



Cuando las bases de datos se quedan cortas

BIGTABLE

Internet está cambiando la manera de entender la informática, y Google es su mayor impulsor. En este número veremos qué se cuece dentro de los gigantes de la informática. **POR JOSÉ MARÍA RUÍZ**

Siempre se han escuchado rumores sobre instalaciones informáticas descomunales de empresas como Google, Yahoo!, Hotmail, Amazon o Microsoft. Se hablaba de sistemas informáticos (muchos de ellos basados en Linux y en Software Libre) que podían rondar perfectamente los 50.000 servidores. Lo increíble es que en casos como el de Google el número probablemente se acerque a ¡¡ 500.000 servidores!!

¿Para qué tantos servidores? Y lo más importante ¿cómo demonios consiguen sacar partido a semejantes infraestructuras?

El santo grial de la arquitectura de sistemas informáticos ha sido siempre conseguir que al introducir nuevos recursos (discos, memoria, procesadores, o servidores completos) aumente proporcionalmente la potencia del sistema. No sólo eso, sino que además el sistema sea tolerante a fallos en cascada, pudiendo recuperarse de forma autónoma de un desastre.

Y lo interesante es que al parecer estamos viviendo una época en la que estos sistemas al fin están comenzando a ser

creados y usados. Las grandes empresas de Internet ya ofertan servicios que serían inimaginables sin estos sistemas. El término que se ha acuñado, Cloud Computing (computación en nube), está empezando a sonar con gran fuerza como el paradigma de desarrollo y funcionamiento con el que conviviremos durante la próxima década.

Por ejemplo, existe un proyecto libre llamado Hadoop que trata de implementar procesos distribuidos empleando ordenadores comunes. Está diseñado usando los conceptos de los sistemas MapReduce y GFS de Google.

Como ahora es posible ejecutar Hadoop sobre el sistema EC2 (Elastic Compute Cloud) de Amazon empleando el sistema de almacenamiento S3 (Simple Storage Service), también de Amazon, el periódico New York Times ha contratado 100 instancias de Amazon EC2 en las que ha instalado Hadoop y ha creado un sistema que le permite procesar 4 Terabytes de imágenes TIFF almacenadas en Amazon S3 y generar 1 millón de PDFs a partir de ellas en 24 horas. ¡Y por sólo 240 dólares!

(sí, Amazon EC2 y Amazon S3 son realmente baratas). Ver referencia [1].

No es sorprendente, por tanto, que los mejores clientes de los sistemas Cloud de Microsoft, Google o Amazon sean grandes farmacéuticas y bancos.

Cambio de Paradigma

Google, Amazon, Yahoo, Microsoft, Facebook, Hotmail, todas estas empresas ofrecen servicios a través de Internet y necesitan hacerlo a una cantidad enorme de clientes. Los sistemas tradicionales de servidores se quedan cortos, y de hecho no son recomendables.

El primer problema es la fiabilidad. Hasta ahora hemos considerado a los ordenadores como máquinas perfectas en las que se ejecutaba software imperfecto. Pero a día de hoy la fiabilidad ha tomado otra dimensión. Ahora mismo las empresas de Internet suelen necesitar miles de equipos trabajando al unísono, y con tantos, el fallo de uno de ellos en cualquier momento está prácticamente garantizado. Da igual lo fiables que sean, siempre fallará alguno.

ROW	TIMESTAMP	CLIENTE:NOMBRE	CLIENTE:APODO	CLIENTE:EMAIL
linux-magazine	2008-03-01 12:30	Jose Ruiz	Pepito	pepito@hotmail.com
linux-magazine	2008-03-01 13:50			pepito@gmail.com
linux-magazine	2008-03-05 17:13	José Ruiz	joseruiz	
linux-magazine	2008-03-18 12:07			jose.ruiz@gmail.com

Figura 1: Esquema de GFS.

El segundo problema es el ancho de banda. Un ordenador puede tener un procesador muy potente, pero al final los datos que pasan por él deben salir de una memoria y puede que de un disco duro. Los caminos que los datos siguen hasta el procesador son muchísimo más lentos que éste. Es imposible sacar más ancho de banda de un solo servidor cuando sus buses ya están saturados.

El tercer problema es el consumo eléctrico. Si tu principal activo son los miles de ordenadores que posees, entonces tu principal gasto es la electricidad. Los equipos domésticos consumen menos que los servidores más potentes. Google, por ejemplo, compra equipos cuyas placas base han sido específicamente diseñadas para él entre otras cosas para eliminar componentes inútiles. Incluso ha patentado su propia unidad de alimentación de bajo consumo.

Estos tres problemas llevan a la siguiente conclusión: necesitamos muchos equipos, poco potentes e intercambiables, compuestos por piezas de ordenadores domésticos baratas y fáciles de encontrar, y que consuman poco. ¿Cómo podemos hacer que un sistema con estas características, una auténtica pesadilla para los ingenieros hardware, pueda dar los servicios que se demandan a día de hoy?

Sólo existe una solución: hacer que el software sea el que reine, y desterrar el hardware como solución.

Google File System: La Nube en el Horizonte

La arquitectura de nube es diferente. Consiste en una gran cantidad de equipos con características más parecidas a las de un "Personal Computer", pero que sólo son parte de un sistema que los engloba. Ninguno de ellos es esencial, son indistinguibles, como células en un organismo.

Google fue de los primeros en hacer uso de una infraestructura así. Pero el simple hecho de poner un montón de PCs

juntos no te permite explotar su potencia. Crearon su propia versión modificada de Linux sobre la que diseñaron un sistema de ficheros llamado Google File System (GFS). El objetivo de GFS era poder almacenar información sobre un sistema de ficheros distribuido de forma segura y que soportase una gran carga de trabajo.

GFS es un sistema distribuido que no guarda todos los datos de un fichero en un solo disco duro ni en un solo ordenador, sino que emplea toda una red de ordenadores para hacerlo. GFS divide los datos en trozos, y hace al menos 3 copias de cada trozo. Cada copia irá a un ordenador diferente dentro de la basta red de Google.

Se emplean dos tipos de servidores, los Master y los Chunkservers. Los primeros almacenan la situación física de los trozos, "chunks", que componen los ficheros, así como la jerarquía de ficheros y directorios, lo que llamamos los metadatos. Para no saturar los Masters, los clientes que acceden a ellos cachean, almacenan temporalmente los datos relativos a qué Chunkservers contactar para un determinado fichero, ver Figura 1.

Los Chunkservers almacenan los trozos en sí. Los ficheros en GFS son un poco especiales. Normalmente son ficheros muy grandes, más de 100MB, y hay unos cuantos millones de ellos. No se pueden sobrescribir, sólo es posible añadir (*append*) datos al final del fichero. La operación *append* es atómica, por lo que no es necesario poseer un sistema de bloqueos muy sofisticado. Si un fichero está realizando un *append*, el resto debe esperar. Los ficheros serán leídos completamente o por partes. Existen Chunkservers primarios y secundarios que almacenan las copias de los chunks del primario. Cuando se escribe en un fichero, el cliente solicita la escritura al Chunkserver primario, manda los datos a todas las réplicas y éstas los almacenan a la orden del primario. El primario asigna a los datos a escribir unos números de serie y realiza la escritura. Entonces solicita a los

secundarios que ejecuten la escritura con el número de serie. El objetivo es que en el peor de los casos la escritura se realice en el primario y en algunos de los secundarios.

Para no saturar la red, uno de los objetivos de todo este sistema, los datos se envían a los Chunkservers de forma lineal, es decir, se envían al primero, que conforme los recibe los envía al siguiente, como si fuese una cuerda que va pasando por varios aros.

GFS además es capaz de asignar más recursos a aquellos ficheros que más se emplean, de forma que cuanto más importante es un fichero, más copias de seguridad se harán de sus trozos, lo que protege mejor esa información y además, al haber más copias disponibles, acelera el acceso.

GFS no está integrado en el sistema operativo, sino que es accesible a través de librerías, de forma que puede ser empleado desde distintos sistemas operativos.

Bigtable

Tener un sistema de ficheros masivamente distribuido, fiable y potente está muy bien, pero lo que realmente necesitan las aplicaciones es una base de datos, y Bigtable es esa base de datos.

Google decidió crear Bigtable porque los sistemas de bases de datos tradicionales no le permitían crear sistemas lo suficientemente grandes. Las bases de datos relacionales, como pueden ser MySQL, PostgreSQL, Firebird u Oracle se diseñaron pensando que se ejecutarían en una solo servidor con mucha potencia. Jamás se pensó en la posibilidad de que estuviesen distribuidas en miles de servidores.

Google creó Bigtable para que fuese, sobre todo, una base de datos en la que se almacenaría una cantidad de información enorme, del orden de Petabytes. Para ello, cada tabla está dividida en "tablets" que pueden llegar a ocupar 200 Megabytes. Si superasen ese tamaño serían automáticamente divididas y comprimidas para ser enviadas a más máquinas usando un sistema de compresión propietario de Google.

Bigtable posee muchas peculiaridades. Para comenzar, aunque se parece mucho a una base de datos relacional, rompe con algunas de sus premisas. Por ejemplo, las tablas se dividen en conjuntos de

columnas y éstos en columnas. Es posible añadir columnas a una tabla en cualquier momento y no es posible borrar filas, sólo podemos reemplazarlas por unas nuevas que ocultan las anteriores. Los datos se localizan empleando tres llaves: la fila, la columna y un timestamp (la fecha y la hora). Así, acceder a filas “borradas” consiste en hacer referencias a ellas con un timestamp de la fecha anterior al borrado.

Cada celda almacena una cadena de texto sin tipo, por consideraciones de rendimiento se suelen almacenar los datos de forma binaria, es decir, como volcados de la memoria que los contenían. Como crear columnas es tan sencillo, la columna en sí es un nuevo almacén.

Por desgracia Bigtable es software propietario de Google, y no tenemos posibilidad de emplearlo ni hacer pruebas con él.

Existen varias alternativas libres en desarrollo para poder hacer uso de esta tecnología. La principal es la combinación de Hadoop e Hypertable.

Hadoop

Hadoop, ver Referencia [2], surgió como una necesidad del proyecto libre Nutch de Doug Cutting. Nutch es un proyecto que implementa un buscador web libre. Cutting leyó el artículo publicado por Google donde se explicaba cómo funciona su sistema MapReduce así como GFS. Sorprendido por las técnicas empleadas, y viendo que Nutch necesitaba algo parecido, se puso manos a la obra y creó Hadoop.

Hadoop se compone de dos sistemas, el propio Hadoop, que realiza las tareas de MapReduce, y HDFS, que realiza las tareas de GFS.

MapReduce es el sistema que Google creó para poder procesar cantidades enormes de información usando un enfoque totalmente distribuido. Para ello, la consulta a los datos se divide en dos fases.

En la fase Map se crea una función que asigna a cada dato algún valor especial. En la fase Reduce se recogen listas con los datos y los valores asignados por la función Map y se filtran usando una nueva función, para dejar sólo aquéllos que cumplan cierta restricción. Tanto Map como Reduce pueden ejecutarse en máquinas diferentes, y por tanto podemos hacer uso de una red de servidores.

Para ello se emplea el sistema de ficheros HDFS, que como GFS, almacena los

datos en distintos servidores usando chunks, y volvemos a tener servidores de metadatos y servidores de chunks. Al igual que en GFS, los chunks se replican en distintas máquinas.

Hypertable

El proyecto Hypertable, ver Referencia [3], surgió como un desarrollo dentro de la empresa Zvents, pero a día de hoy es software libre. Hypertable descansa sobre Hadoop y está programado en C++.

En lugar del lenguaje SQL disponemos de una versión especial llamada HQL. HQL no dispone de todas las sentencias que posee SQL y además introduce nuevas.

Para comenzar, las filas en Hypertable no se comportan de la misma manera que en SQL. En SQL, normalmente, disponemos una fila que se suele denominar *id*, que es de tipo numérico y que empleamos para realizar enlaces entre tablas. En HQL no existe ese tipo de fila, entre otras razones porque no existe el tipo numérico.

Las columnas están agrupadas por familias. Una columna cualquiera será accesible mediante el nombre “familia:nombre_columna”. En la Figura [2] podemos ver varias columnas que pertenecen a la familia *cliente*:

Como en Bigtable, todos los datos almacenados son cadenas. En cambio existen siempre dos filas implícitas en toda tabla Hypertable: ROW y TIMESTAMP, ver Figura [2].

La primera sería la llave de la tabla, y por tanto debe ser única. Puede ser una url, un número o lo que queramos, siempre que se cumpla esta restricción. TIMESTAMP se genera automáticamente cada vez que realicemos una inserción. Veamos cómo podemos trabajar con ellas:

```
hypertable> create table
Paginas (fecha, "refer-url",
"http-code");
hypertable> insert into Paginas
```

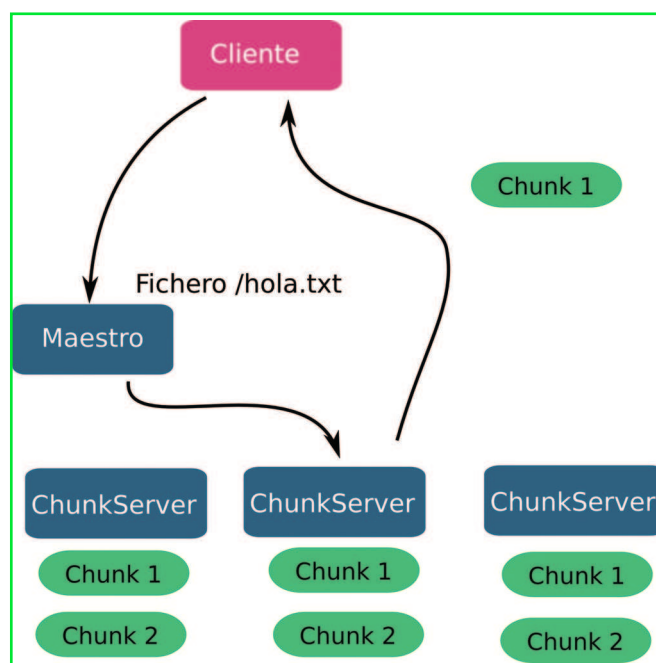


Figura 2: Ejemplo de tabla Hypertable.

```
values ('2008-05-01 23:59:59',
"www.linuxmagazine.es",
"refer-url", "www.google.com");
```

La creación de tablas tiene un poco de truco, tanto la fila ROW como TIMESTAMP son implícitas, por lo que no las creamos. Hemos creado una tabla con tres columnas, pero al contrario que con SQL, no indicamos el tipo de las mismas. Después hemos insertado en la tabla un solo dato ¿Uno solo? Sí, los dos primeros valores que damos son el TIMESTAMP y el ROW, después indicamos un nombre de columna y por último el dato a insertar, en este caso “www.google.com”.

Como dijimos antes, en Hypertable o BigTable se emplean tres valores para indicar una celda, por lo que para insertar un datos necesitaremos esos tres valores de nuevo. Vamos a borrar la fila insertada:

```
hypertable> delete * from
Paginas where ROW =
"www.linux-magazine.es ";
delete: row=
'www.linux-magazine.es'
family=0 ts=0
hypertable> select * from
Paginas where ROW =
"www.linux-magazine.es";
hypertable>
```

Todo funciona como estamos acostumbrados a que funcione, la diferencia apa-

rece cuando ejecutamos el siguiente comando:

```
hypertable> select * from
Paginas where ROW =
"www.linux-magazine.es"
DISPLAY_TIMESTAMPS;
2008-05-01 23:59:59.000000000
www.linux-magazine.org
refer-url www.google.com
```

¿Pero no habíamos borrado ya esa fila de la tabla? No, habíamos borrado toda referencia de la fila en la tabla, pero las filas anteriores, con **TIMESTAMP** anteriores siguen en la tabla. Si queremos eliminarlas habrá que especificar el **TIMESTAMP** que tienen:

```
hypertable> delete * from
Paginas where ROW =
"www.linux-magazine.es"
TIMESTAMP '2008-05-01'
```

```
23:59:59';
delete: row=
'www.linux-magazine.es' family=
0 ts=2146031999000000001
hypertable> select * from
Paginas where ROW =
"www.linux-magazine.es";
hypertable>
```

Si nos fijamos de nuevo en la Figura 2 veremos que aparecen varias filas para el mismo **ROW**, y que en cada una de ellas hay alguna columna cambiada. Hypertable sólo almacena las columnas con datos, mientras que el resto se calcula en base al último valor asignado. Así, el contenido de la última columna en la Figura 2 sería: ("linux-magazine", "2008-03-18 12:07", "José Ruiz", "jose-ruiz", "jose.ruiz@gmail.com")

Al contrario que en SQL, por el momento hay un montón de limitaciones en HQL. Al fin y al cabo es un pro-

yecto libre bastante moderno, pensemos que Hadoop está aún en la versión ¡0.16!

Por desgracia, para los programadores Python como yo, Hypertable aún no posee enlaces para lenguajes diferentes a C++, aunque es un proyecto en estudio y puede que en unos meses sea posible emplear Python o Perl.

HBase

El proyecto HBase es parte de Hadoop y busca implementar Bigtable en Java. A día de hoy es usable. Implementa las mismas operaciones que Hypertable, y es programable desde Jython. Veamos un ejemplo (ver Listado 1).

Este ejemplo es realmente simple: creamos una tabla, y una vez creada introducimos en ella una fila para después recuperarla y borrar la tabla. El proceso no es complicado (uno de los objetivos de todos estos sistemas es la simplicidad).

Imaginemos por un momento todo lo que ocurre en este proceso. Creamos una tabla sobre Hbase que trabaja sobre HDFS, y por tanto estará almacenando los datos de forma distribuida, guardando varias copias en distintos Chunkservers. Y todo ello con unas pocas líneas de código.

Conclusión

El mundo de la computación en nube avanza a pasos agigantados, pero el software libre le sigue los pasos apoyado por empresas como Yahoo! Hadoop ha conseguido una gran relevancia estando aún en su versión 0.16 y ha demostrado ser capaz de rendir en sistemas con miles de servidores.

El software libre no puede quedarse atrás en esta batalla, porque es ahora cuando se está definiendo el futuro de la informática. Si con Linux fue posible soñar con conquistar el escritorio, espere-mos que proyectos como Hadoop nos permitan soñar con no acabar en manos de dos o tres empresas de cuya tecnología no podremos disponer. ■

Listado 1: Bigtable.py

```
01 import java.lang 19
02 admin.deleteTable(tablename_text)
03 from org.apache.hadoop.hbase 20
import HBaseConfiguration, admin.createTable(desc)
HBaseAdmin, HTableDescriptor, 21
HColumnDescriptor, HTable, 22
HConstants 23
04 tablas = admin.listTables()
05 nombre_tabla = "test" 24
06 nombre_tabla_text = 25
Text(nombre_tabla) 26
07 desc = 27
HTableDescriptor(nombre_tabla_text) 28
08 desc.addFamily(HColumnDescriptor("contenido:")) 29
09 desc.addFamily(HColumnDescriptor("enlace:")) 30
10 admin = HBaseAdmin(conf) 31
11 # Borramos la tabla si ya 32
existe 33
12 if 34
admin.tableExists(tablename_text): 35
admin.deleteTable(desc.getName()) 36
13 37
```

RECURSOS

- [1] <http://open.nytimes.com/2007/11/01/self-service-prorated-super-computing-fun/>
- [2] <http://hadoop.apache.org/>
- [3] <http://www.hypertable.org>