The Linux Page Cache and pdflush: Theory of Operation and Tuning for Write-Heavy Loads

As you write out data ultimately intended for disk, Linux caches this information in an area of memory called the page cache. You can find out basic info about the page cache using tools like free, vmstat or top. See http://gentoo-wiki.com/FAQ_Linux_Memory_Management to learn how to interpret top's memory information, or atop to get an improved version.

Full information about the page cache only shows up by looking at /proc/meminfo. Here is a sample from a system with 4GB of RAM:

```
MemTotal:      3950112 kB
MemFree:        622560 kB
Buffers:         78048 kB
Cached:        2901484 kB
SwapCached:          0 kB
Active:        3108012 kB
Inactive:        55296 kB
HighTotal:           0 kB
HighFree:            0 kB
LowTotal:      3950112 kB
LowFree:        622560 kB
SwapTotal:     4198272 kB
SwapFree:      4198244 kB
Dirty:             416 kB
Writeback:           0 kB
Mapped:         999852 kB
Slab:            57104 kB
Committed_AS:  3340368 kB
PageTables:       6672 kB
VmallocTotal: 536870911 kB
VmallocUsed:     35300 kB
VmallocChunk: 536835611 kB
HugePages_Total:     0
HugePages_Free:      0
Hugepagesize:     2048 kB
```

The size of the page cache itself is the "Cached" figure here, in this example it's 2.9GB. As pages are written, the size of the "Dirty" section will increase. Once writes to disk have begun, you'll see the "Writeback" figure go up until the write is finished. It can be very hard to actually catch the Writeback value going high, as its value is very transient and only increases during the brief period when I/O is queued but not yet written.

Linux usually writes data out of the page cache using a process called pdflush. At any moment, between 2 and 8 pdflush threads are running on the system. You can monitor how many are active by looking at **/proc/sys /vm/nr_pdflush_threads**. Whenever all existing pdflush threads are busy for at least one second, an additional pdflush daemon is spawned. The new ones try to write back data to device queues that are not congested, aiming to have each device that's active get its own thread flushing data to that device. Each time a second has passed without any pdflush activity, one of the threads is removed. There are tunables for adjusting the minimum and maximum number of pdflush processes, but it's very rare they need to be adjusted.

pdflush tunables

Exactly what each pdflush thread does is controlled by a series of parameters in /proc/sys/vm:

/proc/sys/vm/dirty_writeback_centisecs (default 500): In hundredths of a second, this is how often pdflush wakes up to write data to disk. The default wakes up the two (or more) active threads every five seconds.

There can be undocumented behavior that thwarts attempts to decrease dirty_writeback_centisecs in an attempt to make pdflush more aggressive. For example, in early 2.6 kernels, the Linux mm/page-writeback.c code includes logic that's described as "if a writeback event takes longer than a dirty_writeback_centisecs interval, then leave a one-second gap". In general, this "congestion" logic in the kernel is documented only by the kernel source itself, and how it operates can vary considerably depending on which kernel you are running. Because of all this, it's unlikely you'll gain much benefit from lowering the writeback time; the thread spawning code assures that they will automatically run themselves as often as is practical to try and meet the other requirements.

The first thing pdflush works on is writing pages that have been dirty for longer than it deems acceptable. This is controlled by:

/proc/sys/vm/dirty_expire_centiseconds (default 3000): In hundredths of a second, how long data can be in the page cache before it's considered expired and must be written at the next opportunity. Note that this default is very long: a full 30 seconds. That means that under normal circumstances, unless you write enough to trigger the other pdflush method, Linux won't actually commit anything you write until 30 seconds later.

The second thing pdflush will work on is writing pages if memory is low. This is controlled by:

/proc/sys/vm/dirty_background_ratio (default 10): Maximum percentage of active that can be filled with dirty pages before pdflush begins to write them

Note that some kernel versions may internally put a lower bound on this value at 5%.

Most of the documentation you'll find about this parameter suggests it's in terms of total memory, but a look at the source code shows this isn't true. In terms of the meminfo output, the code actually looks at

```
MemFree + Cached - Mapped
```

So on the system above, where this figure gives 2.5GB, with the default of 10% the system actually begins writing when the total for Dirty pages is slightly less than 250MB--not the 400MB you'd expect based on the total memory figure.

Summary: when does pdflush write?

In the default configuration, then, data written to disk will sit in memory until either a) they're more than 30 seconds old, or b) the dirty pages have consumed more than 10% of the active, working memory. If you are writing heavily, once you reach the dirty_background_ratio driven figure worth of dirty memory, you may find that all your writes are driven by that limit. It's fairly easy to get in a situation where pages are always being written out by that mechanism well before they are considered expired by the dirty_expire_centiseconds mechanism.

Other than laptop_mode, which changes several parameters to optimize for keeping the hard drive spinning as infrequently as possible (see http://www.samwel.tk/laptop_mode/ for more information) those are all the important kernel tunables that control the pdflush threads.

Process page writes

There is another parameter involved though that can spill over into management of user processes:

/proc/sys/vm/dirty_ratio (default 40): Maximum percentage of total memory that can be filled with dirty pages before processes are forced to write dirty buffers themselves during their time slice instead of being allowed to do more writes.

Note that all processes are blocked for writes when this happens, not just the one that filled the write buffers. This can cause what is perceived as an unfair behavior where one "write-hog" process can block all I/O on the system. The classic way to trigger this behavior is to execute a script that does "dd if=/dev/zero of=hog" and watch what happens. See Kernel Korner: I/O Schedulers for examples showing this behavior.

Tuning Recommendations for write-heavy operations

The usual issue that people who are writing heavily encouter is that Linux buffers too much information at once, in its attempt to improve efficiency. This is particularly troublesome for operations that require synchronizing the filesystem using system calls like fsync. If there is a lot of data in the buffer cace when this call is made, the system can freeze for quite some time to process the sync.

Another common issue is that because so much must be written before any phyiscal writes start, the I/O appears more bursty than would seem optimal. You'll have long periods where no physical writes happen at all, as the large page cache is filled, followed by writes at the highest speed the device can achieve once one of the pdflush triggers is tripped.

dirty_background_ratio: Primary tunable to adjust, probably downward. If your goal is to reduce the amount of data Linux keeps cached in memory, so that it writes it more consistently to the disk rather than in a batch, lowering dirty_background_ratio is the most effective way to do that. It is more likely the default is too large in situations where the system has large amounts of memory and/or slow physical I/O.

dirty_ratio: Secondary tunable to adjust only for some workloads. Applications that can cope with their writes being blocked altogether might benefit from substantially lowering this value. See "Warnings" below before adjusting.

dirty_expire_centisecs: Test lowering, but not to extremely low levels. Attempting to speed how long pages sit dirty in memory can be accomplished here, but this will considerably slow average I/O speed because of how much less efficient this is. This is particularly true on systems with slow physical I/O to disk. Because of the way the dirty page writing mechanism works, trying to lower this value to be very quick (less than a few seconds) is unlikely to work well. Constantly trying to write dirty pages out will just trigger the I/O congestion code more frequently.

dirty_writeback_centisecs: Leave alone. The timing of pdflush threads set by this parameter is so complicated by rules in the kernel code for things like write congestion that adjusting this tunable is unlikely to cause any real effect. It's generally advisable to keep it at the default so that this internal timing tuning matches the frequency at which pdflush runs.

Swapping

By default, Linux will aggressively swap processes out of physical memory onto disk in order to keep the disk cache as large as possible. This means that pages that haven't been used recently will be pushed into swap long before the system even comes close to running out of memory, which is an unexpected behavior compared to some operating systems. The /proc/sys /vm/swappiness parameter controls how aggressive Linux is in this area.

As good a description as you'll find of the numeric details of this setting is in section 4.15 of http://people.redhat.com/nhorman/papers/rhel4_vm.pdf It's based on a combination of how much of memory is mapped (that total is in /proc/meminfo) as well as how difficult it has been for the virtual memory manager to find pages to use.

A value of 0 will avoid ever swapping out just for caching space. Using 100 will always favor making the disk cache bigger. Most distributions set this value to be 60, tuned toward moderately aggressive swapping to increase disk cache.

The optimal setting here is very dependant on workload. In general, high values maximize throughput: how much work your system gets down during a unit of time. Low values favor latency: getting a quick response time from applications. Some desktop users so favor low latency that they set swappiness to 0, so that user applications are never swapped to disk (as can happen when the system is executing background tasks while the user is away). That's perfectly reasonable if the amount of memory in the system exceeds the usual working set for the applications used. Servers that are very active and usually throughput bound could justify setting it to 100. On the flip side, a desktop system that is so limited in memory that every active byte helps might also prefer a setting of 100.

Since the size of the disk cache directly determines things like how much dirty data Linux will allow in memory, adjusting swappiness can greatly influence that behavior even though it's not directly tied to that.

Warnings

-There is a currently outstanding Linux kernel bug that is rare and difficult to trigger even intentionally on most kernel versions. However, it is easier to encounter when reducing dirty_ratio setting below its default. An introduction to the issue starts at http://lkml.org/lkml/2006/12/28/171 and comments about it not being specific to the current kernel release are at http://lkml.org/lkml/2006/12/28/131

-The standard Linux memory allocation behavior uses an "overcommit" setting that allows processes to allocate more memory than is actually available were they to all ask for their pages at once. This is aimed at increasing the amount of memory available for the page cache, but can be dangerous for some types of applications. See http://www.linuxinsight.com/proc_sys_vm_overcommit_memory.html for a note on the settings you can adjust. An example of an application that can have issues when overcommit is turned on is PostgreSQL; see "Linux Memory Overcommit" at http://www.postgresql.org/docs/current/static/kernel-resources.html for their warnings on this subject.

References: page cache

Neil Horman, "Understanding Virtual Memory in Red Hat Enterprise Linux 4" http://people.redhat.com/nhorman/papers/rhel4_vm.pdf

Daniel P. Bovet and Marco Cesati, "Understanding the Linux Kernel, 3rd edition", chapter 15 "The Page Cache". Available on the web at http://www.linux-security.cn/ebooks/ulk3-html/

Robert Love, "Linux Kernel Development, 2nd edition", chapter 15 "The Page Cache and Page Writeback"

"Runtime Memory Management", http://tree.celinuxforum.org/CelfPubWiki/RuntimeMemoryMeasurement

"Red Hat Enterprise Linux-Specific [Memory] Information", http://www.redhat.com/docs/manuals/enterprise/RHEL-4-Manual/admin-guide/s1-memory-rhlspec.html

"Tuning Swapiness", http://kerneltrap.org/node/3000

"FAQ Linux Memory Management", http://gentoo-wiki.com/FAQ_Linux_Memory_Management

From the Linux kernel tree:

- Documentation/filesystems/proc.txt (the meminfo documentation there originally from http://lwn.net/Articles/28345/)
- Documentation/sysctl/vm.txt

- Mm/page-writeback.c

References: I/O scheduling

While not directly addressed here, the I/O scheduling algorithms in Linux actually handle the writes themselves, and some knowledge or tuning of them may be synergistic with adjusting the parameters here. Adjusting the scheduler only makes sense in the context where you've already configured the page cache flushing correctly for your workload.

D. John Shakshober, "Choosing an I/O Scheduler for Red Hat Enterprise Linux 4 and the 2.6 Kernel" http://www.redhat.com/magazine/008jun05/features/schedulers/

Robert Love, "Kernel Korner: I/O Schedulers", http://www.linuxjournal.com/article/6931

Seelam, Romero, and Teller, "Enhancements to Linux I/O Scheduling", http://linux.inet.hr/files/ols2005/seelam-reprint.pdf

Heger, D., Pratt, S., "Workload Dependent Performance Evaluation of the Linux 2.6 I/O Schedulers", http://linux.inet.hr/files/ols2004/pratt-reprint.pdf

Upcoming Linux work in progress

-There is a patch in testing from SuSE that adds a parameter called dirty_ratio_centisecs to the kernel tuning which fine-tunes the write-throttling behavior. See "Patch: per-task predictive write throttling" at http://lwn.net/Articles/152277/ and Andrea Arcangeli's article (which has a useful commentary on the existing write throttling code) at http://www.lugroma.org/contenuti/eventi/LinuxDay2005/atti/Arcangeli-MemoryManagementKernel26.pdf

-SuSE also has suggested a patch at http://lwn.net/Articles/216853/ that allows setting the dirty_ratio settings below the current useful range, aimed at systems with very large memory capacity. The commentary on this patch also has some helpful comments on improving dirty buffer writing, although it is fairly specific to ext3 filesystems.

-The stock 2.6.22 Linux kernel has substantially reduced the default values for the dirty memory parameters. dirty_background_ratio defaulted to 10, now it defaults to 5. vm_dirty_ratio defaulted to 40, now it's 10

-A recent lively discussion on the Linux kernel mailing list discusses some of the limitations of the fsync mechanism when using ext3.