zswap

Overview

Zswap is a lightweight compressed cache for swap pages. It takes pages that are in the process of being swapped out and attempts to compress them into a dynamically allocated RAM-based memory pool. zswap basically trades CPU cycles for potentially reduced swap I/O. This trade-off can also result in a significant performance improvement if reads from the compressed cache are faster than reads from a swap device.

> **Note**
>
> Zswap is a new feature as of v3.11 and interacts heavily with memory reclaim. This interaction has not been fully explored on the large set of potential configurations and workloads that exist. For this reason, zswap is a work in progress and should be considered experimental.
>
> Some potential benefits:

- Desktop/laptop users with limited RAM capacities can mitigate the performance impact of swapping.
- Overcommitted guests that share a common I/O resource can dramatically reduce their swap I/O pressure, avoiding heavy handed I/O throttling by the hypervisor. This allows more work to get done with less impact to the guest workload and guests sharing the I/O subsystem
- Users with SSDs as swap devices can extend the life of the device by drastically reducing life-shortening writes.

Zswap evicts pages from compressed cache on an LRU basis to the backing swap device when the compressed pool reaches its size limit. This requirement had been identified in prior community discussions.

Whether Zswap is enabled at the boot time depends on whether the `CONFIG_ZSWAP_DEFAULT_ON` Kconfig option is enabled or not. This setting can then be overridden by providing the kernel command line `zswap.enabled=` option, for example `zswap.enabled=0`. Zswap can also be enabled and disabled at runtime using the sysfs interface. An example command to enable zswap at runtime, assuming sysfs is mounted at `/sys`, is:

```
echo 1 > /sys/module/zswap/parameters/enabled
```

When zswap is disabled at runtime it will stop storing pages that are being swapped out. However, it will _not_ immediately write out or fault back into memory all of the pages stored in the compressed pool. The pages stored in zswap will remain in the compressed pool until they are either invalidated or faulted back into memory. In order to force all pages out of the compressed pool, a swapoff on the swap device(s) will fault back into memory all swapped out pages, including those in the compressed pool.

Design

Zswap receives pages for compression through the Frontswap API and is able to evict pages from its own compressed pool on an LRU basis and write them back to the backing swap device in the case that the compressed pool is full.

Zswap makes use of zpool for the managing the compressed memory pool. Each allocation in zpool is not directly accessible by address. Rather, a handle is returned by the allocation routine and that handle must be mapped before being accessed. The compressed memory pool grows on demand and shrinks as compressed pages are freed. The pool is not preallocated. By default, a zpool of type selected in `CONFIG_ZSWAP_ZPOOL_DEFAULT` Kconfig option is created, but it can be overridden at boot time by setting the `zpool` attribute, e.g. `zswap.zpool=zbud`. It can also be changed at runtime using the sysfs `zpool` attribute, e.g.:

```
echo zbud > /sys/module/zswap/parameters/zpool
```

The zbud type zpool allocates exactly 1 page to store 2 compressed pages, which means the compression ratio will always be 2:1 or worse (because of half-full zbud pages). The zsmalloc type zpool has a more complex compressed page storage method, and it can achieve greater storage densities. However, zsmalloc does not implement compressed page eviction, so once zswap fills it cannot evict the oldest page, it can only reject new pages.

When a swap page is passed from frontswap to zswap, zswap maintains a mapping of the swap entry, a combination of the swap type and swap offset, to the zpool handle that references that compressed swap page. This mapping is achieved with a red-black tree per swap type. The swap offset is the search key for the tree nodes.

During a page fault on a PTE that is a swap entry, frontswap calls the zswap load function to decompress the page into the page allocated by the page fault handler.

Once there are no PTEs referencing a swap page stored in zswap (i.e. the count in the swap_map goes to 0) the swap code calls the zswap invalidate function, via frontswap, to free the compressed entry.

Zswap seeks to be simple in its policies. Sysfs attributes allow for one user controlled policy:

- max_pool_percent - The maximum percentage of memory that the compressed pool can occupy.

The default compressor is selected in `CONFIG_ZSWAP_COMPRESSOR_DEFAULT` Kconfig option, but it can be overridden at boot time by setting the `compressor` attribute, e.g. `zswap.compressor=lzo`. It can also be changed at runtime using the sysfs "compressor" attribute, e.g.:

```
echo lzo > /sys/module/zswap/parameters/compressor
```

When the zpool and/or compressor parameter is changed at runtime, any existing compressed pages are not modified; they are left in their own zpool. When a request is made for a page in an old zpool, it is uncompressed using its original compressor. Once all pages are removed from an old zpool, the zpool and its compressor are freed.

Some of the pages in zswap are same-value filled pages (i.e. contents of the page have same value or repetitive pattern). These pages include zero-filled pages and they are handled differently. During store operation, a page is checked if it is a same-value filled page before compressing it. If true, the compressed length of the page is set to zero and the pattern or same-filled value is stored.

Same-value filled pages identification feature is enabled by default and can be disabled at boot time by setting the `same_filled_pages_enabled` attribute to 0, e.g. `zswap.same_filled_pages_enabled=0`. It can also be enabled and disabled at runtime using the sysfs `same_filled_pages_enabled` attribute, e.g.:

```
echo 1 > /sys/module/zswap/parameters/same_filled_pages_enabled
```

When zswap same-filled page identification is disabled at runtime, it will stop checking for the same-value filled pages during store operation. In other words, every page will be then considered non-same-value filled. However, the existing pages which are marked as same-value filled pages remain stored unchanged in zswap until they are either loaded or invalidated.

In some circumstances it might be advantageous to make use of just the zswap ability to efficiently store same-filled pages without enabling the whole compressed page storage. In this case the handling of non-same-value pages by zswap (enabled by default) can be disabled by setting the `non_same_filled_pages_enabled` attribute to 0, e.g. `zswap.non_same_filled_pages_enabled=0`. It can also be enabled and disabled at runtime using the sysfs `non_same_filled_pages_enabled` attribute, e.g.:

```
echo 1 > /sys/module/zswap/parameters/non_same_filled_pages_enabled
```

Disabling both `zswap.same_filled_pages_enabled` and `zswap.non_same_filled_pages_enabled` effectively disables accepting any new pages by zswap.

To prevent zswap from shrinking pool when zswap is full and there's a high pressure on swap (this will result in flipping pages in and out zswap pool without any real benefit but with a performance drop for the system), a special parameter has been introduced to implement a sort of hysteresis to refuse taking pages into zswap pool until it has sufficient space if the limit has been hit. To set the threshold at which zswap would start accepting pages again after it became full, use the sysfs `accept_threshold_percent` attribute, e. g.:

```
echo 80 > /sys/module/zswap/parameters/accept_threshold_percent
```

Setting this parameter to 100 will disable the hysteresis.

A debugfs interface is provided for various statistic about pool size, number of pages stored, same-value filled pages and various counters for the reasons pages are rejected.