Google

Swap Abstraction / Native Zswap

LSF/MM/BPF 2023

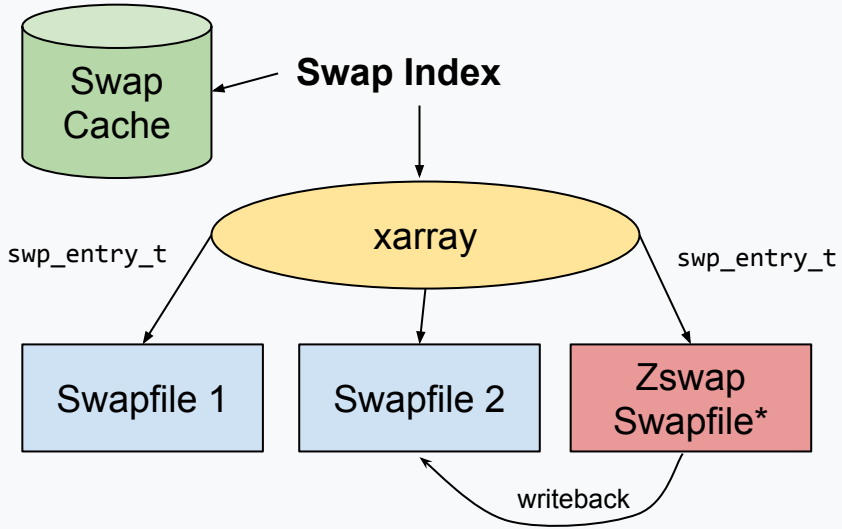
Yosry Ahmed (yosryahmed@google.com)

May 2023

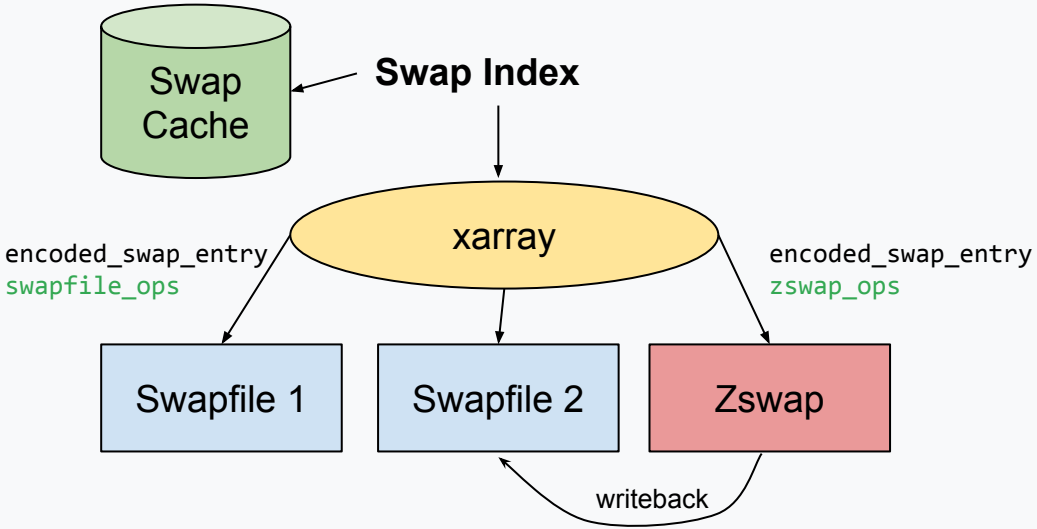
Problem

- Zswap is implemented as an in-memory compressed cache for swap, which means:
    - Zswap cannot be used without a backing swapfile.
    - Capacity is wasted on the backing swapfile.
    - Unnecessary work in zswap path (e.g. swap entry lookup).
    - Reclaim is unaware of zswap.
- Who cares?
    - Google’s prodkernel (and maybe others).
    - Meta.
    - kdevops.
    - Who else?

Short Term: Indirection Layer, Virtual Zswap Swapfile



Medium Term: Indirection Layer, Swap Ops

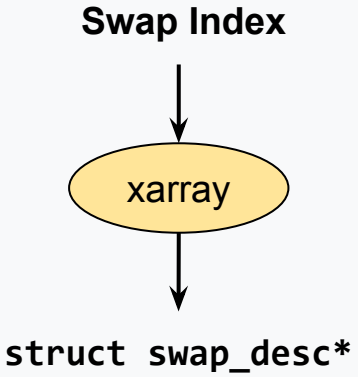



```
struct swap_ops {
    int (*readpage)(..);
    int (*writepage)(..);
    swp_entry_t (*alloc)(..);
    void (*duplicate)(..);
    void (*free)(..);
    ...
}
```

`encoded_swap_entry` can be a `swp_entry_t` or a `struct zswap_entry` pointer

Long Term: Swap Descriptors

```
struct swap_desc {
    encoded_swap_entry entry;
    struct folio *cache;
    u32 count;
}
```



Long Term: Swap Descriptors

- Pros
  - Cleaner abstraction.
  - Potential for a lot of code cleanups.
- Cons
  - Memory overhead for swapfiles (0.6-0.8% per swapped page).
  - Need a separate `swap_entry_t` → swap index reverse mapping for cluster readahead.
  - Larger surgery.

Long Term: VFS-Like Implementation For Swapfiles (Chris Li)

- Allow each swapfile has own VFS like implementation
  - Swap count.
  - Swap slot management.
- Allow swapfile with or without indirection layer co-exist.
- Don’t need to pay the price for indirection layer when not needed.

Google

Discussion