Linux Page Cache

Der **Page Cache** unter Linux beschleunigt zahlreiche Lese-Zugriffe von Dateien. Dies geschieht, indem Linux Daten beim erstmaligen Lesen von oder Schreiben auf Datenträgern wie Festplatten zusätzlich in **ungenutzten Bereichen des Arbeitsspeichers** cacht. Werden diese Daten später erneut gelesen, können diese schnell aus dem Arbeitsspeicher gelesen werden. Dieser Artikel liefert wertvolle Hintergrundinformationen zum Page Cache.

Page Cache oder Buffer Cache

Oft wird für den Page Cache noch der Begriff Buffer Cache verwendet. Bei Linux Kerneln bis Version 2.2 gab es sowohl einen Page Cache, als auch einen Buffer Cache. Mit Kernel 2.4 wurden diese beiden Caches vereint. Heute gibt es nur noch einen Cache, den Page Cache.[1]

Funktionsweise

Auslastung

Unter Linux gibt das Kommando `free -m` in der Spalte **cached** an, wieviel MB des Arbeitsspeichers momentan für den Page Cache verwendet werden:

```
[root@testserver ~]# free -m
             total       used       free     shared    buffers     cached
Mem:         15976      15195        781          0        167       9153
-/+ buffers/cache:       5874      10102
Swap:         2000          0       1999
[root@testserver ~]#
```

Schreiben

Wenn Daten geschrieben werden, kommen diese zuerst in den Page Cache und werden dort als *Dirty Pages* verwaltet. *Dirty* deshalb, weil diese Daten zwar im Page Cache liegen, aber erst auf das darunterliegende Speichergerät geschrieben werden müssen. Der Inhalt dieser *Dirty Pages* wird regelmäßig (bzw. auch beim Aufruf von Systemcalls wie sync oder fsync) auf das darunterliegende Speichergerät weitergegeben. Dies kann in letzter Instanz etwa ein RAID-Controller oder direkt eine Festplatte sein.

Das folgende Beispiel zeigt die Erstellung einer 10 MByte großen Datei, die zuerst in den Page Cache geschrieben wird. Der Umfang der Dirty Pages steigt dadurch, bis diese in diesem Fall manuell per sync Kommando auf die darunterliegende SSD geschrieben werden:

```
wfischer@pc:~$ dd if=/dev/zero of=testfile.txt bs=1M count=10
10+0 records in
10+0 records out
10485760 bytes (10 MB) copied, 0,0121043 s, 866 MB/s
wfischer@pc:~$ cat /proc/meminfo | grep Dirty
Dirty:             10260 kB
wfischer@pc:~$ sync
wfischer@pc:~$ cat /proc/meminfo | grep Dirty
Dirty:                 0 kB
```

bis Kernel 2.6.31: pdflush

Bis inklusive Linux Kernel 2.6.31 sorgten die pdflush Threads dafür, dass Dirty Pages regelmäßig auf die darunterliegenden Speichergeräte geschrieben wurden.

ab Kernel 2.6.32: Per-backing-device based writeback

Da pdflush einige Performance-Nachteile hatte, entwickelte Jens Axboe für den Linux Kernel 2.6.32 einen neuen effektiveren writeback Mechanismus.[2]

Dabei gibt es nun Threads für jedes Gerät, wie folgendes Beispiel eines Rechners mit einer SSD (/dev/sda) und einer Festplatte (/dev/sdb) zeigt:

```
root@pc:~# ls -l /dev/sda
brw-rw---- 1 root disk 8, 0 2011-09-01 10:36 /dev/sda
root@pc:~# ls -l /dev/sdb
brw-rw---- 1 root disk 8, 16 2011-09-01 10:36 /dev/sdb
root@pc:~# ps -eaf | grep -i flush
root       935     2  0 10:36 ?        00:00:00 [flush-8:0]
root       936     2  0 10:36 ?        00:00:00 [flush-8:16]
```

Lesen

Nicht nur beim Schreiben, auch beim Lesen von Dateien kommen Dateiblöcke in den Page Cache. Wenn Sie z.B. eine 100 MB Datei zweimal hintereinander lesen, so wird der zweite Zugriff schneller sein, da die Dateiblöcke direkt aus dem Page Cache im Arbeitsspeicher kommen und nicht mehr erneut von der Festplatte gelesen werden müssen. Das folgende Beispiel zeigt, dass nach der Wiedergabe eines gut 200 MB großen Videos die Größe des Page Caches angestiegen ist:

```
user@adminpc:~$ free -m
             total       used       free     shared    buffers     cached
Mem:          3884       1812       2071          0         60       1328
-/+ buffers/cache:        424       3459
Swap:         1956          0       1956
```

```
user@adminpc:~$ vlc video.avi
[...]
user@adminpc:~$ free -m
             total       used       free     shared    buffers     cached
Mem:          3884       2056       1827          0         60       1566
-/+ buffers/cache:        429       3454
Swap:         1956          0       1956
user@adminpc:~$
```

Benötigt Linux mehr Arbeitsspeicher für normale Applikationen als aktuell frei ist, werden bereits länger nicht mehr genutzte Bereiche des Page Caches automatisch gelöscht.

Page Cache optimieren

Das automatische Caching von Dateiblöcken im Page Cache ist meistens sehr vorteilhaft. Manche Daten (etwa Logfiles oder MySQL-Dumps) werden aber oft nach dem Schreiben nicht mehr benötigt. Solche Dateiblöcke belegen also oft unnötig Platz im Page Cache. Andere Dateiblöcke, deren Caching mehr Vorteile bringen würde, fliegen durch die neuen Logfiles oder MySQL aus dem Page Cache hinaus.[3]

Bei Logdateien hilft Ihnen hier etwa ein regelmäßiges Logrotate mit gzip Kompromierung. Wenn eine Logdatei mit beispielsweise 500 MB durch logrotate und gzip auf 10 MB komprimiert wird, wird die ursprüngliche Logdatei und somit auch deren Cache ungültig. 490 MB im Page Cache werden dadurch frei. Die Gefahr, dass eine kontinuierlich wachsenende Logdatei dadurch sinnvollere Dateiblöcke aus dem Page Cache "hinausschiebt" sinkt somit.

Es macht also durchaus Sinn, wenn manche Anwendungen bestimmte Dateien/Dateiblöcke gar nicht cachen würden. Für rsync gibt es dazu auch bereits einen Patch.[4]

Revision #2
Created 2025-12-20 13:31:27 UTC by willi
Updated 2025-12-20 13:34:23 UTC by willi