Optimizing SquashFS at the Kernel Level

SquashFS is a compressed read-only filesystem for Linux. The file system is read-only by design and thus suitable for use on the system partition. Many Android devices may benefit from using this file system for their system partition, for example, the following:

- Devices with a low capacity storage such as Android Watch.
- Devices with a slow flash storage (compression reduces the number of block I/Os).

Unfortunately the performance of SquashFS lags behind ext4.

Optimizations

The following optimizations have been implemented to improve the performance of SquashFS.

Reduce the memory usage and memcpy

When reading a block (default 128K), SquashFS tries to grab all the pages covering this block.

If a single page is up-to-date or locked, it falls back to allocating a full block, submitting a read request, and then copying its content to the pages.

This approach is very ineffective; after some time the page cache is likely to contain pages that are up-to-date even if the adjacent pages are not.

The code is now able to handle blocks with holes (=missing pages). This improves performance in the following ways:

- Reduces the number of **`memcpy`** calls
- Decreases memory allocations

Asynchronous reads

SquashFS still uses the deprecated **`ll_rw_block()`** function. There are two problems with this approach:

- As the name implies, the function waits for the reads to complete before returning. This is redundant since **`.readpage()`** already waits on the page's lock. Moreover, we need an asynchronous mechanism to efficiently implement **`.readpages()`**.
- Merging the read requests entirely depends on the I/O scheduler. **`ll_rw_block()`** simply creates one request per buffer. SquashFS has more information than the I/O scheduler about what should be merged. Moreover, merging the request means that we rely less on the I/O scheduler.

For these reasons, the **`ll_rw_block()`** function has been replaced with **`submit_bio()`**.

Readpages (prefetching)

SquashFS does not implement **`.readpages()`**, so the kernel repeatedly calls **`.readpage()`**.

Now that our read requests are asynchronous, the kernel can truly prefetch pages using its asynchronous read-ahead mechanism.

Optimize reading uncompressed blocks

Modern systems such as Android contain a lot of files that are already compressed. As a consequence, the image contains a lot of blocks that can't be compressed.

SquashFS handles compressed and uncompressed blocks using the same logic: when asked to read a single page, it actually reads a full block (default 128k). While this is necessary for compressed blocks, it is just a waste of resources for uncompressed blocks.

Instead of reading a full block, SquashFS now just reads what is advised by the readahead algorithm.

This greatly improves the performance of random reads.

Code

SquashFS code optimizations are available in AOSP:

- https://android-review.googlesource.com/#/q/topic:squashfs (https://android-review.googlesource.com/#/q/topic:squashfs)

Last updated August 21, 2017.