Design and Implementation of Autocaching for CXLSSD

Heekwon Park(heekwon.p@Samsung.com), Jongmin Gim, Jaemin Jung, Adam Manzanares, Davidlohr Bueso, Javier Gonzalez, Yang Seok Ki

- Samsung Electronics -
Agenda

• Motivation
• Design of Autocaching
• Implementation details & Challenges
• Preliminary results
• TO-DO List
Motivation: The advent of new types of storage device based on CXL

SAMSUNG Memory-Semantic CXL SSD (Announced at Flash Memory Summit 0222)

Intel CXL Optane (Proposed at Intel Tech Field Day 25(04/2022))

Optane PMem on CXL

CXL interface

NAND

DRAM

Cache CTRL

Load/Store

File system based access supports legacy NVMe

Load/store access to file data
Motivation: The advent of new types of storage device based on CXL

Autocaching can handle I/O efficiently for both architectures by leveraging DRAM and Device Memory simultaneously.
Motivation: Read Test

- CPU: Intel(R) Xeon(R) Gold 6338
- Available system memory: 32GB
- Optane DIMM: 126GB
- Kernel: linux-5.18.0
- Benchmark program: fio
- ioengine: mmap
- Random Read
- Block size: 4KB
- Direct is 0 (no msync after write)
Motivation: Write Test

- CPU: Intel(R) Xeon(R) Gold 6338
- Available system memory: 32GB
- Optane DIMM: 126GB
- Kernel: linux-5.18.0
- Benchmark program: fio
- ioengine: mmap
- Random Write
- Block size: 4MB
- Direct is 0 (no msync after write)

Flush daemon continues to write-back dirty pages to device and interferes application (lock page and cache pollution)

Optane could not support concurrent write efficiently
Motivation: Linux Kernel Internal

User Space

FS with page cache

VFS

Page Fault Handler

Application

(Read()) / (Write())

File APIs

Page Fault Handler

VFS

Filemap

F/S

FS DAX

User Virtual Memory

Kernel Virtual Memory

Create Mapping

DRAM

pmem/NVMe driver

pmem(CXL SSD, Optane, ...)

Kernel Space

FS with page cache + DAX

Page Fault Handler

VFS

Filemap

F/S

FS DAX

User Virtual Memory

Kernel Virtual Memory

Create Mapping

DRAM

pmem/CXL SSD

pmem(CXL SSD, Optane, ...)

H/W

FS with FSDAX (mount with –o dax)

Page Fault Handler

VFS

Filemap

F/S

FS DAX

User Virtual Memory

Kernel Virtual Memory

Create Mapping

DRAM

CXL.mem

pmem(CXL SSD, Optane, ...)

Hot data

Cold data

FS with page cache

Page cache + DAX

FS with FSDAX (mount with –o dax)
Motivation: Linux Kernel Internal

User Space

Kernel Space

H/W

Page cache + DAX

Application (mmap(), Read(), Write()) File APIs

Page Fault Handler

VFS Filemap

F/S FS DAX

User Virtual Memory

Kernel Virtual Memory

Create Mapping

Create Mapping

DRAM

CXL.io block I/O (CXLSSD) CXL.mem (Optane)

Hot data pmem

Cold data pmem(CXL SSD, Optane, ...)

Linux Plumbers Conference | Dublin, Ireland Sept. 12-14, 2022
Design of Autocaching

User Space

Kernel Space

H/W

• Hot data
• Throughput sensitive request (ex. Readahead, buffered I/O)
• Low memory utilization

• Cold data
• Latency sensitive request (ex. Page fault, direct I/O, Sync I/O)
• High memory utilization

Page Fault Handler

Page cache/Device page

Filemap

VFS

Create Mapping

Caching Transition

DRAM

CXL Mem (pmem)

CXL io (CXLSSD)

CXL SSD

Linux Plumbers Conference | Dublin, Ireland Sept. 12-14, 2022
Implementation of Autocaching

User Space

Virtual Memory

Page cache

Direct access

Device Physical memory

DRAM memory

mmap()

Memory Management

Memory allocator & page fault handler

Caching Transition

Page cache ↔ Device page

Read/write()

VFS

if (autocaching_range)
folio = autocaching_alloc_folio(mapping, offset, ...);
if (!folio)
folio = filemap_alloc_folio();

During mount
sb->autocaching_range = find_autocaching_range(sb, sb_loc)

File System

Autocaching

Mange device pages (device page initialization & allocation(decision))

Device Driver

Driver for .io

register_autocaching(range);

pmem driver

CXLSSD/Optane

CXL.mem

CXL.io
Implementation of Autocaching

• Challenges

1. Space overhead of device page structure
   • device page can waste precious DRAM memory

2. Cache transition
   • Detect warm page

3. Allocation Policy
   • Allocation policy to determine whether to allocate page cache or device pages
Implementation of Autocaching
- Space Overhead of device struct pages -

- Unlike ZONE_NORMAL pages, the ZONE_DEVICE pages may not be actively used if the corresponding blocks are not mapped to a file or the file has not been opened yet.

- Memory allocator allocates ZONE_NORMAL pages using page struct.
  - Inactive dram page’s struct pages are actively used.
  - File system allocates blocks using the block map
    - Inactive device page’s struct pages are not used at all.

- Even though page structures are not used at all, system should reserve memory for all struct pages belonging to ZONE_DEVICE
  - struct page size is 64B per 4KB ⇧ Need 64GB for 4TB Storage
  - Waste of system memory
    - The system reserves 64GB memory for device struct page on booting(or pmem device initialization), and the reserved region can not be used for any other purpose, even if CXL storage memory is not actively used.
Implementation of Autocaching
- Space Overhead of device struct pages -

• Dynamic memory allocation for device memory struct page
Implementation of Autocaching  
- Space Overhead of device struct pages -  

- Dynamic memory allocation for device memory struct page

```
Struct page{/*For thumbnail page*/
    flag & (1<<PG_thumbnail)
    union{
        struct{
            struct page* pages;
            spinlock_t lock;
            int count;
        }
    }
}``

Proposed struct page for Autocaching enabled device memory

```
0xFFFF C90 0000 0000 (vmalloc_base)
```

Linux kernel struct page for device memory

```
%ZONE_DEVICE's (struct page*)mem_map
%ZONE_DEVICE's (struct page*)mem_map
0xFFFF 8880 0000 0000 (page_offset_base)
```

Kernel Virtual Memory  
Physical Memory

64GB for a 4TB storage device

ZONE_DEVICE's (struct page*)mem_map

0xFFFF 8880 0000 0000 (page_offset_base)

Kernel Virtual Memory  
Physical Memory

32KB for 512 struct pages

128MB for a 4TB storage device

Dynamic allocation

28 bits  9 bits

Page Frame Number(PFN)

Thumbnail Page index  4KB page Index

4KB PAGE Index

CXL(2MB) pages'
PFN

47 53

Depends on paging level

Filemap(xarray) Node

Struct page{/*For thumbnail page*/
    flag & (1<<PG_thumbnail)
    union{
        struct{
            struct page* pages;
            spinlock_t lock;
            int count;
        }
    }
```
Implementation of Autocaching
- Space Overhead of device struct pages -

- Dynamic memory allocation for device memory struct page
### Implementation of Autocaching
- **Space Overhead of device struct pages**
- Dynamic memory allocation of struct page
- Address conversion macros

#### Address conversion macros

```c
int page_to_pfn(page)
{
    page - mem_map;
}

int pfnto_page(pfn)
{
    mem_map + pfn;
}
```

#### Proposed struct page for device memory

```c
#define PG_THM_SHIFT 0x9
#define PG_THM_MASK 0x1ff

struct page{
    flag & (1<<PG_thm)
    union{
        struct{
            struct page* pages;
            spinlock_t lock;
        }
    }
}
```

#### Address conversion macros

<table>
<thead>
<tr>
<th>Page Frame Number (PFN)</th>
<th>28 bits</th>
<th>9 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thumbnail Page’s index</td>
<td>4KB page Index</td>
<td></td>
</tr>
</tbody>
</table>

1. page->pfn; (add field)
2. use rmap (filemap node’s PFN)
3. Reserve VM for all struct pages in vmalloc area (to use struct page offset for PFN)

```c
int page_to_pfn(page)
{
    page - mem_map;
}

int pfnto_page(pfn)
{
    mem_map + pfn;
}
```
Implementation of Autocaching
- Caching Transition -

**Caching Transition: Demotion**
(Cache invalidation & Direct Mapping)

1. **Select victims**
2. **Get LBA in CXLSSD & convert to device folio**
3. **Free Folio**
4. **Update mapping**
5. **Free CXL storage**

**Caching Transition: Promotion**

1. **Allocate new pages**
2. **Allocate pages**
3. **Load data**
4. **Update mapping**
5. **Change folio status**
6. **Autocaching**

- User VM
- Node 0: DRAM
- Node 1: CXL storage
- NUMA Scan
  - Scan VM range periodically
  - if(is device folio?) Change the PTE to inaccessible
- VFS (filemap)
- File System (ext4)
- File Folio LRU
  - Active List
  - Inactive List
- CXL.mem
- CXL.io
- Memory Allocator
- User
Implementation of Autocaching
- Caching Transition -

• Flex zone
  • A promotion daemon virtually promotes accessed pages to flex zone and keeps watching whether the page is hot or warm.
  • A flex zone has a limited number of pages.
    • If ping-pong happens, increase the size of flex zone.
    • If the flex zone is stable, decrease the size of flex zone.
  • Keep warm(ping-pong) pages in flex zone.
    • Warm page detection based on IO type(read or write), access cycle, access frequency, and recently accessed pages

Filemap Node for CXL device page

To detect concurrent write, if PTE’s dirty bit is set or write(), store write ID(Hash(tid))

Access count

(Periodically check and left-shift by 3)
Implementation of Autocaching
- Allocation Policy -

- Allocation policy
  - Consider device characteristics
    - CXLSSD (dual interface, CXL.io & CXL.mem) & Optane (Support .mem only)
    - Provides policy control parameters to user space so that different devices can have different policies.
  - Use DRAM
    - File metadata
    - Inline data (store small size of data in inode)
    - New file write or append to support delayed allocation
  - Use device memory
    - sync I/O, Direct I/O

<table>
<thead>
<tr>
<th>metadata</th>
<th>Read operation</th>
<th>Write operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>memory usage &lt; 80%</td>
<td>Page cache</td>
<td>• Balance Page cache (50%) and Device page (50%)</td>
</tr>
<tr>
<td>80% &lt; memory usage &lt; 97% Decrease the DRAM allocation rate.</td>
<td></td>
<td>• Allocate page cache for concurrent write</td>
</tr>
<tr>
<td>97% &lt; memory usage before kswapd starting to reclaim</td>
<td>Reduce page cache allocation ratio</td>
<td>• Device page</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Allocate page cache for concurrent write</td>
</tr>
</tbody>
</table>
Preliminary results (Read)

- CPU: Intel(R) Xeon(R) Gold 6338
- Available system memory: 32GB
- Optane DIMM: 126GB
- Kernel: linux-5.18.0
- Benchmark program: fio
- ioengine: mmap
- Random Read
- Block size: 4KB
- Direct is 0 (no msync after write)
Preliminary results(Write)

Autocaching shows little bit better performance than DAX because it uses DAX and page cache at the same time (Interference by flush daemons is reduced)

Autocaching shows a little bit worse performance than page cache because Autocaching uses some device memory allocated before detecting concurrent write.

- CPU: Intel(R) Xeon(R) Gold 6338
- Available system memory: 32GB
- Optane DIMM: 126GB
- Kernel: linux-5.18.0
- Benchmark program: fio
- ioengine: mmap
- Random Write
- Block size: 4MB
- Direct is 0 (no msync after write)
DONE and TO-DO list

• DONE
  • Framework
    • Initialization
  • Device page allocation
  • F/S & VFS changes to support Autocaching
  • Simple Policy

• TO-DO
  • More test with CXL storage devices for accurate and general policy
  • Dynamic struct page allocation
  • Cache transition
  • Alleviate concurrent write issue for Optane
  • Awareness of flush daemon
  • Submit rfc by Q4 for review
Linux Plumbers Conference
Dublin, Ireland September 12-14, 2022
Motivation: The advent of new types of storage device based on CXL

Dual mode support
Memory-Semantic CXLSSD
What is Autocaching?

- Use page cache for **hot** data
- Use direct access for **cold** data
Implementation of Autocaching

User Space

- mmap()
- Virtual Memory
  - Page cache (DRAM folio)
  - Direct access (Device folio)
- Create mapping
- Direct access (Device folio)
- Cold/latency sensitive data
- Direct access (Device folio)

Memory Management

- Page fault handler
- Page(Folio) allocator
- Caching allocator
  - Demotion (DRAM Folio -> Device Folio)
  - Promotion (Device Folio -> DRAM Folio)

Caching Transition

- Update VFS mapping
- Update memory mapping
- Caching or direct access

File System

- VFS internal (filemap management)
- Specific File System (EXT4)
  - filter out device folio for dirty write
  - LBA info
  - Autocaching range
  - Write Metadata
  - Device folio
  - Return (device/dram folio)
- Autocaching
  - register_autocaching()
  - autocaching_alloc_folio()
  - autocaching_free_folio()
  - find_autocaching_range()

Device Driver

- Block Layer & NVMe Device Driver
  - pmem driver
  - Detecting pmem device range
  - Create kernel direct mapping area for the detected memory range
  - Creates device pages (@ ZONE_DEVICE)

Linux Plumbers Conference | Dublin, Ireland | Sept. 12-14, 2022
Write scales poorly with concurrent writes (from Oracle)
### Virtual Memory Map

**Virtual Memory Map**

<table>
<thead>
<tr>
<th>Start addr</th>
<th>Offset</th>
<th>End addr</th>
<th>Size</th>
<th>VM area description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000000000000</td>
<td>0</td>
<td>0000000000000000</td>
<td>128 TB</td>
<td>user-space virtual memory, different per mm</td>
</tr>
<tr>
<td>0000000000000000</td>
<td>+128</td>
<td>ff7fffffff00000000</td>
<td>+16M TB</td>
<td>huge, almost 64 bits wide hole of non-canonical virtual memory addresses up to the +128 TB starting offset of kernel mappings.</td>
</tr>
</tbody>
</table>

**Kernel-space virtual memory, shared between all processes:**

<table>
<thead>
<tr>
<th>Start addr</th>
<th>Offset</th>
<th>End addr</th>
<th>Size</th>
<th>VM area description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ffff800000000000</td>
<td>-128</td>
<td>ffffffff00000000</td>
<td>8 TB</td>
<td>guard hole, also reserved for hypervisor</td>
</tr>
<tr>
<td>ffff800000000000</td>
<td>-120</td>
<td>ffffffff00000000</td>
<td>0.5 TB</td>
<td>LDT remap for PIT</td>
</tr>
<tr>
<td>ffff800000000000</td>
<td>-119.5</td>
<td>ffffffff00000000</td>
<td>64 TB</td>
<td>direct mapping of all physical memory (page_offset_base)</td>
</tr>
<tr>
<td>ffff000000000000</td>
<td>-55.5</td>
<td>ffffffff00000000</td>
<td>0.5 TB</td>
<td>unused hole</td>
</tr>
<tr>
<td>ffff000000000000</td>
<td>-55</td>
<td>ffffffff00000000</td>
<td>32 TB</td>
<td>vmmloc/loremap space (vmmloc_base)</td>
</tr>
<tr>
<td>ffffe000000000000</td>
<td>-23</td>
<td>ffffffff00000000</td>
<td>1 TB</td>
<td>unused hole</td>
</tr>
<tr>
<td>ffffe000000000000</td>
<td>-22</td>
<td>ffffffff00000000</td>
<td>1 TB</td>
<td>virtual memory map (vmmemmap_base)</td>
</tr>
<tr>
<td>ffffe000000000000</td>
<td>-21</td>
<td>ffffffff00000000</td>
<td>1 TB</td>
<td>unused hole</td>
</tr>
<tr>
<td>ffffe000000000000</td>
<td>-20</td>
<td>ffffffff00000000</td>
<td>16 TB</td>
<td>KASAN shadow memory</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Start addr</th>
<th>Offset</th>
<th>End addr</th>
<th>Size</th>
<th>VM area description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ffffc000000000000</td>
<td>-4</td>
<td>ffffffff00000000</td>
<td>2 TB</td>
<td>unused hole</td>
</tr>
<tr>
<td>ffffc000000000000</td>
<td>-2</td>
<td>ffffffff00000000</td>
<td>0.5 TB</td>
<td>vadnr_end for KASLR</td>
</tr>
<tr>
<td>ffffc000000000000</td>
<td>-1.5</td>
<td>ffffffff00000000</td>
<td>0.5 TB</td>
<td>cpu_entry_area mapping</td>
</tr>
<tr>
<td>ffffc000000000000</td>
<td>-1</td>
<td>ffffffff00000000</td>
<td>0.5 TB</td>
<td>Cexp fixup stacks</td>
</tr>
<tr>
<td>ffffc000000000000</td>
<td>-512</td>
<td>ffffffff00000000</td>
<td>444 GB</td>
<td>unused hole</td>
</tr>
<tr>
<td>ffffc000000000000</td>
<td>-68</td>
<td>ffffffff00000000</td>
<td>64 GB</td>
<td>EFI region mapping space</td>
</tr>
<tr>
<td>ffffc000000000000</td>
<td>-4</td>
<td>ffffffff00000000</td>
<td>2 GB</td>
<td>unused hole</td>
</tr>
<tr>
<td>ffffc000000000000</td>
<td>-2</td>
<td>ffffffff00000000</td>
<td>512 MB</td>
<td>kernel text mapping, mapped to physical address 0</td>
</tr>
<tr>
<td>ffffc000000000000</td>
<td>-2048</td>
<td>ffffffff00000000</td>
<td>1536 MB</td>
<td>module mapping space</td>
</tr>
<tr>
<td>ffffc000000000000</td>
<td>-1536</td>
<td>ffffffff00000000</td>
<td>1520 MB</td>
<td>module mapping space</td>
</tr>
<tr>
<td>ffffc000000000000</td>
<td>-16</td>
<td>ffffffff00000000</td>
<td>4 KB</td>
<td>legacy vsyscall ABI</td>
</tr>
<tr>
<td>ffffc000000000000</td>
<td>-10</td>
<td>ffffffff00000000</td>
<td>2 MB</td>
<td>unused hole</td>
</tr>
<tr>
<td>ffffc000000000000</td>
<td>-2</td>
<td>ffffffff00000000</td>
<td>2 MB</td>
<td>unused hole</td>
</tr>
</tbody>
</table>

12-14, 2022
Implementation of Autocaching

- Page structure is used for reverse mapping for caching transition
Implementation of Autocaching

- **Dynamic memory allocation of struct page**
  - Only active files use struct page
  - Does not used for block allocation (free page management)
  - File system allocates blocks using its block allocator.

- Challenge
  - Address conversion: page_to_pfn(), pfn_to_page()
  - Dynamically allocated pages may not be in contiguous memory.
  - VFS manages file in 4KB unit (nodes in filemap(xarray) of VFS).
Implementation of Autocaching

- **Thumbnail struct page**
  - Dynamic memory allocation of struct page
  - Reserve memory for cxl_pages (40KB for 4TB)

```
101 (*Page)  102 (PFN)  103 (*Page)  104 (PFN)

filemap nodes

struct page

struct page

105 (*Page)  106 (PFN)  107 (*Page)  108 (PFN)

filemap nodes

struct page

struct page

struct page

Dynamic allocation

LBA Range 0 - 511
```

- **DRAM**
  - Backed by pmem

- **ZONE_DEVICE's**
  - (struct page*) mem_map

- **ZONE_NORMAL's**
  - (struct page*) mem_map

- **Memory**
  - Dynamically allocated 512 struct pages
  - Page Frame Number (PFN)

- **File A**
  - Nodes & LBA

- **File B**
  - Nodes & LBA

- **CXL (2MB) pages' Index in 2MB PAGE**
  - 28 bits
  - 9 bits

- **4KB PAGE**
  - 9 bits

- **Index in 2MB PAGE**
  - CXL (2MB)

- **Page Frame Number (PFN)**

- **32KB for 512 struct pages**
  - 128M for 4TB

- **contiguous**

- **Drawing of filemap nodes**
Implementation of Autocaching

- Thumbnail struct page in detail
Implementation of Autocaching

- Address conversion from CXL page to physical address

### Address Conversion

#### Virtual File System
- **Write**
  - `copy_from_user(virt, ubuf)`
- **Read**
  - `copy_to_user(ubuf, virt)`

#### Page Fault Handler
- If(page)
  - `pfn = page_to_pfn(page)`
  - `phys = pfn_to_phys(pfn)`
  - `virt = phys_to_virt(phys)`

- **Get struct page or PFN**

- **Set PTE(phys | VMA attribute)**

### User

#### R/W

#### PAGE FAULT

- **User VMA**
  - **PGD**
  - **PUD**
  - **PMD**
  - **PTE**
  - **Update PTE**
**Implementation of Autocaching**

- Promotion

**NUMA Scan**
- Scan VM range periodically
- Check USER VM

**User**
- Increase current access count

**Node for CXL device page**
- 52 bits: PAGE identifier, Write ID, PFN
- 12 bits: Access count, 3rd prior period, 2nd prior period, 1st prior period, Current period

- If dirty/write, Store write ID = Hash(tid)

**User VM**
- `if(PTE ∈ CXL memory && ptep_test_and_clear_young())`
- Increase current access count

**VFS Filemap node**

**Step 1. Increasing Access count**

**Step 2. Promotion based on access count**

**Fast Memory (ex. DRAM)**
- An adjustable threshold (# of pages) for physical promotion
- Periodically scan flex zone for page
  1. "Access count << 3" after the scan
  2. If access count becomes 0, then demote
  3. If system has free memory, and # of page in flex zone > threshold, move some pages to DRAM
  4. If multiple threads write data directly to device, move some pages having same write ID to DRAM (for Optane)

**Priority (key is 12 bits: 4 bits + 5 bits + 3 bits)**
  1. Overflow current period (heavily accessed)
    - 8 or more accesses during the current period
    - Move pages immediately if system has enough free memory
  2. (Periodically accessed page (4 bits)
    - Check each period (current ~ 3rd prior)
  3. (Frequently accessed page (5 bits)
    - How many times has the page been accessed in the last 4 periods?
  4. (Recently accessed page (3 bits)
    - The current period has more priority than the prior periods.

**Slow Memory (ex. CXL pmem)**
- Inactive device page scan for promotion

**Scan periodically**
- Flex zone (PER)