente de la Universidad Nacional de Colombia, Sede Medellín. Medellín, Colombia. Contacto: fjmoreno@unal.edu.co

2 Ingeniero de sistemas, doctor en Ingeniería de Sistemas. Docente de la Universidad Nacional de Colombia, Sede Medellín. Medellín, Colombia. Contacto: fjmoreno@unal.edu.co

3 Ingeniero de sistemas e informática, arquitecto desarrollador de Ceiba Software House. Medellín, Colombia. Contacto: ctabordaz@unal.edu.co

INTRODUCCIÓN

En la última década, los volúmenes de datos generados y procesados por las aplicaciones se han incrementado de manera significativa. Por ejemplo, hay aplicaciones que han generado exabytes (un *exabyte* equivale a 10^{18} *bytes*) de datos provenientes de sensores, con cientos de millones de datos espaciales, temporales y de redes sociales (Cheng, Caverlee, Lee y Sui, 2011). El procesamiento distribuido y en paralelo es una alternativa para gestionar estos grandes volúmenes de datos, los cuales pueden provenir de aplicaciones como *chats,* blogs, servicios de correo, redes sociales (datos como tuits), entre otros. En particular Hadoop (http://hadoop.apache.org) es uno de los sistemas más usados para este propósito. Por otro lado, están los sistemas de gestión de bases de datos (SGBD) convencionales. Entre estos se destacan por su posicionamiento en el mercado: Oracle, SQL Server y DB2, los cuales incluyen también opciones para explotar el procesamiento distribuido y en paralelo.

En este artículo, se presenta un caso de estudio donde se analiza el rendimiento de Oracle y de SQL Server considerando estas opciones. Este análisis puede ser valioso para los desarrolladores si se consideran: i) los grandes volúmenes de datos estructurados que se deben gestionar hoy; ii) los pocos estudios comparativos al respecto (de hecho, no se encontró ningún trabajo que compare estos SGBD), incluso muy pocos evalúan el desempeño del paralelismo en un solo SGBD y los pocos que lo hacen (Burleson, 2005; Microsoft, 2015; ORACLE, 2014b, 2014c) no son recientes (se hicieron con versiones antiguas del SGBD) o son publicados por la misma compañía fabricante del SGBD, lo que puede restarle objetividad y generar sesgos, y iii) la poca documentación que existe para el diseño de pruebas de distribución y de paralelismo en SGBD. En cuanto a este aspecto, aunque las pruebas consideradas no son complejas ni exhaustivas, sí ofrecen un punto de partida para la elaboración de un conjunto de pruebas en un ambiente mayor, por ejemplo, en una red (*cluster*) con cientos de procesadores.

El artículo está estructurado así. En Paralelismo: conceptos esenciales, se presentan los conceptos esenciales del paralelismo. En Elementos para el procesamiento de consultas distribuidas y en paralelo en Oracle, se presentan algunos conceptos particulares de Oracle para llevar a cabo consultas distribuidas y en paralelo. En Metodología y experimentos, se presenta el caso de estudio y en Resultados los resultados de los experimentos. En Trabajos relacionados, se exponen trabajos relacionados y en Conclusiones, se concluye el artículo y se plantean trabajos futuros.

Paralelismo: conceptos esenciales

El paralelismo permite que varios procesadores cooperen para la ejecución de un proceso. Usualmente, uno de los procesadores, denominado *coordinador* o *maestro*, distribuye las operaciones y recibe los resultados de los otros procesadores, denominados *esclavos* (Deepak, Kumar, Durgesh y Bhupendra, 2012; ORACLE, 2014b). El coordinador también suele ejecutar las operaciones no paralelas del proceso y puede actuar también como esclavo (Oracle, 2014b).

En Silberschatz, Korth y Sudarsham (2011) el paralelismo se clasifica en paralelismo de *intraoperación* y de *interoperación*. El primero permite que una operación de un proceso se ejecute de forma simultánea mediante *n* procesadores donde cada uno suele operar sobre un conjunto *diferente* de datos; *n* se denomina *grado de paralelismo* (Oracle, 2014b) (en inglés se denomina DOP, *degree of parallelism*). Por ejemplo, considérese la siguiente consulta en SQL donde "cédula" es un atributo numérico:

```
SELECT *
FROM empleado
ORDER BY cedula;
```

Supóngase que se tienen cuatro procesadores P_1, P_2, P_3 y P_4 para ejecutar esta consulta

en paralelo. La operación de ordenamiento ("ORDER BY cedula") se puede ejecutar mediante paralelismo de intraoperación (aquí se supone que *no* existe un mecanismo previo de ordenamiento de las filas de la tabla empleado según el atributo cédula, por ejemplo, mediante un índice B+). En este ejemplo, la operación se va a distribuir entre tres procesadores P_1, P_2 y P_3 (el P_4 va a actuar solo como coordinador). El coordinador distribuye entonces los datos entre los tres procesadores. Por ejemplo, véase la figura 1, las cédulas entre 1 y 4 para el procesador P_1, las cédulas entre 5 y 8 para el procesador P_2 y las cédulas entre 9 y 12 para el procesador P_3. Esta división se conoce como *división por rangos* (Sadat y Lecca, 2009). Cada uno de los tres procesadores (P_1, P_2 y P_3) ordena localmente sus datos y los envía al coordinador, el cual los imprime (para iniciar la impresión, el coordinador debe esperar los datos ya ordenados de P_1, luego los de P_2 y finalmente los de P_3).

La tabla "empleado" puede estar en un disco compartido (cada procesador tiene acceso a él); de no ser así, el coordinador debe enviar a cada procesador su correspondiente partición (Akl, 1989).

Por otro lado, el paralelismo de interoperación permite que *diferentes operaciones* de un proceso se puedan ejecutar simultáneamente. El paralelismo interoperación se clasifica a su vez en *canalizado* (*pipelining*) e *independiente* (Silberschatz, Korth y Sudarsham, 2011). El paralelismo canalizado permite que una operación *B* trabaje con los resultados que genera una operación *A*, así las dos operaciones trabajan simultáneamente (Silberschatz, Korth y Sudarsham, 2011). Por ejemplo, sea *A* la operación de reunión (*join* representada con el símbolo ⋈) de dos relaciones *r* y *s*, y sea *B* la operación de impresión de los resultados de la reunión. De esta forma, un procesador se puede encargar de ejecutar la reunión (operación *A*) y a medida que

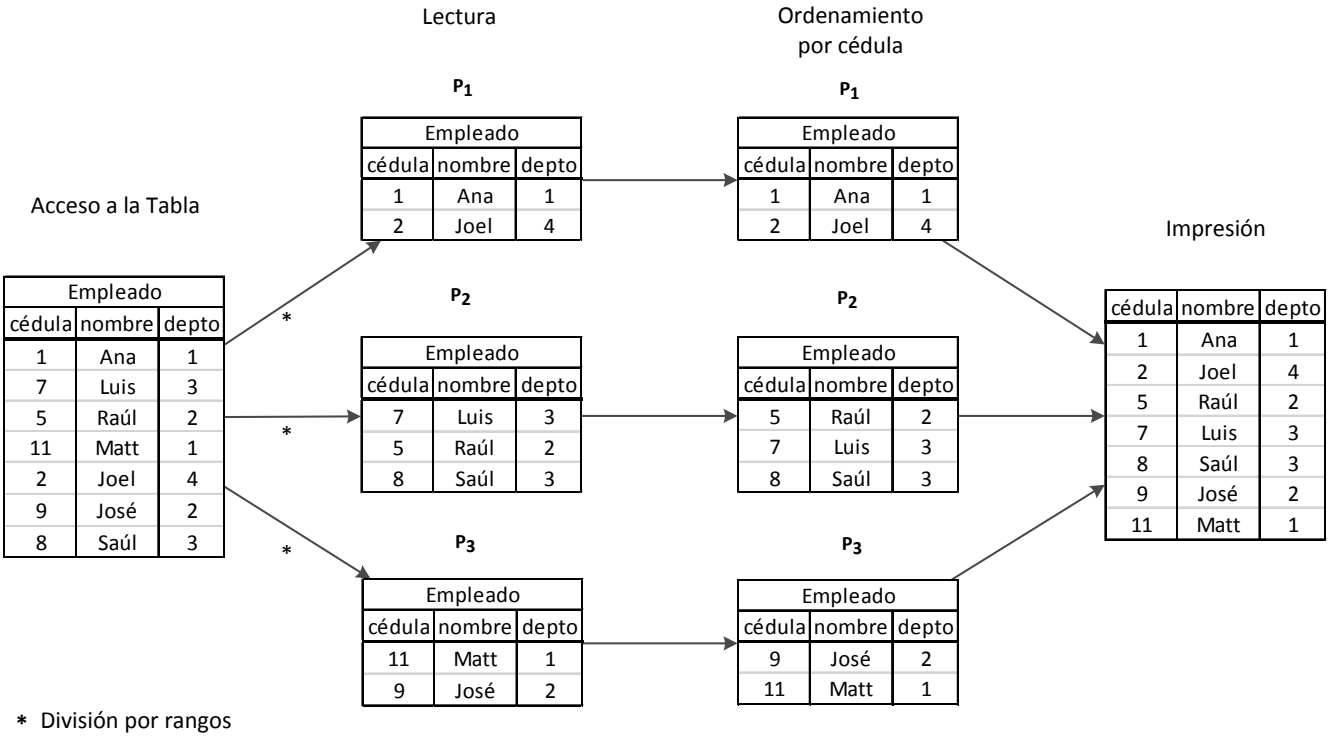


Figura 1. Paralelismo en consulta de ordenamiento

Fuente: elaboración propia

va generando resultados los va enviando al procesador que los imprime (operación *B*).

Por su parte, el paralelismo independiente permite que una operación *A* y una operación *B* se ejecuten simultáneamente, siempre y cuando ninguna de las dos operaciones necesite datos de la otra (Burleson, 2005; Silberschatz et al., 2011). Por ejemplo, considérese la reunión de cuatro relaciones $r \bowtie s \bowtie t \bowtie u$. Aquí, es posible que un procesador ejecute $temp1 \leftarrow r \bowtie s$ (operación *A*) mientras otro procesador ejecuta $temp2 \leftarrow t \bowtie u$ (operación *B*); y el resultado final está dado por $temp1 \bowtie temp2$.

Elementos para el procesamiento de consultas distribuidas y en paralelo en Oracle

A continuación, se explican algunos conceptos necesarios para comprender los experimentos ejecutados.

Database link

En Oracle un *database link* es un elemento de una base de datos (BD) (local) que permite acceder a los datos de otra BD (Burleson-Consulting, 2014). A continuación se crea un *database link* llamado *conexión*:

```
CREATE DATABASE LINK conexion
CONNECT TO username
IDENTIFIED BY password
USING '(DESCRIPTION = (ADDRESS = (PROTOCOL = TCP)
(HOST = 192.168.0.1)(PORT = 1521))
(CONNECT_DATA = (SID = ORCL)))' ;
```

Donde "username" y "password" corresponden al usuario de la BD remota, la cual está ubicada en la dirección IP 192.168.0.1 en el puerto 1521, y su nombre es ORCL. Por ejemplo, para acceder desde la BD local a una tabla departamento ubicada en la BD remota se ejecuta la siguiente consulta:

```
SELECT *
FROM departamento@conexion;
```

Limpieza de memoria cache

Cuando en un SGBD se ejecuta una consulta, los datos pueden quedar alojados en la memoria caché (de esta forma, si la consulta se repite se pueden evitar accesos a disco); estos datos se deben eliminar para evitar sesgos en los resultados de las pruebas ejecutadas. Por ejemplo, en Oracle se usan las siguientes dos sentencias (Burleson-Consulting, 2008; Oracle, s.f.):

```
ALTER SYSTEM FLUSH SHARED_POOL;
ALTER SYSTEM FLUSH BUFFER_CACHE;
```

La primera sentencia elimina planes de ejecución (Burleson-Consulting, 2013) y la segunda limpia la memoria caché de datos (Burleson-Consulting, 2014).

Hint

En Oracle, un *hint* permite modificar el plan de ejecución de una consulta SQL (Antognini, 2013; Oracle, 2014a). Los *hints* se especifican luego de la cláusula SELECT así:

```
SELECT /*+ hints */ *
FROM …;
```

Para las consultas de los experimentos se usó el *hint PARALLEL*. Este permite especificar el DOP de una consulta:

```
/*+ PARALLEL(DOP) */
```

Si se omite el valor del DOP, por ejemplo, si solo se especifica */*+ *PARALLEL* */*, Oracle lo establece automáticamente (Harrison, 2000; Oracle, 2014c). Si el DOP es mayor al número de procesadores físicos disponibles entonces el optimizador puede limitarlo o puede generar procesadores virtuales (por ejemplo, un mismo procesador físico puede generar varios procesadores virtuales). En los experimentos, se ofrecen más detalles.

METODOLOGÍA

Las pruebas se ejecutaron en una red inalámbrica con tres computadores, conectados mediante un enrutador. En un computador, denominado central, se crearon dos *database links*, uno (com1) hacia el computador 1, y otro (com2) hacia el computador 2; análogamente en los computadores 1 y 2 se crearon los respectivos *database links* (central1 y central2) hacia el computador central.

Las especificaciones de los tres computadores fueron las siguientes. Cada uno tenía procesador de dos núcleos (dos físicos y dos lógicos, Intel Core i5-4200U @ 1.60-2.30GHz), memoria RAM de 8 GB, sistema operativo Windows 8.1 – 64 bits y SGBD Oracle Database 12c Enterprise Edition. Las especificaciones del enrutador fueron: marca D-Link, tipo DIR-300, velocidad de transferencia de datos 54Mbps y *switch* de cuatro puertos.

Tablas

Las tablas base para los experimentos fueron "empleado" y "departamento".

CREATE TABLE departamento(codigo **NUMBER**(10) **PRIMARY KEY**,
nombred **VARCHAR**(10) **NOT NULL**);

CREATE TABLE empleado(cedula **NUMBER**(8) **PRIMARY KEY**,
nombre **VARCHAR**(8) **NOT NULL**,
depto **NUMBER**(10) **NOT NULL**,
salario **NUMBER**(10) **NOT NULL**);

La tabla "departamento" se creó solo en el computador central y la tabla "empleado" se creó en los *tres* computadores. Sin embargo, un empleado solo se almacena en uno de los tres computadores; por ejemplo, la intersección de las cédulas entre las tres tablas de empleados es vacía (tabla empleado particionada) (figura 2).

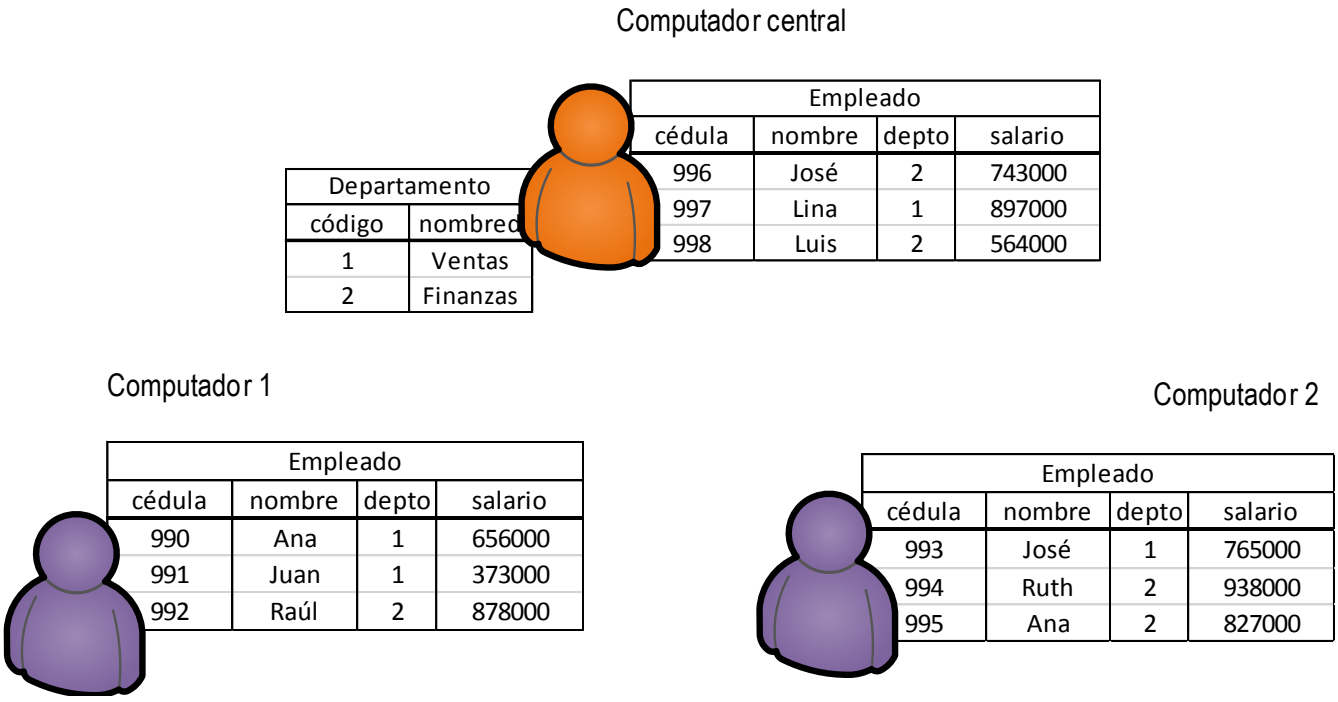


Figura 2. Distribución de datos

Fuente: elaboración propia.

Nótese que las tablas empleado y departamento están relacionadas por medio de los atributos "código" (de departamento) y "depto" (de empleado). No obstante, esta última columna "depto" (de empleado) *no se definió como clave foránea* con respecto a la columna "código de departamento" *debido a que la tabla departamento es remota* con respecto a las tablas de empleados de los computadores 1 y 2.

Consultas

Las consultas de los experimentos *se ejecutaron desde el computador central* y requirieron datos de las tablas de empleados de los computadores 1 y 2. Estos datos pueden ser enviados al computador central para que este ejecute todo el proceso de la consulta o el proceso puede ser *distribuido* y ejecutado entre los tres computadores, y cada computador remoto retornar los resultados de su procesamiento al computador central. Se consideraron cinco consultaslas cuales *generan los mismos resultados,* pero difieren según la forma en que se ejecuta el proceso, el cual se controla mediante el *hint* PARALLEL y las vistas (explicadas más adelante) (Cervantes Ramírez, 2012). Las consultas imprimen el código, el nombre y el promedio de los salarios de cada departamento. Además, los resultados se ordenan ascendentemente según el código del departamento (columna "depto").

Cada consulta se ejecutó con seis muestras de datos (ver tabla 1). Cada consulta con cada muestra se ejecutó tres veces y se registró el valor promedio de las tres ejecuciones. Por ejemplo, en la muestra 1 se tenía una tabla departamento con 30.000 filas y tres tablas de empleados (una en cada computador) cada una con 300.000 filas.

Consulta 1. Procesamiento centralizado sin paralelismo en ningún computador
Considérese la consulta:

```
SELECT depto, nombred, salarioPromedio
FROM (SELECT depto, AVG(salario) AS salarioPromedio
FROM (SELECT * FROM empleado@com1
UNION
SELECT * FROM empleado@com2
UNION
SELECT * FROM empleado)
GROUP BY depto), departamento
WHERE depto = codigo
ORDER BY depto;
```

El computador central (desde el cual se emite la consulta) recibe los datos de las tablas empleado del computador 1 y del computador 2, los une (UNION), ejecuta la subconsulta (agrupamiento por "depto"), luego ejecuta la reunión con su tabla "departamento" y ordena por "depto". Aquí, los computadores 1 y 2 *se limitan* a enviar los datos al computador central donde se ejecutan las operaciones de agrupamiento, reunión y ordenamiento de la consulta.

Tabla 1. Número de filas de las muestras usadas en los experimentos

	Muestra					
	1	**2**	**3**	**4**	**5**	**6**
Departamento (solo en el computador central)	30000	60000	90000	120000	150000	180000
Empleado (en cada computador)	300000	600000	900000	1200000	1500000	1800000

Fuente: elaboración propia.

Consulta 2. Procesamiento centralizado con paralelismo en el computador central

Considérese ahora la consulta:

```
SELECT /*+ PARALLEL(3) */ depto, nombred, AVG(salario)
FROM (SELECT /*+ PARALLEL(3) */ codigo, nombred, e1.*
FROM empleado@com1 e1, departamento
WHERE e1.depto = codigo
UNION
SELECT /*+ PARALLEL(3) */ codigo, nombred, e2.*
FROM empleado@com2 e2, departamento
WHERE e2.depto = codigo
UNION
SELECT /*+ PARALLEL(3) */ código, nombred, ec.*
FROM empleado ec, departamento
WHERE ec.depto = codigo)
GROUP BY depto, nombred
ORDER BY depto;
```

Esta consulta es similar a la anterior, pero con el *hint* PARALLEL. Esto significa que las operaciones de la consulta se ejecutarán en el computador central *pero con paralelismo* en este (en la consulta anterior con DOP = 3). De nuevo, los computadores 1 y 2, se limitan a enviar sus datos al computador central.

Consulta 3. Procesamiento descentralizado, pero sin paralelismo en ningún computador

Se crea una vista en cada uno de los computadores como se explica a continuación. Considérese la vista creada en el computador 1:

```
CREATE OR REPLACE VIEW no_parallel_view_c1 AS
SELECT depto, nombred, salarioPromedio
FROM (SELECT depto, AVG(salario) AS salarioPromedio
FROM empleado
        GROUP BY depto
        ), departamento@central1
        WHERE depto = codigo;
```

Aquí, se agrupan los empleados del computador 1 por el atributo "depto" y se ejecuta la reunión con la tabla departamento *enviada desde el computador central* (por ejemplo, *la reunión se ejecuta en el computador 1*). En forma análoga, se crean también las vistas "no_parallel_view_c2" (en el computador 2) y "no_parallel_view_central" (en el computador central).

Luego las vistas son invocadas desde el computador central mediante la consulta:

```
SELECT *
FROM (SELECT depto, nombred, AVG(salarioPromedio)
FROM (SELECT * FROM no_parallel_view_c1@com1
UNION
SELECT * FROM no_parallel_view_c2@com2
UNION
SELECT * FROM no_parallel_view_central)
GROUP BY depto, nombred)
ORDER BY depto;
```

La diferencia con la consulta 1 es que los datos a unir (UNION) ya están agrupados (por "depto") y reunidos con la tabla departamento. Sin embargo, al igual que en la consulta 1, las operaciones de unión (de los resultados de las vistas enviadas por los computadores 1, 2 y de los resultados de la vista local del computador central), agrupamiento (por "depto" y "nombred") y ordenamiento por "depto" se terminan de ejecutar en el computador central.

Consulta 4. Procesamiento descentralizado y con paralelismo en los computadores 1 y 2

Se crea una vista en cada uno de los computadores como se explica a continuación. Considérese la siguiente vista creada en el computador 1:

```
CREATE OR REPLACE VIEW parallel_view_c1
AS
SELECT /*+ PARALLEL(3) */ depto, nombred, salarioPromedio
```

```
FROM (SELECT /*+ PARALLEL(3) */ depto, AVG(salario) AS salarioPromedio
        FROM empleado
        GROUP BY depto
        ), departamento@central1
        WHERE depto = codigo;
```

Al igual que en la vista de la consulta 3, se agrupan los empleados del computador 1 por el atributo "depto" y se ejecuta la reunión (en el computador 1) con la tabla departamento *enviada desde el computador central*. Sin embargo, la diferencia con la consulta 3 está en el *hint* PARALLEL que fuerza a que la consulta que define la vista se ejecute con paralelismo (con DOP = 3) en el computador 1 (cuando la vista es invocada). De forma análoga ocurre para la vista "parallel_view_c2" del computador 2 y con la vista "parallel_view_central" del computador central.

Luego, las vistas son invocadas desde el computador central, mediante la consulta:

```
SELECT *
FROM (SELECT depto, nombred, AVG (salarioPromedio)
FROM (SELECT * FROM parallel_view_c1@com1
UNION
SELECT * FROM parallel_view_c2@com2
UNION
SELECT * FROM parallel_view_central)
GROUP BY depto, nombred)
ORDER BY depto;
```

Nótese que aunque la formulación de esta consulta es prácticamente igual a la de la consulta 3 (solo cambian los nombres de las vistas), aquí las consultas correspondientes a las vistas de los computadores 1 y 2 son ejecutadas *en paralelo en sus respectivos computadores* y sus resultados (ya agrupados por "depto" y ya reunidos con la tabla "departamento", la cual fue enviada desde el computador central a cada uno de estos dos computadores) son enviados al computador central donde se termina de ejecutar la unión (UNION) de los resultados de las vistas, el agrupamiento final (por "depto" y "nombred") y el ordenamiento por "depto" (*sin* ejecutar paralelismo en el computador central).

Consulta 5. Procesamiento descentralizado con paralelismo en los tres computadores

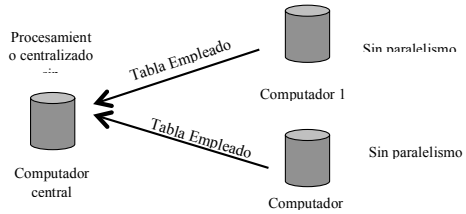
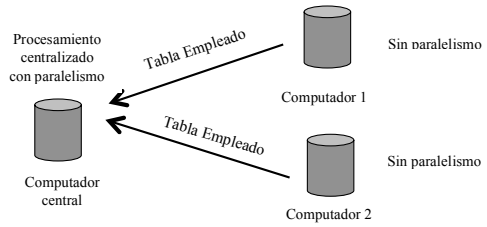
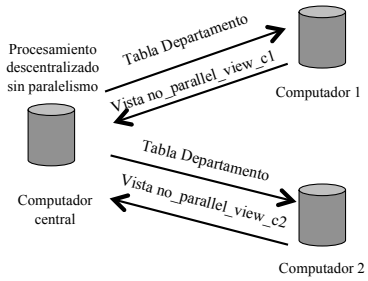
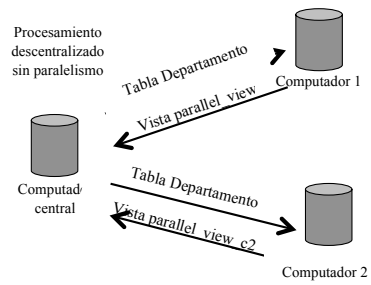
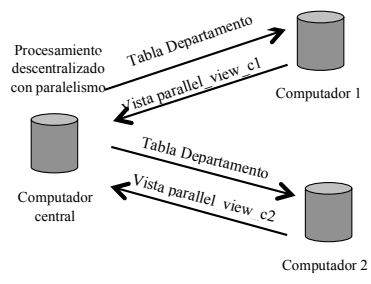
Se usan las mismas vistas creadas para la consulta 4 y se invocan desde el computador central mediante la siguiente consulta:

```
SELECT /*+ PARALLEL(3)*/ *
FROM (SELECT /*+ PARALLEL(3) */ depto, nombred, AVG (salarioPromedio)
FROM (SELECT /*+ PARALLEL(3) */ * FROM parallel_view_c1@com1
UNION
SELECT /*+ PARALLEL(3) */ * FROM parallel_view_c2@com2
UNION
SELECT /*+ PARALLEL(3) */ * FROM parallel_view_central)
GROUP BY depto, nombred)
ORDER BY depto;
```

La diferencia con la consulta 4 es que aquí el computador central termina de ejecutar la unión (UNION) de los resultados de las vistas, el agrupamiento final (por "depto" y "nombred") y el ordenamiento por "depto" *con paralelismo* (paralelismo ejecutado *en el computador central*), luego de recibir los datos de los computadores 1 y 2 ya agrupados por "depto" y ya reunidos con la tabla departamento (*la cual fue enviada a estos dos computadores desde el computador central*). Nótese que en los computadores 1 y 2 también se ejecuta paralelismo *en cada uno* de ellos para generar sus correspondientes resultados, como en la consulta 4.

En la tabla 2 se ilustra cada una de las cinco consultas.

Tabla 2. Flujo de datos y de operaciones de cada consulta

Consulta 1		
Centralizado sin paralelismo: UNION Tablas Empleado GROUP BY depto ⋈ Empleado y Departamento ORDER BY depto		
Consulta 2		
Centralizado con paralelismo (en el computador central): ⋈ de cada Tabla Empleado con Departamento UNION reuniones anteriores GROUP BY depto, nombred ORDER BY depto		
Consulta 3		
Descentralizado sin paralelismo (en cada computador): GROUP BY depto ⋈ con Departamento (computador central) **Sin paralelismo** (en el computador central): UNION Vistas no_parallel_view GROUP BY depto, nombred ORDER BY depto		Vista sin paralelismo GROUP BY depto ⋈ con Departamento (enviada desde el computador central) Vista sin paralelismo GROUP BY depto ⋈ con Departamento (enviada desde el computador central)
Consulta 4		
Descentralizado con paralelismo (en los computadores 1 y 2): GROUP BY depto ⋈ con Departamento (computador central) **Con paralelismo** (en el computador central): UNION Vistas parallel_view GROUP BY depto, nombred ORDER BY depto		Vista con paralelismo GROUP BY depto ⋈ con Departamento (enviada desde el computador central) Vista con paralelismo GROUP BY depto ⋈ con Departamento (enviada desde el computador central)
Consulta 5		
Descentralizado con paralelismo (en los tres computadores): GROUP BY depto ⋈ con Departamento (computador central) **Con paralelismo** (en el computador central): UNION Vistas parallel_view GROUP BY depto, nombred ORDER BY depto		Vista con paralelismo GROUP BY depto ⋈ con Departamento (enviada desde el computador central) Vista con paralelismo GROUP BY depto ⋈ con Departamento (enviada desde el computador central)

Fuente: elaboración propia.

RESULTADOS

En la tabla 3 y en la figura 3 se presentan los resultados de los experimentos.

Tabla 3. Resultados de la ejecución de pruebas

Consulta	Muestra	Tiempo primera ejecución (s)	Tiempo segunda ejecución (s)	Tiempo tercera ejecución (s)	Tiempo promedio ejecuciones (s)
1	1	18,67	17,93	22,05	19,55
	2	31,18	31,75	27,12	30,02
	3	43,63	44,01	43,92	43,85
	4	56,32	54,67	51,89	54,29
	5	83,70	70,82	71,40	75,31
	6	111,43	117,27	114,87	114,52
2	1	17,04	23,21	25,40	21,88
	2	33,44	27,13	32,27	30,95
	3	52,38	46,22	47,62	48,74
	4	56,93	59,60	60,10	58,88
	5	64,87	62,49	67,29	64,88
	6	68,59	69,39	68,71	68,90
3	1	16,71	20,08	18,00	18,26
	2	27,42	29,14	29,53	28,70
	3	34,65	41,62	38,04	38,10
	4	51,55	53,69	50,21	51,82
	5	58,64	65,69	62,28	62,20
	6	75,64	82,78	88,65	82,36
4	1	20,89	20,65	17,54	19,69
	2	28,69	28,35	29,71	28,92
	3	46,08	41,55	42,01	43,21
	4	46,39	45,92	49,61	47,31
	5	58,81	53,16	56,34	56,10
	6	64,69	66,23	67,29	66,07
5	1	17,78	16,29	17,69	17,25
	2	30,02	29,60	29,24	29,62
	3	34,42	37,28	44,89	38,86
	4	50,48	52,27	44,03	48,93
	5	59,53	71,08	66,11	65,57
	6	109,33	89,15	98,81	99,10

Fuente: elaboración propia.

Para el menor tamaño de muestra, el tiempo de ejecución de las cinco consultas fue prácticamente el mismo. A partir de la muestra 5, las consultas 2 y 4 obtuvieron un menor tiempo de ejecución. Con la muestra mayor, la 6, las consultas 1 y 5 registraron los tiempos de ejecución más altos, esto sugiere que el costo del paralelismo puede ser igual o más alto que el procesamiento secuencial; de hecho, hay operaciones que no convienen ser paralelizadas o que su ejecución secuencial es difícil de superar (Cervantes Ramírez, 2012). Los costos de coordinación y de comunicación, los volúmenes de datos y el número de procesos en paralelo son decisivos a la hora de elegir una alternativa.

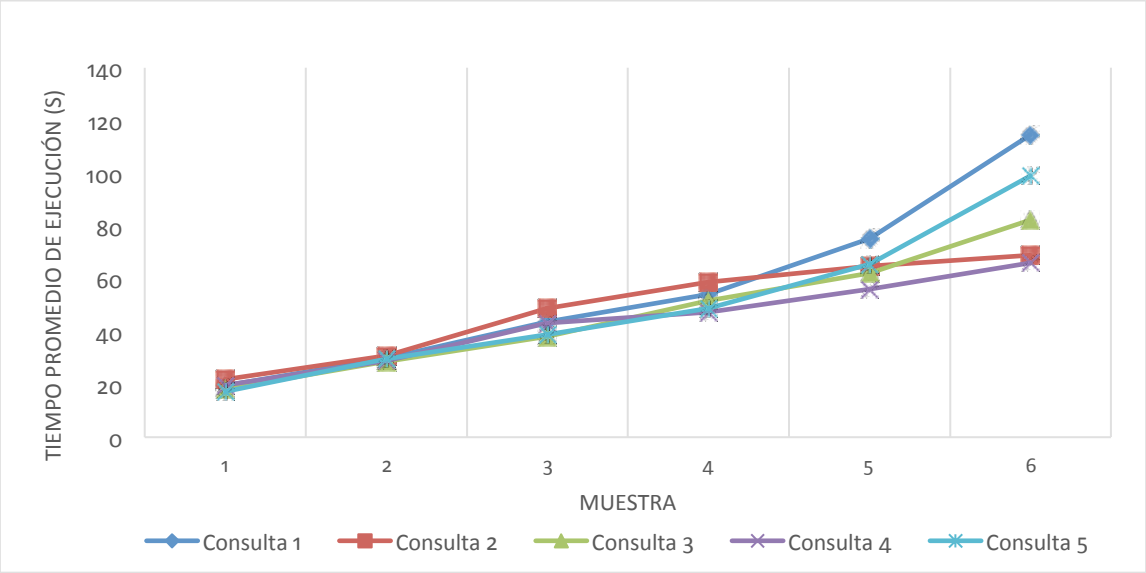


Figura 3. Comparación del tiempo promedio de ejecución de las cinco consultas

Fuente: elaboración propia.

Experimentos con variación en el DOP

Para la ejecución de consultas con paralelismo, se recomienda usar como DOP el número total de procesadores que posee el computador, tanto los físicos como los lógicos (Oracle, 2014b). Otras fuentes recomiendan que el DOP debe corresponder al número total de procesadores – 1, con el fin de dejar un procesador como coordinador de la consulta (Burleson-Consulting, 2012). Un DOP menor o mayor al recomendado puede afectar negativamente el desempeño de la consulta. Por ejemplo, si se ejecuta una consulta, con un DOP recomendado igual a 4 y luego con un DOP igual a 200, el rendimiento entre las dos consultas será muy distinto: la generación y coordinación de los 200 procesos de las segunda consulta afectarán negativamente su tiempo de respuesta. Por ello, el DOP óptimo se suele encontrar mediante pruebas para cada *consulta específica* (Baskan y Oracle, 2015; Burleson-Consulting, 2012). A continuación, se muestran los resultados de los experimentos con tres DOP.

Como se expuso en Metodología y experimentos, cada computador usado en los experimentos posee cuatro procesadores (dos físicos y dos lógicos). Se usaron los siguientes DOP: 3, 6 y 9. Los resultados para DOP = 3 se presentaron en la tabla 3. Los resultados con DOP = 6 y con DOP = 9 se muestran en las tablas 4 y 5. En las figuras 4, 5 y 6 se comparan los resultados para las consultas con paralelismo (consultas 2, 4 y 5).

Tabla 4. Resultados de la ejecución de pruebas don DOP = 6

Consulta	Muestra	Tiempo primera ejecución (s)	Tiempo segunda ejecución (s)	Tiempo tercera ejecución (s)	Tiempo promedio ejecuciones (s)
2	1	14,01	13,71	12,04	13,25
	2	22,50	21,46	21,50	21,82
	3	35,68	31,92	33,73	33,78
	4	43,46	45,60	42,01	43,69
	5	52,53	52,21	52,23	52,32
	6	62,21	62,39	65,84	63,48
4	1	24,97	21,38	20,12	22,16
	2	32,84	32,63	36,17	33,88
	3	36,96	38,69	36,28	37,31
	4	49,69	44,45	45,91	46,68
	5	55,79	59,42	55,89	57,03
	6	68,46	67,20	71,69	69,12
5	1	19,40	16,85	17,88	18,04
	2	26,73	28,25	27,85	27,61
	3	44,75	37,98	39,66	40,80
	4	46,11	48,84	50,03	48,33
	5	65,74	59,44	58,62	61,27
	6	74,35	69,29	76,49	73,38

Fuente: elaboración propia.

Tabla 5. Resultados de la ejecución de pruebas con DOP = 9

Consulta	Muestra	Tiempo primera ejecución (s)	Tiempo segunda ejecución (s)	Tiempo tercera ejecución (s)	Tiempo promedio ejecuciones (s)
2	1	12,03	14,95	15,51	14,16
	2	23,37	26,82	24,51	24,90
	3	35,66	33,68	34,23	34,52
	4	42,26	43,84	46,61	44,24
	5	53,21	53,36	53,87	53,48
	6	62,56	60,67	61,47	61,57
4	1	22,60	22,03	23,13	22,59
	2	38,67	35,94	34,22	36,28
	3	47,93	46,93	48,45	47,77
	4	53,02	57,02	54,40	54,81
	5	72,02	70,32	74,45	72,26
	6	82,52	84,25	79,36	82,04
5	1	22,38	23,29	23,20	22,96
	2	32,30	34,68	33,61	33,53
	3	47,68	42,83	50,00	46,84
	4	61,28	60,78	62,34	61,47
	5	75,78	69,42	68,67	71,29
	6	81,87	81,34	84,06	82,42

Fuente: elaboración propia.

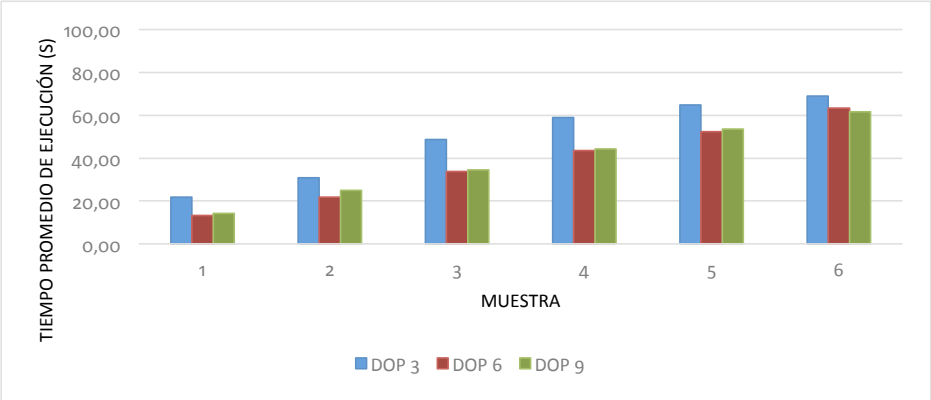


Figura 4. Comparación de los tiempos obtenidos en la consulta 2 para da DOP

Fuente: elaboración propia.

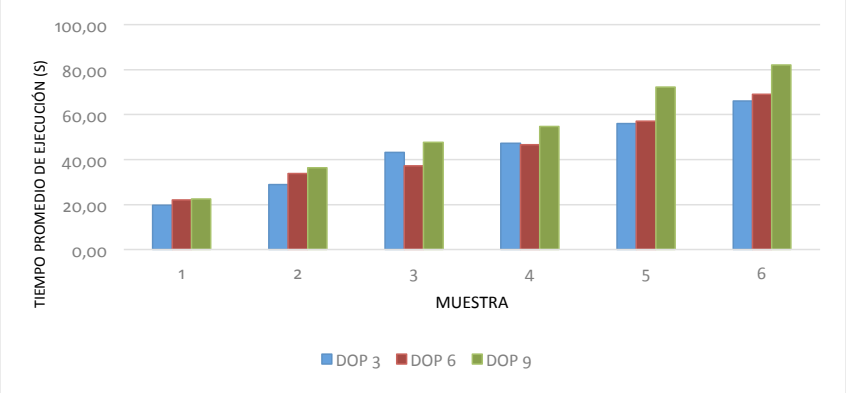


Figura 5. Comparación de los tiempos obtenidos en la consulta 4 para cada DOP

Fuente: elaboración propia.

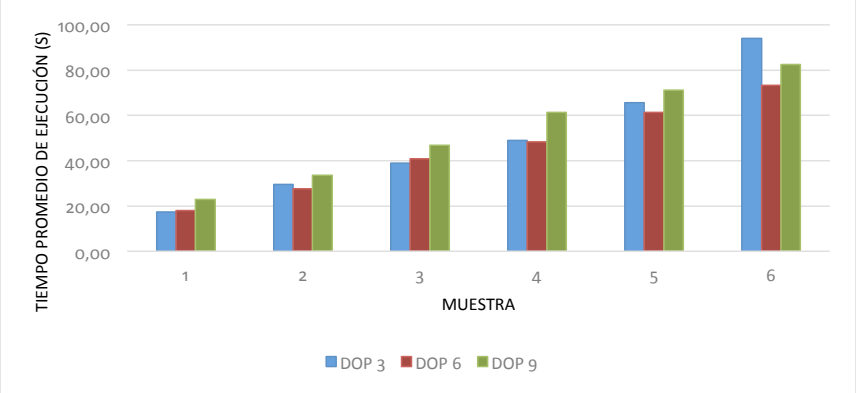


Figura 6. Comparación de los tiempos obtenidos en la consulta 5 para cada DOP

Fuente: elaboración propia.

Como se observa, el DOP óptimo depende de cada consulta. Por ejemplo, para la consulta 4, el DOP que ofreció el mejor desempeño fue el DOP = 3, mientras que para la 5, el mejor fue el DOP = 6. Para la consulta 2, se evidenció que el DOP = 3 es menos eficiente que el DOP = 6 o el DOP = 9. No obstante, el tiempo de ejecución para las consultas con DOP = 6 o DOP = 9 suele crecer más rápido que el tiempo de ejecución para las consultas con DOP = 3, a medida que aumentan los tamaños de muestra. Así, para tamaños de muestra más grandes, el DOP 3 fue más eficiente.

Oracle vs SQL Server

Con el fin de comparar los tiempos de respuesta que ofrecen Oracle 12c y SQL Server 2014 se hicieron los siguientes experimentos. Se compararon estos dos SGBD porque, además de soportar paralelismo, son dos de los de mayor posicionamiento en el mercado (Solid-IT y DB-Engines, 2015).

Las condiciones de los experimentos siguen las descritas en Metodología y experimentos. Se ejecutaron dos consultas:

- La primera consulta corresponde a la consulta 1 presentada en el aparte Consultas y con los datos (departamentos y empleados) distribuidos como allí se explica.
- En la segunda consulta, todos los empleados y todos los departamentos se encuentran en el computador central; por tanto, *los computadores 1 y 2 no participan de la consulta*. Además, se usó el *hint* PARALLEL en Oracle (y MAXDOP en forma correspondiente en SQL Server), por ejemplo, la consulta se ejecuta con paralelismo en el computador central. A esta consulta se le denominará consulta 6 y se esquematiza en la figura 7.

La razón por la que se usan estas consultas para las pruebas de los dos SGBD es que el procesamiento en paralelo y distribuido simultáneamente tiene limitaciones en SQL Server, al menos en la versión trabajada (2014).

Consulta sobre datos remotos sin paralelismo
Considérese la consulta 1 en SQL Server:

```
SELECT depto, nombre, salarioPromedio
FROM (
SELECT depto, AVG(salario) AS salarioPromedio
FROM ( SELECT * FROM [com1].master.dbo.
empleado
UNION
SELECT * FROM [com2].master.dbo.empleado
UNION
SELECT * FROM dbo.empleado
) AS t0
```

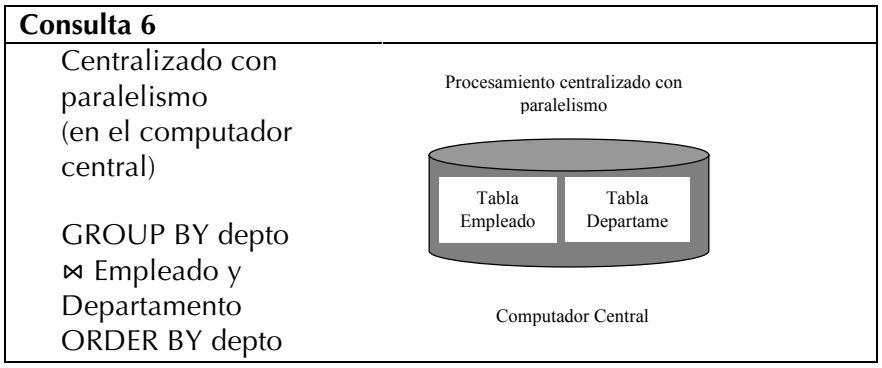


Figura 7. Flujo de datos y de operaciones de la consulta 6

Fuente: elaboración propia.

```
GROUP BY depto
) AS t1, dbo.departamento
WHERE depto = codigo
ORDER BY depto;
GO
```

Y considérese la misma consulta en Oracle (la consulta 1 presentada en el aparte Consultas). En la figura 8 se presentan los resultados.

Consulta sobre datos locales con paralelismo

Considérese ahora la consulta 6 en SQL Server:

```
SELECT depto, nombre, salarioPromedio
FROM (
SELECT depto, AVG(salario) AS salarioPromedio
FROM dbo.empleado
GROUP BY depto
) AS t1, dbo.departamento
WHERE depto = codigo
ORDER BY depto
OPTION (MAXDOP 4)
```

Y considérese la misma consulta en Oracle (en la consulta anterior se quita OPTION (MAXDOP 4) y se adiciona al SELECT principal el *hint* /*+ PARALLEL(4)). En la figura 9 se presentan los resultados.

Como se explicó en el aparte Experimentos con variación en el DOP, algunos autores recomiendan usar como DOP el número total de procesadores del sistema. En particular, en SQL Server si el MAXDOP es mayor al número de procesadores del computador, *este lo ignora* y solo usa el número de procesadores disponibles (Microsoft, 2015).

En el experimento, el computador central tenía cuatro procesadores; por consiguiente, el DOP que siempre eligió SQL Server fue 4. En ese sentido, para la comparación con las consultas paralelas en Oracle también se usó un DOP = 4. De otra parte, en SQL Server es posible establecer un umbral para el paralelismo: a cada consulta SQL Server le asigna un costo, si este supera el umbral establecido, la consulta se hace de manera paralela. Si se establece este umbral en el valor mínimo (1), se garantiza la ejecución en paralelo para todas las consultas.

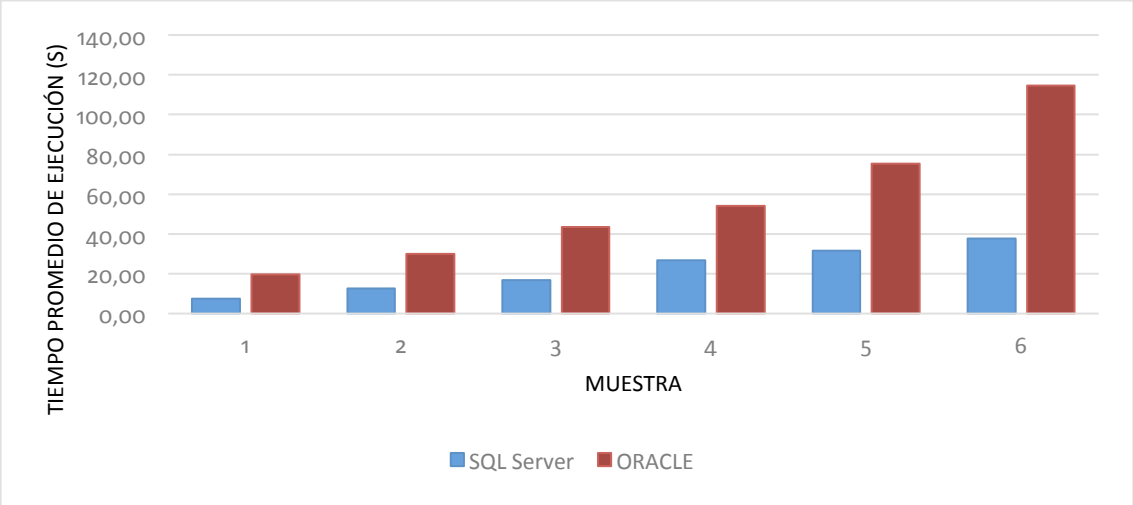


Figura 8. Comparación de los tiempos de la consulta 1 para cada SGBD

Fuente: elaboración propia.

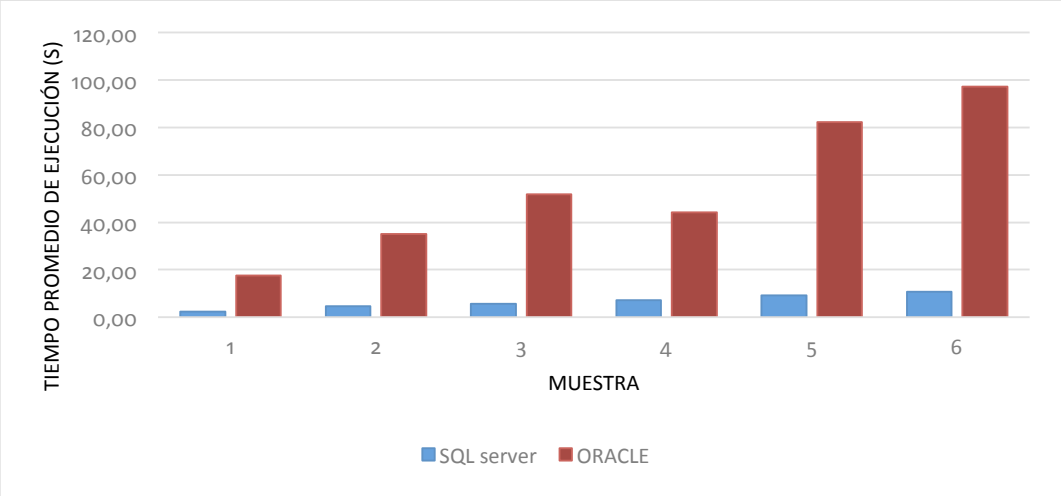


Figura 9. Comparación de los tiempos obtenidos en la consulta 6 para cada SGBD

Fuente: elaboración propia.

Como se aprecia en las figuras 8 y 9, SQL Server ofreció menores tiempos de respuesta para las dos consultas. Lo anterior puede estar relacionado con el sistema operativo Windows, ya que este y SQL Server son de la misma compañía. De hecho, algunos autores informan de mayores tardanzas en la ejecución de acciones de Oracle sobre Windows (Burleson-Consulting, 2009; Hussain, Farooq, Shamsudeen y Yu, 2013) que sobre otros sistemas operativos. Adicionalmente, De Carvalho y Mattos (2009) argumentan que, en casos donde el número de usuarios es pequeño o el sistema operativo es Windows, se obtiene un mejor desempeño con SQL Server que con Oracle o con PostgreSQL. Considerando lo anterior, y a partir de los experimentos con diferentes DOP en Oracle, se observó que SQL Server ofreció menores tiempos de ejecución que Oracle *sin importar* el DOP elegido en Oracle.

TRABAJOS RELACIONADOS

En los últimos años se han hecho varios estudios sobre el desempeño de las consultas con paralelismo en SGBD. En Cervantes (2012) se desarrolla una investigación sobre cómo aplicar *GPU-computing* (*graphics processing unit*) para mejorar el tiempo de respuesta de las consultas en paralelo. Basándose en el plan de ejecución del SGBD SQLite (http://www.sqlite.org), las operaciones de dicho SGBD son programadas en la plataforma CUDA (NVIDIA, 2012; Hernández, Gaviria y Montenegro, 2014) donde son procesadas en una GPU. Así, se obtiene un motor de ejecución capaz de solucionar un subconjunto de operaciones del lenguaje SQL en paralelo (en particular, consultas unitabla, multitabla y con funciones de agregación). Por último, los autores concluyen que los tiempos de ejecución y procesamientos en la GPU son mucho menores que en la CPU. El motor desarrollado plantea una alternativa de desarrollo para SGBD con procesamiento en GPU.

Hong y Stonebraker (1993) presentan un enfoque para la optimización de consultas en XPRS (*eXtended Postgres on Raid and Sprite*), un SGBD paralelo multiusuario basado en una memoria compartida con varios procesadores. Los autores presentan una estrategia de optimización en dos fases: la primera es una optimización convencional que encuentre el plan óptimo secuencial y la

segunda encuentra la mejor "paralelización" del plan secuencial anterior. Además incluyen evidencia experimental a través de una serie de pruebas de rendimiento que les permite validar su estrategia de optimización.

Akal, Böhm y Schek (2002) se enfocan en las consultas OLAP (*online analytical processing*). Los autores hicieron numerosos experimentos para evaluar el desempeño de varias consultas. Como resultado, se llega al desarrollo de un optimizador en dos fases: en una primera, se divide una consulta en subconsultas las cuales son enviadas a cada nodo participante. Luego, en la segunda, cada nodo optimiza y evalúa su subconsulta. Como parte de sus resultados, los investigadores proveen una clasificación para las consultas OLAP, la cual es usada para decidir cuándo y cómo las consultas deberían ser paralelizadas.

Miranda, Lima, Valduriez y Mattoso (2006) proponen el motor Apuama, el cual añade paralelismo para mejorar el rendimiento de las consultas OLAP en C-JDBC (*Clustered JDBC: c-jdbc.ow2.org*). Presentan pruebas de rendimiento para consultas OLAP sobre PostgreSQL, donde se demuestra que se mejora el proceso de lectura y el rendimiento de las operaciones de actualización para dichas consultas, con una mejora en el desempeño de manera lineal y en ciertos casos, de manera superlineal.

Lima, Furtado, Valduriez y Mattoso (2009) presentan una propuesta para mejorar el desempeño de las aplicaciones OLAP en un DBC (*database cluster*). Se propone una estrategia de balanceo de cargas que toma ventaja de un particionamiento virtual adaptativo para redistribuir la carga. La validación experimental se hizo sobre SmaQSS (*smashing queries while shrinking disk space*). El DBC muestra que la estrategia es eficiente y obtiene unos resultados superlineales.

Garofalakis y Ioannidis (1997) analizan los problemas que suelen presentar las consultas en un entorno paralelo donde se comparte, entre otros, la memoria, el disco, el costo de comunicación remota entre los procesadores involucrados. Los autores revelan la complejidad detrás de la programación distribuida para consultas paralelas. Se proponen algunos algoritmos heurísticos, algunos de ellos casi óptimos.

Finalmente, Stonebraker *et al*. (2010) exponen la importancia de combinar los SGBD paralelos y el paradigma de programación en paralelo MapReduce en la resolución de problemas analíticos complejos, argumentando que los DBMS paralelos son apropiados para el procesamiento eficiente de grandes volúmenes de datos, mientras que MapReduce es apropiado para las tareas analíticas complejas y de ETL (*extract, transform and load*). El resultado es un sistema más eficiente que combina lo mejor de ambos mundos.

CONCLUSIONES

En este artículo se presentó un caso de estudio donde se evaluó el desempeño de una consulta en dos SGBD. La consulta que incluía entre sus operaciones unión, reunión (*join*), agrupamiento y ordenamiento, se ejecutó en un ambiente distribuido y con diferentes grados de paralelismo; se evaluaron también diferentes formas de distribución del trabajo entre los computadores de la red (*cluster*). Aunque el caso de estudio es simple y se deben evaluar consultas más complejas (por ejemplo, una consulta que involucre varias reuniones), los resultados evidenciaron las diferencias de rendimiento entre los dos productos y cómo se afecta este según el grado del paralelismo y del número de procesadores usado. Incluso, algunos experimentos mostraron que en ocasiones el procesamiento secuencial puede ser más eficiente que un procesamiento en paralelo y distribuido. De hecho, a pesar de que existen modelos de costos (Rajaraman, Leskovecm y Ullman, 2013; Silberschatz et al., 2011) (que consideran factores como los costos de comunicación y de procesamiento en cada nodo de la red) que indican para una consulta u operación en particular a partir de que volumen de datos el procesamiento distribuido y en paralelo puede ser beneficioso, el rendimiento tan distinto de los dos SGBD analizados muestra que

existen otros factores que deben ser considerados (por ejemplo, el acople entre un producto (SGBD) y el sistema operativo parece ser decisivo así como las particularidades de cada SGBD) a la hora de evaluar el rendimiento.

Como trabajo futuro se planea evaluar la consulta en una red de más computadores y con muestras de datos mayores. También se planea comparar el rendimiento de los SGBD frente a sistemas como Hadoop y MongoDB, donde la programación distribuida y en paralelo se ejecuta mediante el paradigma de programación MapReduce. Otra línea de investigación es considerar la interacción del paralelismo con la computación ubicua (Sánchez, 2015).

REFERENCIAS

Akal, F.; Böhm, K. y Schek, H.J. (2002). OLAP Query Evaluation in a Database Cluster: A performance Study on Intra-Query Parallelism. En: Y. Manolopoulos y P. Návrat (eds.). *Advances in Databases and Information Systems* (pp. 218–231). Zúrich: Springer Berlin Heidelberg.

Akl, S.G. (1989). *The Design and Analysis of Parallel Algorithms*. Ontario, Canada: Prentice-Hall International Editions.

Antognini, C. (2013). SQL Optimization Techniques. En: *Oracle Database 12c Performance Tuning Recipes*. Nueva York: Apress.

Baskan, Y. y Oracle (2015). *Finding the Reason for DOP Downgrades*. Recuperado el 20 de enero de 2016 de: https://blogs.oracle.com/datawarehousing/entry/finding_the_reason_for_dop

Burleson Consulting (2008). *Oracle: flush the data buffer cache*. Recuperado el 11 de febrero de 2015 de: http://www.dba-oracle.com/t_opq_parallel_query.htm

Burleson-Consulting (2009). *The real costs of SQL Server vs. Oracle*. Recuperado el 20 de enero de 2016 de: http://www.dba-oracle.com/t_cost_sql_server_vs_oracle.htm

Burleson-Consulting (2012). *Oracle parallel hint tips*. Recuperado el 20 de enero de 2016 de: https://blogs.oracle.com/datawarehousing/entry/finding_the_reason_for_dop

Burleson-Consulting. (2013). *Flush shared pool tips*. Recuperado el 11 de febrero de 2015 de: http://www.dba-oracle.com/10g_flush_shared_pool.htm

Burleson-Consulting. (2014). *Oracle SQL performance with database links–db link*. Recuperado el 12 de febrero de 2015 de: http://www.dba-oracle.com/t_sql_dblink_performance.htm

Burleson, D. (2005). *Inside Oracle Parallel Query*. Recuperado el 12 de febrero de 2015 de: http://www.dba-oracle.com/t_opq_parallel_query.htm

Cervantes R., A.O. (2012). Paralelización de un subconjunto de consultas SQL con unión natural utilizando una GPU. Tesis de maestría, Instituto Politécnico Nacional, Mexico D.F.

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

De Carvalho S., R. y Mattos M., L.A. (2009). *Estudo Comparativo dos Sistemas Gerenciadores de Bancos de Dados: Oracle, SQL Server e PostgreSQL*. Informe técnico. Universidade Presidente Antônio Carlos, Barbacena.

Deepak, S.; Kumar, S.U.; Durgesh, M. y Bhupendra, P. (2012). Query Processing and Optimization of Parallel Database System in Multi Processor Environments. In *Sixth Asia Modelling Symposium* (pp. 191-194). Bali: IEEE. http://doi.org/10.1109/AMS.2012.49

Garofalakis, M. y Ioannidis, Y. (1997). Parallel Query Scheduling and Optimization with Time–and Space–Shared Resources. *Proceedings of the 23rd VLDB Conference Athens, Greece*, 1, 1–10.

Harrison, G. (2000). *Oracle Sql High Performance Tuning*. 2a. ed. Boston: Prentice Hall.

Hernández, E.; Gaviria, G. de J.M. y Montenegro, C.E. (2014). Parallel programming languages on heterogeneous architectures using OPENMPC, OMPSS, OPENACC an *Revista Tecnura, 18* (Edición especial doctorado), 160-170.

OPENMP. Hong, W. y Stonebraker, M. (1993). Optimization of Parallel Query Execution Plans in XPRS. *Distributed and Parallel Databases*, 1 (1), 9-32.

Hussain, S.J.; Farooq, T.; Shamsudeen, R. y Yu, K. (2013). *Expert Oracle RAC 12c*. Nueva York: Apress.

Lima, A.; Furtado, C.; Valduriez, P. y Mattoso, M. (2009). Parallel OLAP query processing in database clusters with data replication. *Distributed and Parallel Databases*, 25 (1), 97-123.

Microsoft (2015). *Recommendations and guidelines for the "max degree of parallelism" configuration option in SQL Server*. Recuperado el 20 de enero de 2016 de: https://support.microsoft.com/en-us/kb/2806535

Miranda, B.; Lima, A.A.B.; Valduriez, P. y Mattoso, M. (2006). Apuama: Combining Intra-query and Inter-query Parallelism in a Database Cluster. En: T. Grust *et al.* (eds.). *Current Trends in Database Technology – EDBT 2006* (pp. 649–661). Río de Janeiro: Springer Berlin Heidelberg.

NVIDIA. (2012). *What is CUDA?* Recuperado el 10 de enero de 2016 de: http://www.nvidia.com/object/cuda_home_new.html

Oracle (s.f.). *Alter System*. Recuperado el 11 de febrero de 2015 de: https://docs.oracle.com/database/121/SQLRF/statements_2017.htm#SQLRF00902

Oracle (2014a). *About Optimizer Hints*. Recuperado el 11 de febrero de 2015 de: http://docs.oracle.com/database/121/TGSQL/tgsql_influence.htm#TGSQL255

Oracle (2014b). *Parallel Execution with Oracle Database 12c Fundamentals*. Recuperado de: http://www.oracle.com/technetwork/articles/datawarehouse/twp-parallel-execution-fundamentals-133639.pdf

Oracle (2014c). *Parallel_clause*. Recuperado el 11 de febrero de 2015 de: http://docs.oracle.com/database/121/SQLRF/clauses006.htm#SQLRF20024

Rajaraman, A.; Leskovecm, J. y Ullman, J.D. (2013). *Mining of Massive Datasets*. Londres: Cambridge University Press.

Sadat, A.B.M.R.I. y Lecca, P. (2009). On the Performances in Simulation of Parallel Databases: An Overview on the Most Recent Techniques for Query Optimization. En: *International Workshop on High Performance Computational Systems Biology, 2009. HIBI '09* (pp. 113-117). Trento: IEEE. http://doi.org/10.1109/HiBi.2009.25

Sánchez M., C.A. (2015). La computación ubicua: omnipresencia en los sistemas de información. *Revista Tecnura*, 19, 121-128.

Silberschatz, A.; Korth, H.F. y Sudarsham, S. (2011). Parallel Databases. En: *Database System Concepts* (pp. 797–814). Nueva York: McGraw-Hill.

Silva B., L. (2003). *Algoritmos heurísticos*. Valparaíso, Chile. Recuperado de: http://www2.elo.utfsm.cl/~lsb/pascal/clases/cap25.pdf

Solid-IT y DB-Engines (2015). *DB-Engines Ranking*. Recuperado el 15 de agosto de 2015 de: http://db-engines.com/en/ranking

Stonebraker, M.; Abadi, D.; Dewitt, D.; Madden, S.; Paulson, E.; Pavlo, A. y Rasin, A. (2010). MapReduce and Parallel DBMSs: Friends or Foes? *Communications of the ACM, 53*(1), 66-71.