

Re: Slow Filesystem I/O

Source: <http://unix.derkeiler.com/Newsgroups/comp.os.vms/2005-04/1582.html>

From: Bill Todd (billtodd_at_metrocast.net)

Date: 04/24/05

Date: Sun, 24 Apr 2005 16:00:18 -0400

Dan Foster wrote:

...

- > *That's a shame (Spiralog). Wonder why then-DEC dropped it if it was*
- > *pretty good? There's some rumours of boundary conditions presenting*
- > *significant engineering work making it unpalatable to develop further?*

Spiralog was extremely interesting, but was not an unqualified success.

For example, while it helped some workloads enormously it provided little boost for others and actually caused some to slow down (e.g., when its on-disk layout caused files to become severely fragmented).

One of the fundamental assumptions behind log-structured storage was that memory was becoming cheap enough that soon most data would be cached and update performance (rather than read performance) would be the main bottleneck. The explosion in data sizes which accompanied increased memory sizes invalidated this assumption, and the drawbacks associated with log-structured storage prevented it from taking over the world.

>

> *Note to self: a quick primer on filesystem writes -- three major policies:*

>

- > *a) Write-through caching -- app writes to fs; fs doesn't return*
- > *until all data blocks has been successfully written to disk.*

Or at least is guaranteed to be persistent at the relevant level of redundancy (e.g., if the disks are mirrored, then if I/O completion is returned when the data has reached battery-backed cache then that cache should be mirrored also).

Furthermore, any related metadata updates must also be persistent before the I/O completes (it doesn't do you too much good to have your appended data on the disk if EOF hasn't also been updated, for example).

>

> *Safest but quite slow for obvious reasons. Default on OpenVMS.*

Actually, it can be pretty fast. For example, if you've got a small update which can be wholly captured in a system transaction log, and that system log can leverage the existence of (perhaps mirrored, as described above) persistent cache and destaged to disk if necessary only later, in bulk, then many write-through updates can complete with no disk access latency at all (and even in the absence of such non-volatile cache require only a single small log write, potentially batched with log records for other such updates performed concurrently, rather than multiple disk accesses to update both the data and the associated metadata).

- >
- > *b) Write-back caching -- app writes to fs; fs returns as soon as data is in cache, even if cache has not yet flushed to disk.*
- >
- > *Very unsafe -- very good performance but if interrupted (e.g. power failure) at wrong time, filesystem data corruption.*

Not necessarily. For example, the 'soft updates' algorithms in *BSD systems prevent data corruption in the presence of write-back caching by ensuring that when the updates actually get to disk they do so in the appropriate order – very similar to the ordering which ODS-1/2 enforces on its own synchronous disk updates and to the deferred ordering which Spiralog enforced on its disk updates.

Furthermore, a similar ordering can be enforced in the presence of a transaction log, such that all writes (save those specified by the application as synchronous) can be deferred.

- >
- > *c) Write-behind caching -- app writes to fs; fs reorders data as necessary in order to guarantee that data will never be corrupted even if interrupted mid-write, and also retains performance properties of write-back caching.*

The definition of 'write-behind' that I'm familiar with is simply a process which uses multiple buffers to allow it to accumulate new dirty data while previous dirty data is being written out to disk as fast as it can be. In other words, it's pretty similar to write-back caching, just maximally aggressive in its writing.

- >
- > *Provides performance of write-back with safety of write-through.*

No: it may guard against corruption (as the other mechanisms I described above do as well), but it still does not tell you *when* the data has actually made it to disk.

– bill