STORAGE DEVELOPER CONFERENCE
SDC 21
BY Developers FOR Developers

Virtual Conference
September 28-29, 2021

A SNIA Event

Towards Copy-Offload in Linux NVMe

Presented by

Kanchan Joshi
Samsung Semiconductor
India Research
(SSIR)

SelvaKumar S
Samsung Semiconductor
India Research
(SSIR)

Foreword & Acknowledgement

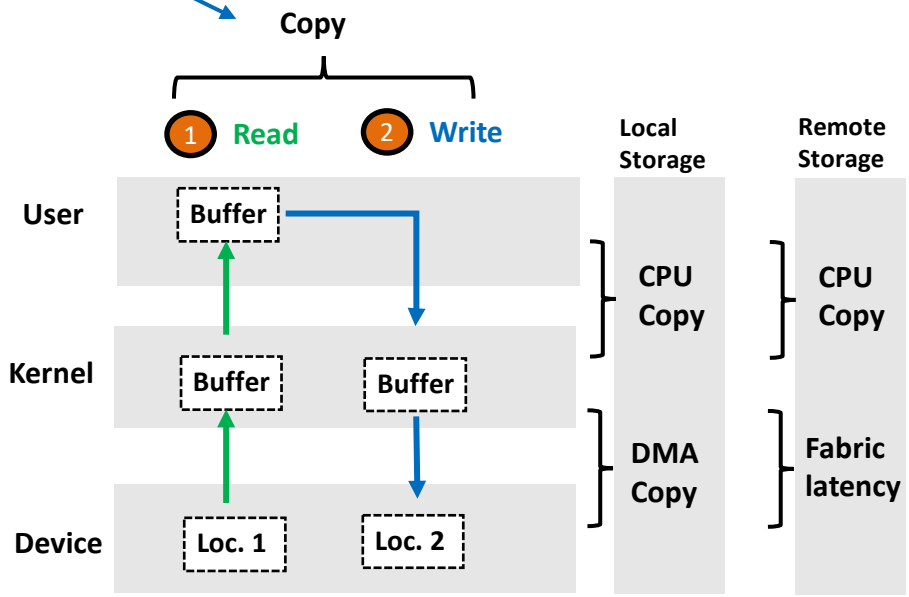
- This has elements that are under discussion in LKML
  - Mechanism, Opcode, API etc. may change in future
- The work captured here is a community effort
  - Feedback on the current plumbing have come from many developers – Damien, Bart, Derrick, Martin to name a few

Agenda

- Copy – and issues around it
- Remedial measures (OS + Storage)
- Copy-offload Interface: SCSI
- Copy-offload Interface: NVMe
- Where we are: Linux Kernel support update
- Next steps

Copying data, and costs that're out there

- Copy has traditionally been a composite operation
  - Pull from source + Push to destination
  - Perhaps the most infallible way, across heterogeneous storage backend
- Costs
  - Expensive on resources
    - Host CPU is involved, and CPU caches too
    - Host RAM is utilized; may evict other data
    - DMA resources
  - When source is same as destination, round-trip is particularly inefficient
  - Gets worse, when over fabrics/network
    - Saturates network
    - Breaks data locality; movements between storage-node and compute-node
    - The farther the storage is from application, the longer it takes for round-trip to be over



Optimizing Copy

- Async Read + Async Write ……still composite though
  - Queue multiple operations using io_uring
    - https://github.com/axboe/liburing/blob/master/examples/io_uring-cp.c
- Pushing copy to kernel
  - Application does not have to pass buffers for copying
  - Linux has a bunch of APIs for 'in-kernel' copy instead
  - Sendfile
    - Perhaps the oldest of the bunch
    - Originally introduced to copy between regular-file to socket

```
ssize_t sendfile(int out_fd, int in_fd, off_t *offset, size_t count);
```

  - Splice
    - Two step operation
    - Copy from file A to pipe (splice-read) and then pipe to file B (splice-write)
    - The kernel-infra is used for implementing *sendfile* too

```
ssize_t splice(int fd_in, off64_t *off_in, int fd_out,
               off64_t *off_out, size_t len, unsigned int flags);
```

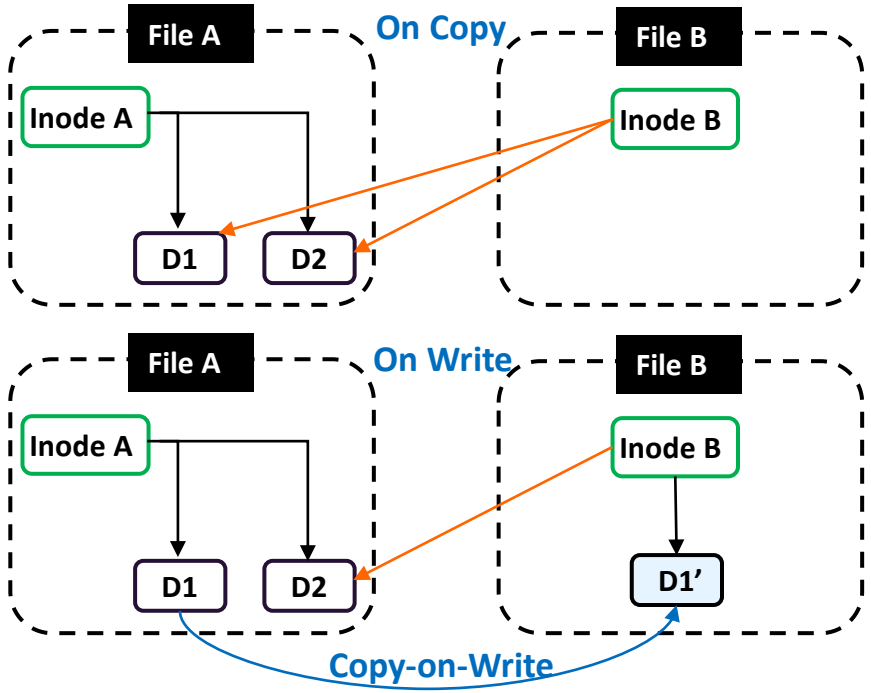
  - Copy_file_range
    - The newest of the bunch
    - Few file systems use this interface to implement custom copy-acceleration
    - Example: server-side-copy in NFS & CIFS

```
ssize_t copy_file_range(int fd_in, off64_t *off_in,
                        int fd_out, off64_t *off_out,
                        size_t len, unsigned int flags);
```


Optimizing Copy ….

- Switch to logical-copy
  - Possible when there is a higher-level construct sitting above raw data-blocks
  - Few filesystems implement logical copy by sharing data-blocks
    - Create meta but share data
    - Copy data on subsequent change
    - Essentially lazy copy!
  - BTRFS, OCFS and XFS
  - How user-space can trigger logical-copy
    - Invoke *FICLONE or FICLONERANGE* ioctl
    - 'cp' provides a knob
      - *cp –reflink=always source_file dest_file*

```
When --reflink[=always] is specified, perform a lightweight copy,
where the data blocks are copied only when modified.  If this is
not possible the copy fails, or if --reflink=auto is specified,
fall back to a standard copy.  Use --reflink=never to ensure a
standard copy is performed.
```



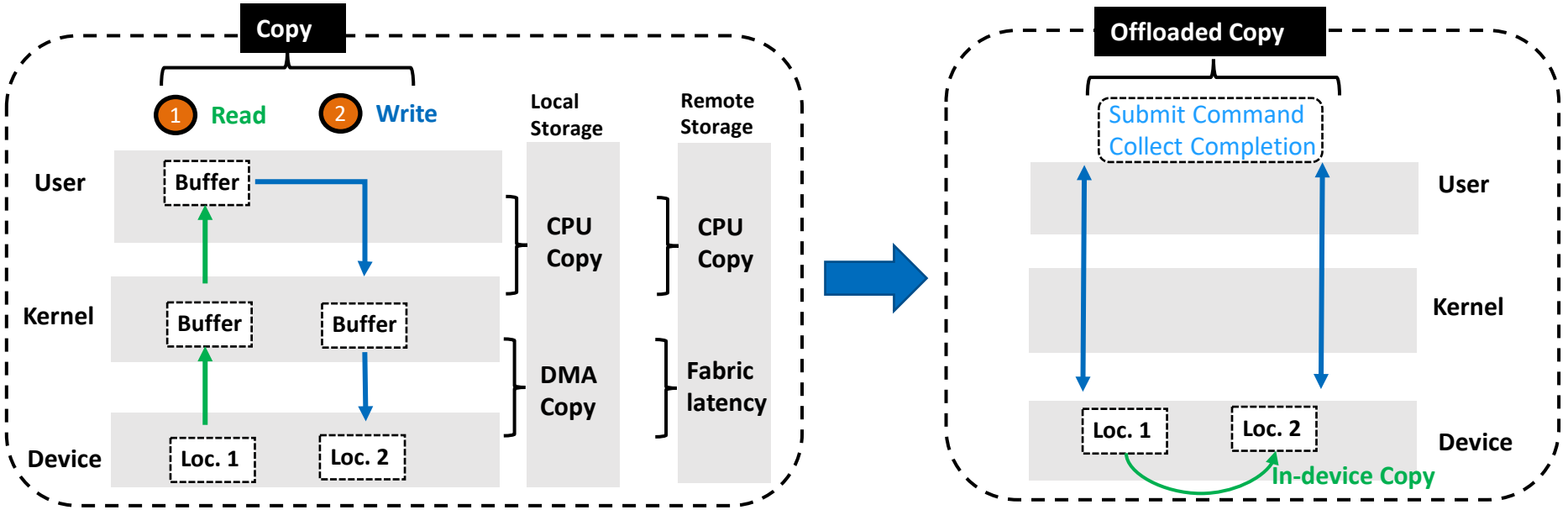

```
int ioctl(int dest_fd, FICLONERANGE, struct file_clone_range *arg);
int ioctl(int dest_fd, FICLONE, int src_fd);

struct file_clone_range {
    __s64 src_fd;
    __u64 src_offset;
    __u64 src_length;
    __u64 dest_offset;
};
```


Pushing copy further down

….to storage

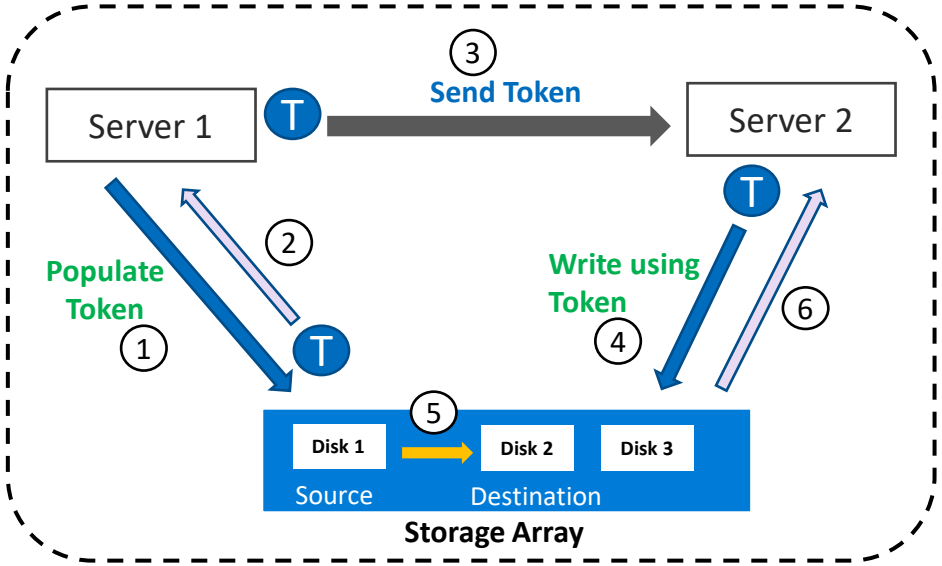
Copy-offload capability of Storage

- A dedicated ‘copy’ interface from the device itself
- Round-trip involving app/kernel/fabric elements is cut short
  - Host does control-plane activity
  - Device does data-plane activity



SCSI: Copy-Offload

- At least a decade old; copy across multiple devices
- Two main variants
  - Extended Copy (XCOPY)
    - Block-ranges describing copy-operation are sent either to source or destination
  - Token Based Copy/ODX
    - Obtain cookie from source device using POPULATE TOKEN
    - Send cookie to destination device using WRITE USING TOKEN



SCSI: Copy-Offload

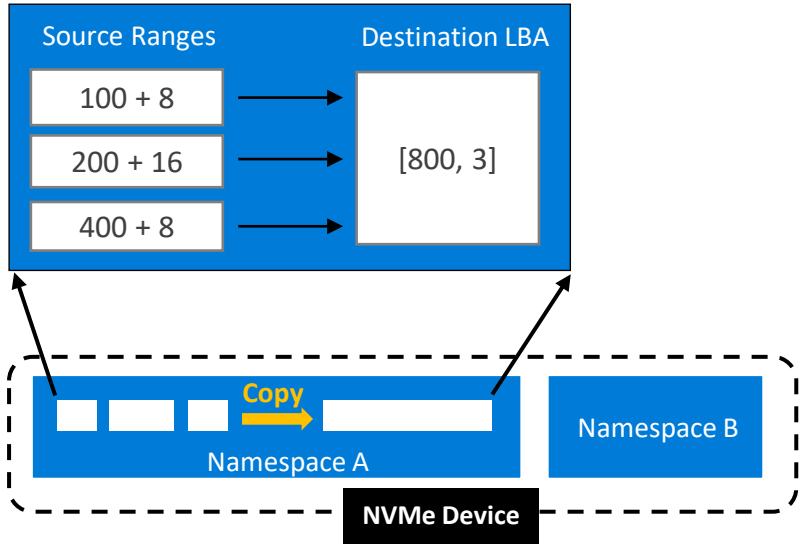
- Kernel support
  - Remained elusive, despite multiple attempts
  - Plumbing efforts in past
    - Martin Petersen, 2014, https://www.mail-archive.com/linux-scsi@vger.kernel.org/msg28998.html
    - Mikulas Patocka, 2014, https://www.mail-archive.com/linux-kernel@vger.kernel.org/msg686111.html
  - Summary
    - An IOCTL exposing copy between single source-range and single destination-range
    - Block layer to SCSI: Two bios, one with COPY_READ another with COPY_WRITE
    - XCOPY issued when both COPY_READ and COPY_WRITE reach to driver without getting split
- Why this's not upstream yet
  - Answer of Martin Petersen (SCSI maintainer): http://mkp.net/pubs/xcopy.pdf
  - Copy operations fails if a copy request ever needs to be split as it traverse the stack preventing working in almost every common deployment configuration
  - Storage stack need to switch away from the iterative stacking approach……this has not happened, not yet!

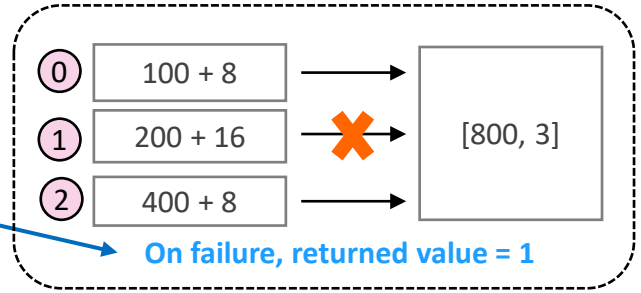
NVMe Interface for Offload

…copy command

NVMe Copy Command

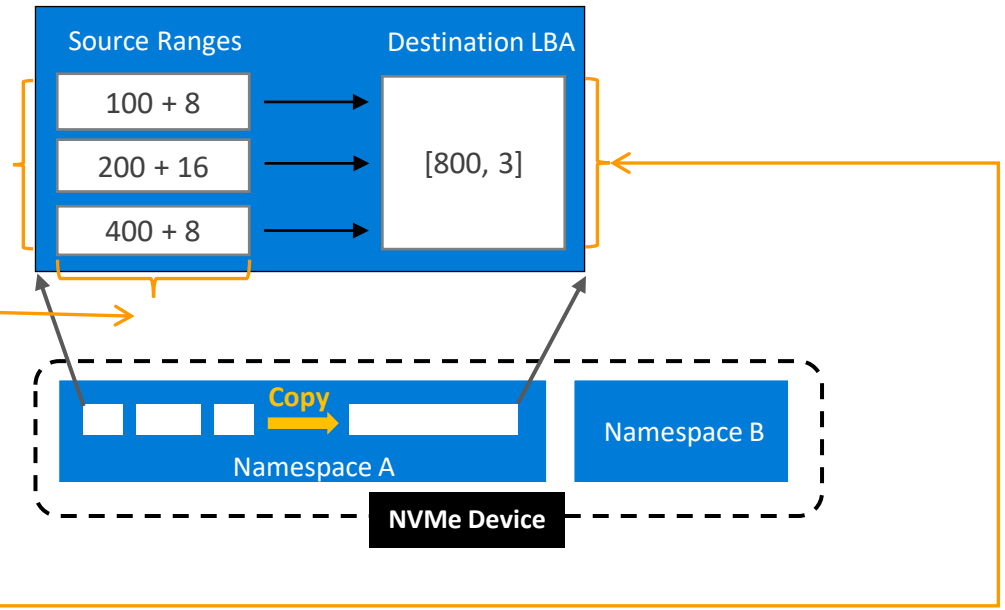
- XCopy turned out to be “complex” command
  - Multitude of options for copy; within LUN, across LUNs (intra array and inter array too)
- NVMe chose “Simple” Copy Command. Simple because scope is within the single namespace
- Single command to copy multiple source LBA ranges to a single destination LBA
  - Each source range is a combination of source LBA offset and length
  - Source ranges are copied in same order
- On command failure
  - Return lowest numbered Source Range entry that was not successfully copied.





NVMe Copy Command

- Number of source ranges in a single copy command is limited by **MSRC**
- Maximum length of a single source range is limited by **MSSRL**
- Overall copy size of single SCC command is limited by **MCL**



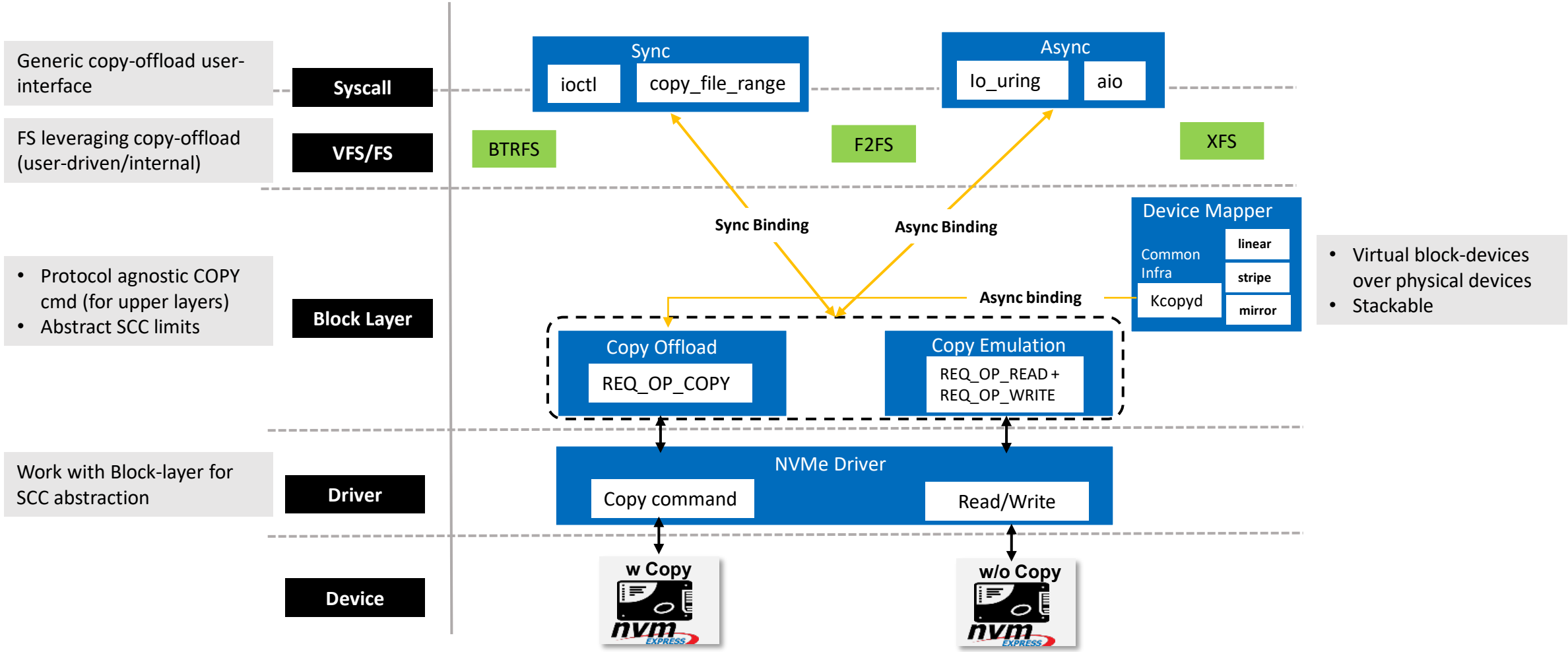
Why now?

- Existing landscape
  - Use cases for in-device-copy have become more relevant
    - Very large SSDs (even without QLC)
    - Emergence of ZNS, requiring host-side garbage-collection
  - NVMe & NVMeOF is widely adopted as storage & networking protocol
    - Disaggregates setups (compute node separate from storage nodes)
  - High-performance HW; while CPUs are not getting faster (https://riscv.org/wp-content/uploads/2018/12/A-New-Golden-Age-for-Computer-Architecture-History-Challenges-and-Opportunities-David-Patterson-.pdf)
    - Single thread performance: stagnant due to Denard Scaling
    - Multi-thread performance: slowing down of Moore's law
- Usecases
  - Host-Side Garbage-Collection
    - ZNS command set proposes zone-abstraction
    - Once full, zone need to be explicitly 'reset' before it can be reused
    - Before reset, host may need to gather valid data of zone(s) and copy that out to free zone
  - Can be useful for log-structured FS/DBs sitting over CNS too
  - Defragmentation
    - FS may develop aging/fragmentation over time
    - With in-device copy, defragmentation process can be kept confined to device

Plumbing scheme in Linux Kernel

…..work-in-progress

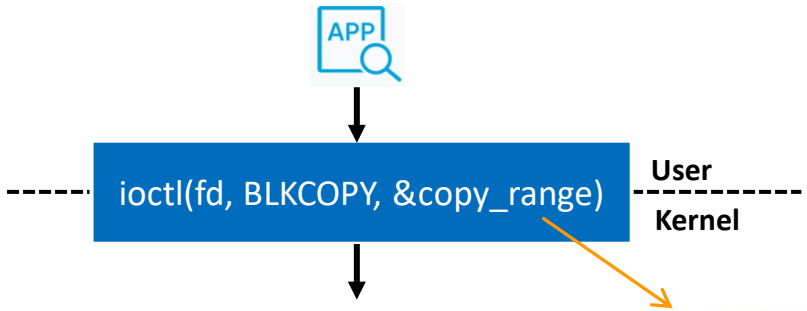
Generic copy-offload components



Source-code and discussions: https://lore.kernel.org/linux-nvme/20210817101423.12367-1-selvakuma.s1@samsung.com/

User-interface

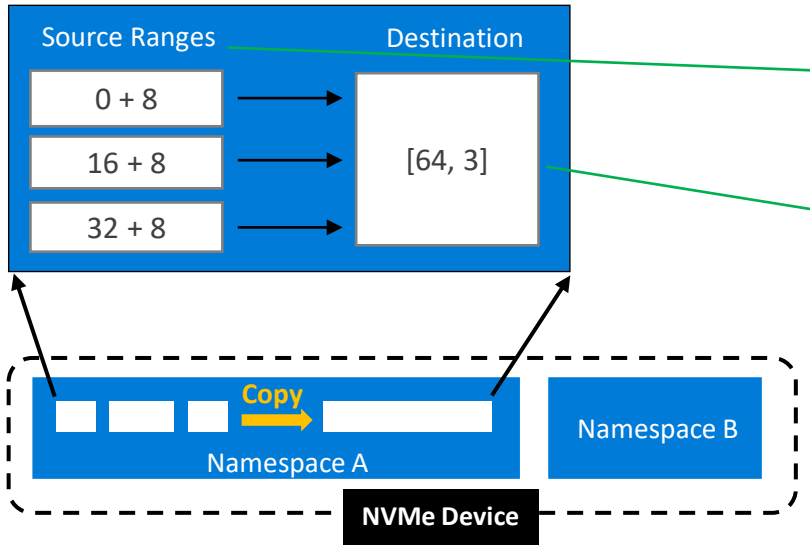
- Current Scheme
  - Existing copy syscalls do not accept a cluster of source locations
  - New BLKCOPY ioctl carrying a payload over raw block device
- In future
  - Expose async interface via io_uring and/or linux aio
  - copy_file_range for FS and raw-block dev




```
struct copy_range {
        __u64 dest;
        __u64 nr_range;
        __u64 range_list;
        __u64 rsvd;
};

/* Source range entry for copy */
struct range_entry {
        __u64 src;
        __u64 len;
};
```


BLKCOPY ioctl

- Example: copy three source ranges to single destination within a namespace




```
int main(void)
{
        int ret, fd;
        struct range_entry source_range[] = {{.src = 0, .len = 8},
                {.src = 16, .len = 8}, {.src = 32, .len = 8},};
        struct copy_range cr;

        cr.dest = 64;
        cr.nr_range = 3;
        cr.range_list = (__u64)&source_range;

        fd = open("/dev/nvme0n1", O_RDWR);
        if (fd < 0) return 1;

        ret = ioctl(fd, BLKCOPY, &cr);
        if (ret < 0) printf("copy failure\n");

        close(fd);

        return ret;
}
```


Copy: Block-layer & NVMe driver

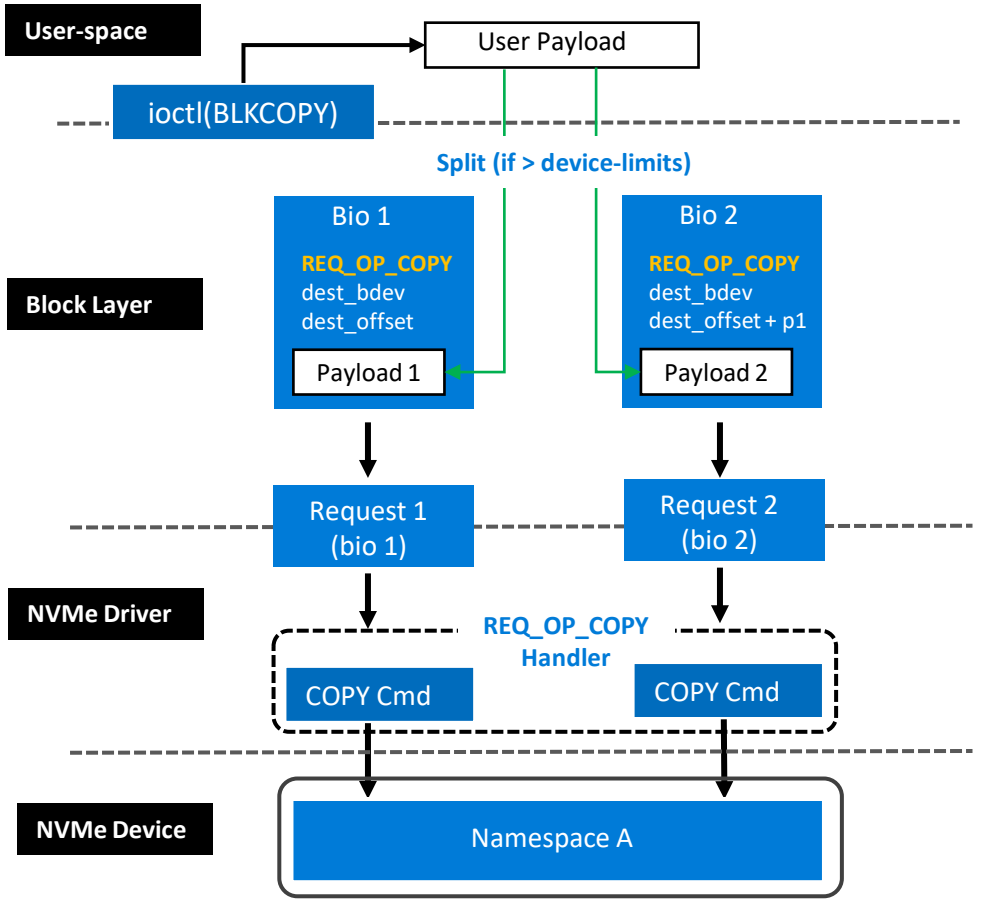
- Generic Copy interface
  - Block-Layer/Driver work together to abstract device details
  - Expose protocol-agonistic REQ_OP_COPY to upper layers (FS, user etc.)
    - Provide sync or async completion, depending on the caller
  - Hide device limits, may impose kernel-defined limits

- Copy emulation
  - When underlying device not support copy-offload interface
  - Implemented by using regular read and write

Copy: Block/Driver operation sequence

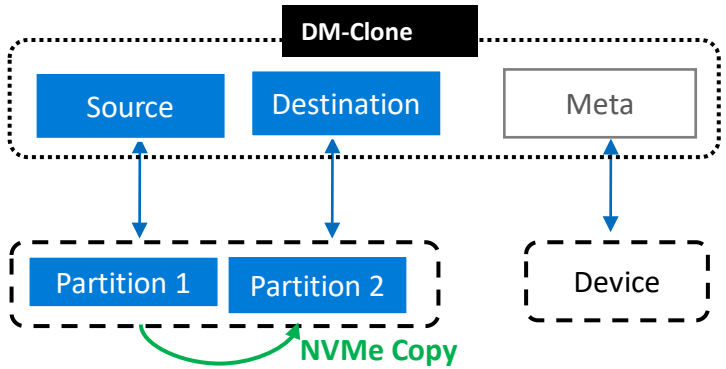
- Process user-payload: validity checks, remapping in case of partitioned device, split if larger than limits
- Form another payload (one-to-many)
- Encapsulate each payload into bio with opcode REQ_OP_COPY and REQ_NOMERGE flag
- Bio, packed into request, travels down
- Post all submissions, caller is notified (either sync or async fashion)

- Converts block-layer payload to NVMe format (sector-to-lba conversion)
- Forms Copy command and dispatches to Device



In-Kernel user: dm-kcopyd

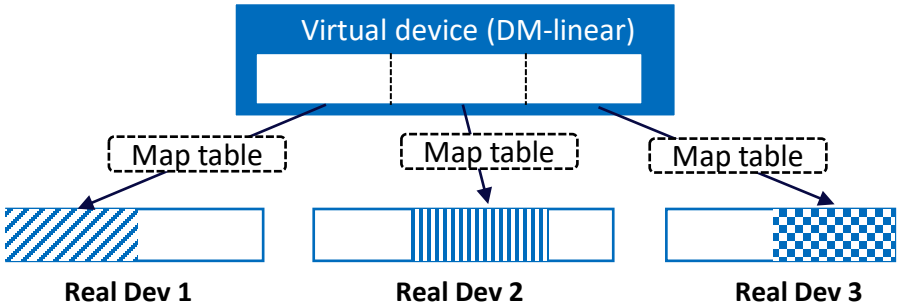
- What is dm-kcopyd
  - Kernel daemon to copy (read+write) from one block-device to one/more block devices
  - Part of the device-mapper infra; used by other device-mappers
- Enabling copy-offload
  - dm_kcopyd_copy() plumbing
    - Switch to offload if both source and destination dev are on single underlying namespace supporting COPY command
- Example: dm-clone
  - one-to-one copy of source-device into destination-device
  - Hydration: trigger copying of ranges




```
# to create clone setup with three partions within a namespace
dmsetup create clone --table "0 2097152 clone /dev/nvme0n1p3 /dev/nvme0n1p2 /dev/nvme0n1p1 8 1 no_hydration"

#to trigger hydration
dmsetup message clone 0 enable_hydration
```


Device-Mapper: challenges

- What is device-mapper
  - Subsystem to create virtual block-device on top of real ones
  - Implement the functionality not present in the underlying device: concatenation, striping, encryption, snapshot etc.
  - Stackable – virtual device over virtual
  - Remap the IO on virtual device to underlying ones
    - Read/write bio is split/remapped as it travels down
- Challenges with copy-offload
  - Defining semantics of copy-operation across various DMs
  - Virtual source/dest device may contain N other underlying device
  - Copy operation needs to be made composite (Read + Write) for propagation
    - Scatter copy into multiple “read + write” at block layer
    - Gather at NVMe driver to form SCC commands



Next steps

- There are many, but top few are -
  - Device-mapper offload support (either have it wired up, or get the consensus on moving without it)
  - Async interface for copy-offload via io_uring
  - Copy offload support in file systems

Please take a moment to rate this session.

Your feedback is important to us.

