Bisoños Usuarios de GNU/Linux de Mallorca y Alrededores | Bergantells Usuaris de GNU/Linux de Mallorca i Afegitons

# 2.4.11, buffer-cache y page-cache unificados (20069 lectures)

Per **Ricardo Galli Granada**, ***gallir*** (http://mnm.uib.es/gallir/)

Creado el 11/10/2001 02:34 modificado el 11/10/2001 02:34

> *Según he leido en la lista de correo de linux-kernel[1] en el kernel versión 2.4.11 se ha logrado algo bastante complicado pero con el cual se logra una mejora importante del rendimiento de operaciones de lectura/escritura sobre ficheros: se han unificado las funciones de buffer-cache y page-cache. Lo explico brevemente.*

Las técnicas conocidas genéricamente como buffer-cache en Unix son aquellas que guardan en memoria RAM los bloques de discos recientemente leídos y/o modificados. Pero con importantes matizaciones, existen diferencias entre guardar sólo la meta-información (i-nodos, super-bloques, etc.) y los datos. Además las diferencias entre tratarlos como **partes de un fichero** o sólo como **bloques lógicos de discos** indexados por dispositivo y número de sector.

En sistemas Unix primitivos se denominaba buffer-cache a la memoria usada para mantener la meta información y los datos. En sistemas Unix más modernos esta técnica se separó en dos partes:

1. tradicionalmente la meta-información en el buffer-cache,
2. los datos en las páginas del sistema de memoria virtual.

Sin embargo el término se usa de modo genérico e indistinto desde hace muchos años.

Pero la cosa no es tan simple, al menos no en Linux, sobre todo por la diferencia en tratar a los datos leídos del disco como **bloques lógicos individuales** del disco o como **partes de ficheros**.

¿Cuál es la diferencia? Un ejemplo sencillo, el **read-ahead**. Esta técnica, implementada en el kernel Linux (por software) como en los discos (hardware), hace que cuando se lea un conjunto de sectores del disco, también se lean los consecutivos con el objetivo de ahorrarse más operaciones de lectura si luego el programa quiere leer los datos de forma secuencial. Pero...

Estamos asumiendo que los bytes de un fichero se almacenan siempre en sectores o bloques consecutivos, aunque el Linux lo intenta, no siempre es posible. Así que tendremos unos sectores que posiblemente sean usado por el fichero o quizás no. Además...

¿Como sabemos a que ficheros corresponden esos sectores? Ah... estamos hablando de **dos niveles** muy distintos en el sistema operativo:

1. Por un lado y a muy bajo nivel, la gestión de E/S a los dispositivos de bloques,
2. por otro lado la gestión de ficheros, a un nivel muy superior en la jerarquía.

Para poder hacer uso de esos sectores leídos, se necesitan hacer un ***mapping* inverso**. De un sector determinado de disco, convertir a información relevante para el sistema de ficheros. Pero es que además...

El Linux soporta varios sistemas de ficheros distintos (ext2, ext3, ReiserFS, XFS, NTFS, HPFS...), por lo que se implementó una capa de alto nivel, llamado *Virtual File System (VFS)* que es una abstracción de un sistema de fichero genérico para que la implementación de las técnicas de buffer-cache se implementen una vez para todos los sistemas (entre otras cosas).

El VFS, con el objetivos de simplificar un problema complicado, mantenía dos copias distintas, con sus respectivas

listas: el **buffer-cache** y el **page-cache**.

En el buffer-cache se mantenían los bloques leídos del disco, sin casi relación con el sistema de gestión de memoria virtual. La metainformación más importante eran número de dispostivo y número de sector (recordar que un bloque puede ser igual a uno o varios sectores). Cada vez que el sistema de ficheros intentaba leer o escribir un bloque de disco, calculaba la clave de dispersión (*hashing*) y se fijaba si estaba en el buffer cache. Si lo estaba hacía las copias correspondientes hacia o desde el page-cache.

¿Porque esta diferenciación? En realidad es bastante sencilla, normalmente el page-cache mantiene los bytes que **van exactamente en una página** del sistema de memoria virtual (4096 bytes en x86), mientras que los bloques lógicos de disco pueden, y suelen, ser de tamaños distintos (de 512 bytes a varios kilobytes). Además, **una página del cache puede estar formada por varios bloques de disco no consecutivos** (no siempre se pueden almacenar en bloques consecutivos).

La copia de datos entre el buffer-cache y el page-cache era bastante ineficiente y fue usada hasta la versión **2.2.x**. En la versión **2.4.x** ya se unificaron las estructuras de datos para que no hiciese falta copiar físicamente los datos, sino que se compartían los datos y se cambiaban los punteros para cambiar los elementos entre las listas del page-cache y el buffer-cache (o compartirlos).

Esta técnica del page-cache, aunque parezca que complica las cosas (¿porque no usar sólo el buffer-cache?), además de mantener *alineados* los datos con el sistema de gestión de memoria, permite tratar de forma uniforme (hasta para el *swapping* inclusive las páginas de código ejecutable de los programas o librerías compartidas. ¿¡¡¡Como!!!? Fácil, esos datos siempre se leen de disco, por lo que habrán pasado por el page-cache, por lo tanto están en memoria y se puede hacer un uso liberal de las mismas (haced un free en un **2.4.11** y veréis la diferencia, dice que sólo usa unos pocos MB de RAM):

```
 gallir@linux:/etc/kde2$ free
total       used       free     shared    buffers     cached Mem:        513620
480004      33616          0      50972     421412 -/+ buffers/cache:       7620
506000                                        ^^^^ Swap:       503992          0     503992
```

Los datos anteriores son de un servidor que tiene el Apache, Postgres y MySQL ejecutándose. Comparemoslo con el **2.4.4** del servidor de Bulma:

```
[gallir@m3d gallir]$ free
                        total       used       free     shared    buffers     cached
Mem:        255300     250632       4668          0       2220     107384
-/+ buffers/cache:     141028     114272
                                      ^^^^^^
Swap:       208804         84     208720
```

Pero aún así, con la compartición de estructuras de datos entre el buffer-cache y el page-cache había **ineficiencias y problemas**. Por ejemplo, si se hacía una copia de seguridad con el *dump*, éste leía los datos del buffer-cache, pero no de la page-cache. Había **problemas de coherencia** serios entre ambas listas. ¿Que pasaba si un bloque estaba en el page-cache (porque acaban de ser modificados por el sistema de ficheros) pero no en el buffer-cache? Ah!, esos datos no iban al backup...

Existían otros problemas de eficiencia, como los que comentaba al principio y además el trabajo extra de estar cambiando punteros y asegurar en todo momento que ambas listas se mantuviesen coherentes a la hora de ser accedidas por los distintos módulos del kernel.

Eso parece haber cambiado completamente a partir del 2.4.11 (aunque yo creo que el primer gran cambio fué en el 2.4.10) donde se cambió radicalmente la gestión de memoria (el VM) al notar Linus que había varios bugs en la gestión de la memoria virtual.

Ahora la gestión del page-cache y buffer-cache están unificadas en el tratamiento interno del kernel, **todo pasa por el page-cache**. Por lo que en principio se acabaron los problemas de coherencia y además facilita enormemente la tarea al gestor de memoria virtual. Ya no tiene que sopesar cuanta memoria se deja para el buffer-cache y cuanto para el page-cache. Ni estar continuamente haciendo el *mapping* inverso enre bloques lógicos de disco y sistemas de ficheros.

Creo que se ha dado un paso muy importante, casi nadie lo recordará ni se percata además de los *hackers* del kernel. De hecho casi ni sale en la *prensa* de Internet especializada. Pero estoy seguro que será tema de estudio de las asignaturas de sistemas operativos y que por fín se eliminará la ambigüedad del término buffer-cache. **Ahora podremos llamarlo directamente page-cache** (bueno, ojalá...).

Por ahora es suficiente, ya son las 4 de la madrugada..., ya seguiremos con el tema en cuanto tenga más información y entienda un poco más los cambios. Se dice que sólo diez personas entienden el funcionamiento del la memoria virtual y el buffer-page-cache del Linux. Y yo no estoy entre esos privilegiados ;-)

> Mas info en las notas de clase de Sistemas[2] de la Universidad de Tennessee. Aunque **no diferencia entre buffer-cache y page-cache**, explica bien los algoritmos y las estructuras de datos utilizadas.

-- Ricardo Galli

---

**Lista de enlaces de este artículo:**

1. http://www.cs.helsinki.fi/linux/linux-kernel/2001-40/1002.html
2. http://www.cs.utk.edu/~rich/classes/cs560/buffer-cache/lecture.html

---

E-mail del autor: gallir _ARROBA_ uib.es
**Podrás encontrar este artículo e información adicional en: http://bulma.net/body.phtml?nIdNoticia=908**