Host-Managed Flash

Flash-Aware Computing

Instructor:

Prof. Sungjin Lee (sungjin.lee@dgist.ac.kr)

1

Today

Why Host-Managed Flash?

- SDF: Software-Defined Flash
- AMF: Application-Managed Flash
- LightNVM: The Linux Open-Channel SSD Subsystem
- Reference

Why Host-Managed Flash?

An emerging approach to managing NAND flash, especially in data-center environments

Why?

- **1.** Changes in data management algorithms
 - Log-structured/COW file systems and LSM-Tree become popular
- 2. Changes in I/O access patterns
 - I/O access patterns become mostly sequential
- **3.** Changes in storage media
 - Low latency storage media (e.g., Flash and 3D-XPoint)

Changes in Data Mgmt. Algorithms

- Modern web services rely upon two types of workloads for small objects
 - Interactive workloads: "in-place update" storage optimized for random reads and worst case write latencies
 - B-Tree or B+ Tree algorithms
 - Analytical workloads: "out-of-place update" storage optimized for high write throughput and sequential reads
 - LSM-tree algorithms



Changes in Data Mgmt. Algorithms (Cont.)

Often use two different storage management policies

- Fast-path processing tasks with "in-place update" storage;
- Asynchronous analytical tasks with "out-of-place update" storage
- Limitations
 - Take several hours for ML models to react to users' behavior
 - Force operators to manage redundant infrastructure
- Hyperscale companies (e.g., Google, Yahoo, Facebook, ...) prefer the "out-of-place update" policy
 - Writes account for a larger portion of the total I/Os
 - For interactive workloads, 10-20% in 2010 \rightarrow 50% in 2012
 - Better algorithms for LSM-Tree
 - e.g., Bloom filter, key sorting, better merge scheduling, ...

Changes in I/O Access Patterns

- **LSM-tree algorithm is based on an append-only write policy**
- Writes sent to storage are almost sequential



Suitable for append-only storage like Flash and SMR

Changes in Storage Media

Existing storage stacks are so complicated!!!



- Okay for slow HDDs with *millisecond* latencies
- Not okay for fast non-volatile memories with *microsecond* latencies (e.g., flash and PRAM) → Software overheads are becoming a new bottleneck

Today

- Write-optimized Storage Systems
- SDF: Software-Defined Flash
- AMF: Application-Managed Flash
- LightNVM: The Linux Open-Channel SSD Subsystem
- Reference

I/O Characteristics in Data Center

- In data canters, small-write traffic to a storage system is managed by a log-structured merge (LSM) tree
 - e.g., Google's BigTable, Baidu's CCDB, Yahoo's PNUTS, HBase, Cassandra, LevelDB, SQLite, MongoDB 3.0, ...
- With LSM, almost all of the write requests become sequential only with few random writes
- Commodity SSDs are not designed to effectively support or to take advantage of such new workloads
 - In terms of I/O performance, capacity, and costs

Problems with Commodity SSDs

I/O performance

- The effective bandwidth received at the storage system serving real world workloads can be much lower
- I/O requests from upper-level software (e.g., file systems and databases) are not optimized to generate SSD-friendly access patterns

Space efficiency

 A large amount of flash space is reserved for overprovisioning and is not available for storing user data

Costs

- With hundreds of thousands of servers, commodity SSDs have severe implications, in terms of both initial and recurring costs
- Increase hardware costs, energy costs, physical space requirement, and maintenance costs

Over-provisioning Ratio

 Throughput of SSDs (Intel 320) when various fractions of its raw capacity are allocated as over-provisioning space



SDF: Software-Defined Flash

- Motivated by a significant workload characteristic in data centers supporting Internet-wide services
- Redesign SSDs and I/O stacks in a data center with its particular workload characteristics

SDF's approaches

- (1) Exposing the channels/ways/blocks in SSD hardware to software
 - Get rid of upper-level software
- (2) Host software to explicitly manage SSD hardware
 - Better understand host workloads
 - Make a device simpler
- (3) No reserve space for internal GC
 - All raw flash space available for storing user data

SDF Hardware

- Virtex-5: PCIe DMA and chip-to-chip bridging
 - Expose all the channels to the host transparently
- **Spartan-6**: Independent FTL for each of 11 channels
 - Perform block-level mapping, dynamic wear-leveling, and bad block management



SDF Hardware (Cont.)

 SDF's hardware includes features and functionalities that are proven truly necessary in web-scale storage environments

(1) No static wear leveling

- Mainly used as a cache for HDDs data that is rarely accessed is not expected to reside in the cache for a long time
 - \rightarrow Simplify HW design and reduce I/O variations

(2) No DRAM for caching

- Data is cached in a host DRAM a storage device is not expected to have strong temporal locality
- Write requests are acknowledged only after the data is stored on flash
 No or less amount of DRAM and no battery/capacitor

SDF Hardware (Cont.)

(3) No parity-based data protection

- Use software-managed replication which can be done by the host software
- Only BCH ECC for flash error correction

 \rightarrow Simplify HW design and reduce I/O variations

(4) No over-provisioning area

 Since software manages SSD directly (Baidu's CCDB), there is no need for GC inside SSD

 \rightarrow 20% to 50% cost reduction

SDF Interface

Expose the asymmetry in the read/write/erase operations to the software

- The block erasure is explicitly done by the software
- The read unit is kept small (e.g., one or a few flash pages)
- The write unit size is set a multiple of the flash erase block size, and write addresses are required to be block-aligned
 - Valid and invalid pages do not coexist in the same block

Expose the device's internal parallelism to the software

- Each channel is exposed to the applications as an independent device
 - e.g., /dev/sda0 through /dev/sda43

SDF Interface (Cont.)



SDF I/O Stack

 Provide a unified user-space block layer to the applications, bypassing almost all of kernel modules (e.g., file system, block layer, and so on)



SDF Data Management Systems

- Various data management applications, file system, database, and key-value store, run atop multiple *slices*
- Slices are based on LSM-tree algorithms and talk with SDF hardware directly



Experimental Results

Latencies of write requests



I/O Throughputs



Today

- Write-optimized Storage Systems
- SDF: Software-Defined Flash
- AMF: Application-Managed Flash
- LightNVM: The Linux Open-Channel SSD Subsystem
- Reference

FTL is a Complex Piece of Software

 FTL runs complicated firmware algorithms to avoid in-place updates and manages unreliable NAND substrates



But, FTL is *a root of evil* in terms of HW resources and performance

- Requires significant hardware resources (e.g., 4 CPUs / I-4 GB DRAM)
- Incurs extra I/Os for flash management (e.g., GC)
- Badly affects the behaviors of host applications

However,

Functionality of FTL is Mostly Useless

 Many host applications manage underlying storage in *a log-like manner*, mostly avoiding in-place updates



This duplicate management not only (1) incurs serious performance penalties but also (2) wastes hardware resources

Which Applications???

File Systems **BlueSky SpriteLFS** WAFL **NILFS** F2FS HDFS Btrfs **Key-value Stores** Dat<u>abases</u> **LevelDB RethinkDB** Which **RocksDB Applications** Cassandra **BigTable** ??? LSM-Tree LogBase **MongoDB** Hyder

What if we removed FTL from storage devices and allowed applications to directly manage NAND flash?

Application-Managed Flash (AMF)



(3) A new AMF block I/O abstraction enables us to separate the roles of the host and the device

AMF Block I/O Abstraction (AMF I/O)

• AMF I/O is similar to a conventional block I/O interface

 A linear array of fixed-size sectors (e.g., 4 KB) with existing I/O primitives (e.g., READ and WRITE)



Append-only Segment

Segment: a group of 4 KB sectors (e.g., several MB)

- A unit of free-space *allocation* and free-space *reclamation*
- Append-only: overwrite of data is prohibited



Case Study with File System



AMF Log-structured File System (ALFS)

- ALFS is based on the F2FS file system
- How did we modify F2FS for ALFS?
 - Eliminate in-place updates
 - F2FS overwrites check-points and inode-map blocks
 - Change the TRIM policy
 - TRIM is issued to individual sectors

How many new codes were added?



<A comparison of source-code lines of F2FS and ALFS>



Check-point and inode-map blocks are overwritten





The FTL appends incoming data to NAND flash



The FTL triggers garbage collection: 4 page copies and 4 block erasures



The LFS triggers garbage collection: 3 page copies

How ALFS Works



How ALFS Works



No obsolete pages – GC is not necessary

How ALFS Works



The ALFS triggers garbage collection: 3 page copies and 2 block erasures

Comparison of F2FS and AMF



Performance with FIO



- For random writes, AMF shows better throughput
- F2FS is badly affected by the duplicate management problem

Performance with Databases



- AMF outperforms EXT4 with more advanced GC policies
- F2FS shows the worst performance

Resource (DRAM & CPU)

FTL mapping table size

SSD Capacity	Block-level FTL	Hybrid FTL	Page-level FTL	AMF
512 GB	4 MB	96 MB	512 MB	4 MB
1 TB	8 MB	186 MB	1 GB	8 MB

Host CPU usage



Today

- Write-optimized Storage Systems
- SDF: Software-Defined Flash
- AMF: Application-Managed Flash
- LightNVM: The Linux Open-Channel SSD Subsystem
- Reference

Open-Channel SSDs

Open-channel SSDs are emerging on the market

- Excellent platform for addressing SSD shortcomings and managing trade-offs related to throughput, latency, power, and capacity
- Examples:
 - Fusion-IO and Violin Memory: a storage stack that manages NAND media and provides a block I/O interface
 - Baidu's SDF: a key-value store integrated with underlying storage media
 - AMF: a new SSD interface, compatible with the legacy block device interface, exposing error-free and append-only segments

The integration of open-channel SSDs into the storage infrastructure has been limited

 A single point in the design space is explored with a fixed collection of trade-offs

Design of LightNVM

 LightNVM is the first open, generic subsystem for Open-Channel SSDs and host-based SSD management



Design of LightNVM

 LightNVM is the first open, generic subsystem for Open-Channel SSDs and host-based SSD management



Use Case #1: Block I/O Device

- The Physical Block Device (*pblk*) is a fully associative, host-based FTL that exposes a block I/O interface
 - Dealing with controller- and media-specific constraints (e.g., write buffering)
 - Flushes forces *pblk*'s in-flight data to the device
 - Map logical addresses onto physical addresses
 - Garbage collection
 - Handling error
- *pblk* allows us to run existing apps and file systems on top of Openchannel SSDs



Use Case #2: Application-specific SSD

- Exposes flash geometry to the applications, which allows us to implement SSD-optimized applications
- Target applications
 - Key-value store: RocksDB optimized for LightNVM is under development by CNEXLab
 - Application-specific FTL implementation



Use Case #2: Application-specific SSD (Cont.)

RocksDB with LightNVM



LightNVM Address Space

- Expose to the host a collection of channels, each containing a set of *Parallel Units* (PUs) (a NAND die)
- Each PU is decomposed into either (*blocks*, *plane*, *page*, *sector*) or (*block*, *sector*)

Logical Block Address (LBA)

MSB



64 bit





LSB

Vector I/O

- Obtain higher throughput through parallel units
- Large overhead if I/Os is separately issued
- Introduce vector I/O interface to enable host to submit I/Os to multiple PUs using one command
- Vector Read/Write/Erase using scatter/gather address list



Write Buffering – Host-side

(1) Host-side write buffering

- Sector writes (4 KB) are buffered until enough data is gathered to fill a flash page (e.g., 16-32 KB); paired pages must be handled similarly
- In case of a flush, add padding in the command
- (+) Avoid interference between the host and devices
- (+) Acknowledged as reads hit the cache
- (-) The contents loss in case of a power failure

\rightarrow The current implementation of LightNVM

Write Buffering – Device-side

(2) Device-side write buffering

- Sector writes are buffered on the device side
- (+) High durability with on-device battery or capacitor
- (-) Unpredictability (interfere with host reads)

Future direction

(3) Device-side write buffering with Controller Memory Buffer (CMB)

- Sector writes are buffered on the device, but *explicitly* flushed by the host
- (+) High durability and predictability
- (-) Complicated implementation

Performance

Latencies for OLTP and OLAP on NVMe SSD and OCSSD



Read latency comparison (4KB random reads / 64 KB writes)



Today

- Write-optimized Storage Systems
- SDF: Software-Defined Flash
- AMF: Application-Managed Flash
- LightNVM: The Linux Open-Channel SSD Subsystem
- Reference

Reference

- Sears, R., & Ramakrishnan, R. (2012, May). bLSM: a general purpose log structured merge tree. In Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (pp. 217-228). ACM.
- O'Neil, P., Cheng, E., Gawlick, D., & O'Neil, E. (1996). The log-structured merge-tree (LSM-tree). Acta Informatica, 33(4), 351-385.
- Facebook, RocksDB, <u>http://rocksdb.org/</u>
- Ouyang, Jian, et al. "SDF: Software-defined flash for web-scale internet storage systems." ACM SIGPLAN Notices 49.4 (2014): 471-484.
- Lee, S., Kim, J., & Mithal, A. (2015). Refactored design of i/o architecture for flash storage. *IEEE Computer Architecture Letters*, *14*(1), 70-74.
- Lee, S., Liu, M., Jun, S. W., Xu, S., Kim, J., & Arvind, A. (2016, February). Application-Managed Flash. In *FAST* (pp. 339-353).
- Bjørling, Matias, Javier Gonzalez, and Philippe Bonnet. "LightNVM: The Linux Open-Channel SSD Subsystem." USENIX FAST 2017
- González, Javier, et al. "Application-Driven Flash Translation Layers on Open-Channel SSDs." NVMSA 2017