Deadline IO scheduler tunables

This little file attempts to document how the deadline io scheduler works. In particular, it will clarify the meaning of the exposed tunables that may be of interest to power users.

Selecting IO schedulers

Refer to Documentation/block/switching-sched.rst for information on selecting an io scheduler on a per-device basis.

---

read_expire (in ms)

The goal of the deadline io scheduler is to attempt to guarantee a start service time for a request. As we focus mainly on read latencies, this is tunable. When a read request first enters the io scheduler, it is assigned a deadline that is the current time + the read_expire value in units of milliseconds.

write_expire (in ms)

Similar to read_expire mentioned above, but for writes.

fifo_batch (number of requests)

Requests are grouped into `batches` of a particular data direction (read or write) which are serviced in increasing sector order. To limit extra seeking, deadline expiries are only checked between batches. fifo_batch controls the maximum number of requests per batch.

This parameter tunes the balance between per-request latency and aggregate throughput. When low latency is the primary concern, smaller is better (where a value of 1 yields first-come first-served behaviour). Increasing fifo_batch generally improves throughput, at the cost of latency variation.

writes_starved (number of dispatches)

When we have to move requests from the io scheduler queue to the block device dispatch queue, we always give a preference to reads. However, we don't want to starve writes indefinitely either. So writes_starved controls how many times we give preference to reads over writes. When that has been done writes_starved number of times, we dispatch some writes based on the same criteria as reads.

front_merges (bool)

Sometimes it happens that a request enters the io scheduler that is contiguous with a request that is already on the queue. Either it fits in the back of that request, or it fits at the front. That is called either a back merge candidate or a front merge candidate. Due to the way files are typically laid out, back merges are much more common than front merges. For some work loads, you may even know that it is a waste of time to spend any time attempting to front merge requests. Setting front_merges to 0 disables this functionality. Front merges may still occur due to the cached last_merge hint, but since that comes at basically 0 cost we leave that on. We simply disable the rbtree front sector lookup when the io scheduler merge function is called.

Nov 11 2002, Jens Axboe <jens.axboe@oracle.com>