Explicit volatile write back cache control

Introduction

Many storage devices, especially in the consumer market, come with volatile write back caches. That means the devices signal I/O completion to the operating system before data actually has hit the non-volatile storage. This behavior obviously speeds up various workloads, but it means the operating system needs to force data out to the non-volatile storage when it performs a data integrity operation like fsync, sync or an unmount.

The Linux block layer provides two simple mechanisms that let filesystems control the caching behavior of the storage device. These mechanisms are a forced cache flush, and the Force Unit Access (FUA) flag for requests.

Explicit cache flushes

The REQ_PREFLUSH flag can be OR ed into the r/w flags of a bio submitted from the filesystem and will make sure the volatile cache of the storage device has been flushed before the actual I/O operation is started. This explicitly guarantees that previously completed write requests are on non-volatile storage before the flagged bio starts. In addition the REQ_PREFLUSH flag can be set on an otherwise empty bio structure, which causes only an explicit cache flush without any dependent I/O. It is recommend to use the blkdev_issue_flush() helper for a pure cache flush.

Forced Unit Access

The REQ_FUA flag can be OR ed into the r/w flags of a bio submitted from the filesystem and will make sure that I/O completion for this request is only signaled after the data has been committed to non-volatile storage.

Implementation details for filesystems

Filesystems can simply set the REQ_PREFLUSH and REQ_FUA bits and do not have to worry if the underlying devices need any explicit cache flushing and how the Forced Unit Access is implemented. The REQ_PREFLUSH and REQ_FUA flags may both be set on a single bio.

Implementation details for bio based block drivers

These drivers will always see the REQ_PREFLUSH and REQ_FUA bits as they sit directly below the submit_bio interface. For remapping drivers the REQ_FUA bits need to be propagated to underlying devices, and a global flush needs to be implemented for bios with the REQ_PREFLUSH bit set. For real device drivers that do not have a volatile cache the REQ_PREFLUSH and REQ_FUA bits on non-empty bios can simply be ignored, and REQ_PREFLUSH requests without data can be completed successfully without doing any work. Drivers for devices with volatile caches need to implement the support for these flags themselves without any help from the block layer.

Implementation details for request_fn based block drivers

For devices that do not support volatile write caches there is no driver support required, the block layer completes empty REQ_PREFLUSH requests before entering the driver and strips off the REQ_PREFLUSH and REQ_FUA bits from requests that have a payload. For devices with volatile write caches the driver needs to tell the block layer that it supports flushing caches by doing:

```
blk_queue_write_cache(sdkp->disk->queue, true, false);
```

and handle empty REQ_OP_FLUSH requests in its prep_fn/request_fn. Note that REQ_PREFLUSH requests with a payload are automatically turned into a sequence of an empty REQ_OP_FLUSH request followed by the actual write by the block layer. For devices that also support the FUA bit the block layer needs to be told to pass through the REQ_FUA bit using:

```
blk_queue_write_cache(sdkp->disk->queue, true, true);
```

and the driver must handle write requests that have the REQ_FUA bit set in prep_fn/request_fn. If the FUA bit is not natively supported the block layer turns it into an empty REQ_OP_FLUSH request after the actual write.