CAFTL: A Content-Aware Flash Translation Layer Enhancing the Lifespan of Flash Memory based Solid State Drives

Feng Chen[1] Tian Luo[2] Xiaodong Zhang[2]

[1] Circuits and Systems Research

Intel Labs

feng.a.chen@intel.com

[2] Dept. Of Computer Science & Eng.

The Ohio State University

{luot,zhang}@cse.ohio-state.edu

Flash Memory based Solid State Drives

Solid State Drive (SSD)

- A semiconductor device built on NAND flash memory
- Mechanical components free

Technical merits

- High performance (e.g. 250MB/sec, 75µs)
- Low power consumption (e.g. 0.06~2w)
- Shock resistance
- Decreasing price (e.g. $150 for 32GB)

A wide scope of usage

- Mobile computers (e.g. Asus EeePC, Dell Inspiron Mini)
- High-performance desktops (e.g. gaming machines)

Limited lifespan–
Achilles' heel of Solid State Drives

Limited program/erase (P/E) cycles of flash memory

- Multi-level Cell (MLC) – 5,000 ~ 10,000
- Single-level Cell (SLC) – 100,000 ~ 1,000,000

Limited lifespan of SSDs

- Naturally limited by the lifetime constraint of flash memory
- Most prior research work focused on wear-leveling techniques*
- SSD manufacturers – *SSDs can sustain "routine usages" for years*

* Gale and Toledo, "Algorithms and data structures for flash memories", ACM Computing Survey, 2005, vol. 37(2), pp. 138-163

SSD Endurance Remains a Serious Concern

Technical trend of flash memory

- Bit density increases → price decreases, endurance decreases
- Sharp drop of program/erase cycles from 10,000 to 5,000 [Anderson'10]

Redundancy-based solution (e.g. RAID) is less effective

- RAID solutions (e.g. 0,1,5) evenly distribute accesses across devices
- High risk of correlated device failures in SSD-based RAID [Balakrishnan'10]

Limited public info on SSD endurance in the field

- Both positive/neg. results reported in prior work [Boboila'10, Grupp'09, Mohan'10]
- "Endurance and retention (of SSDs) not yet proven in the field" [Barroso'10]

Commercial systems are sensitive to reliability issues

- Undergoes highly intensive write traffic than client systems
- Permanent data loss is unacceptable (e.g. financial systems)

SSD endurance remains a serious issue, and solutions effectively enhancing the lifespan of SSDs is highly desirable in practice

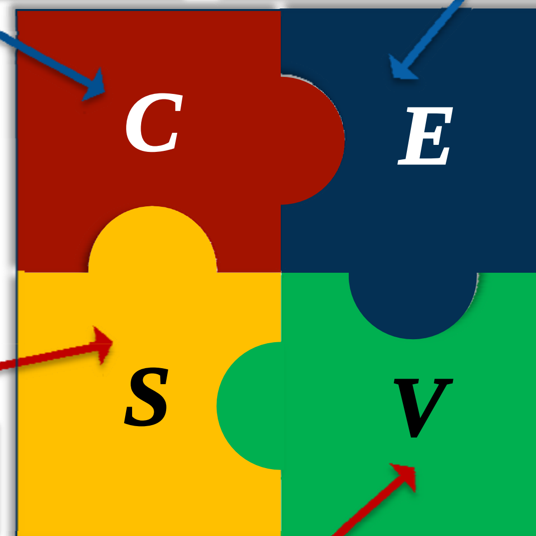
Lifespan of Solid State Drives

Limited lifespan of SSDs

- ***C*** – program/erase Cycles
- ***E*** – Efficiency of FTL designs
- ***V*** – write Volume per day
- ***S*** – available flash memory Space

Limited by flash memory technology

Wear-leveling/GC Techniques*



Designated during manufacturing time

Endurance = $(C \times S)/(V \times E)$

Determined by usage model and workload property

Optimization factors

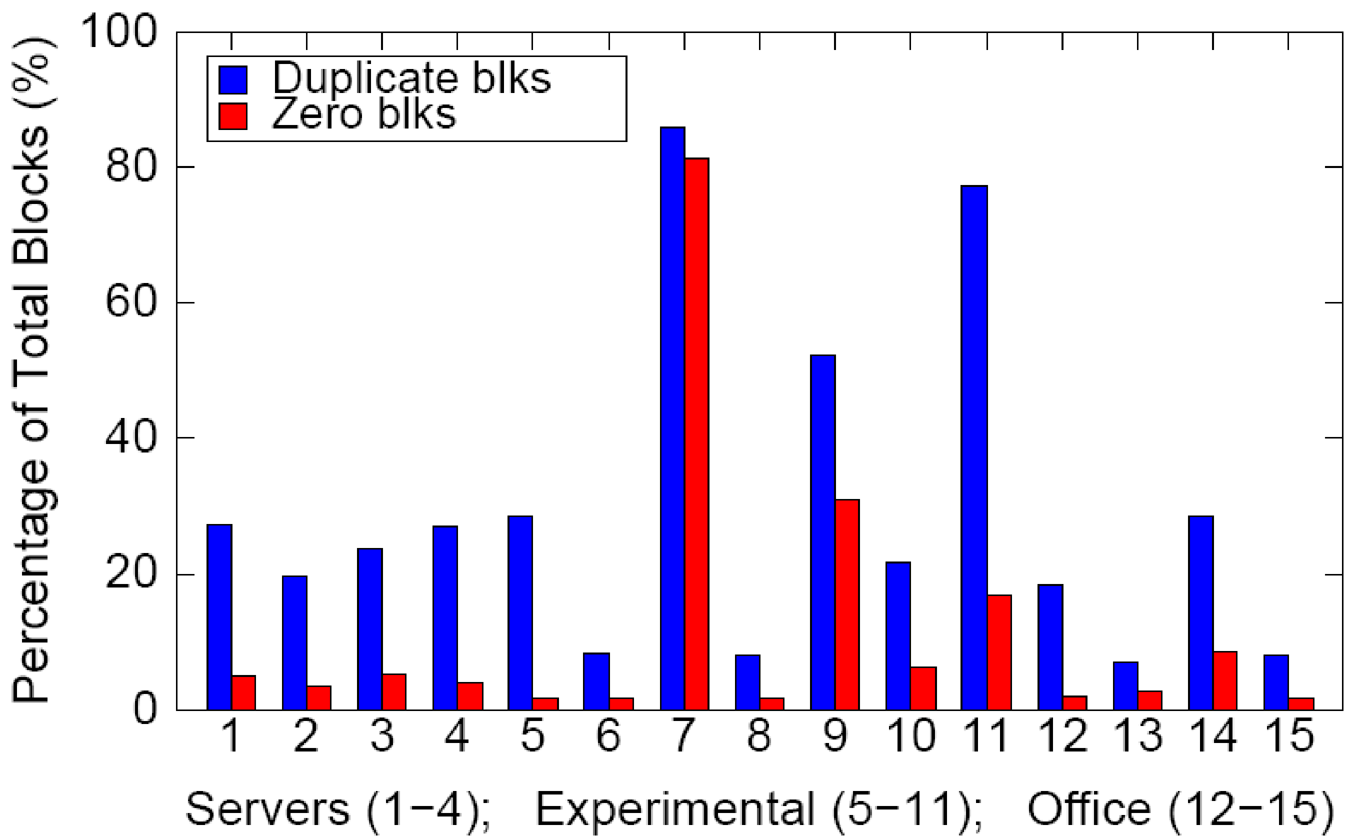
- C – Increasing P/E cycles of flash
- E – Improving efficiency of FTL designs, e.g. GC and wear-leveling
- V – reducing the amount of incoming write traffic
- S – increasing the size of over-provisioned space (e.g. 6~25%)

In this talk, we will show this goal can be achieved based on our observation of a widely existing phenomenon – *data duplication*

Data Duplication is Common

Data redundancy in storage

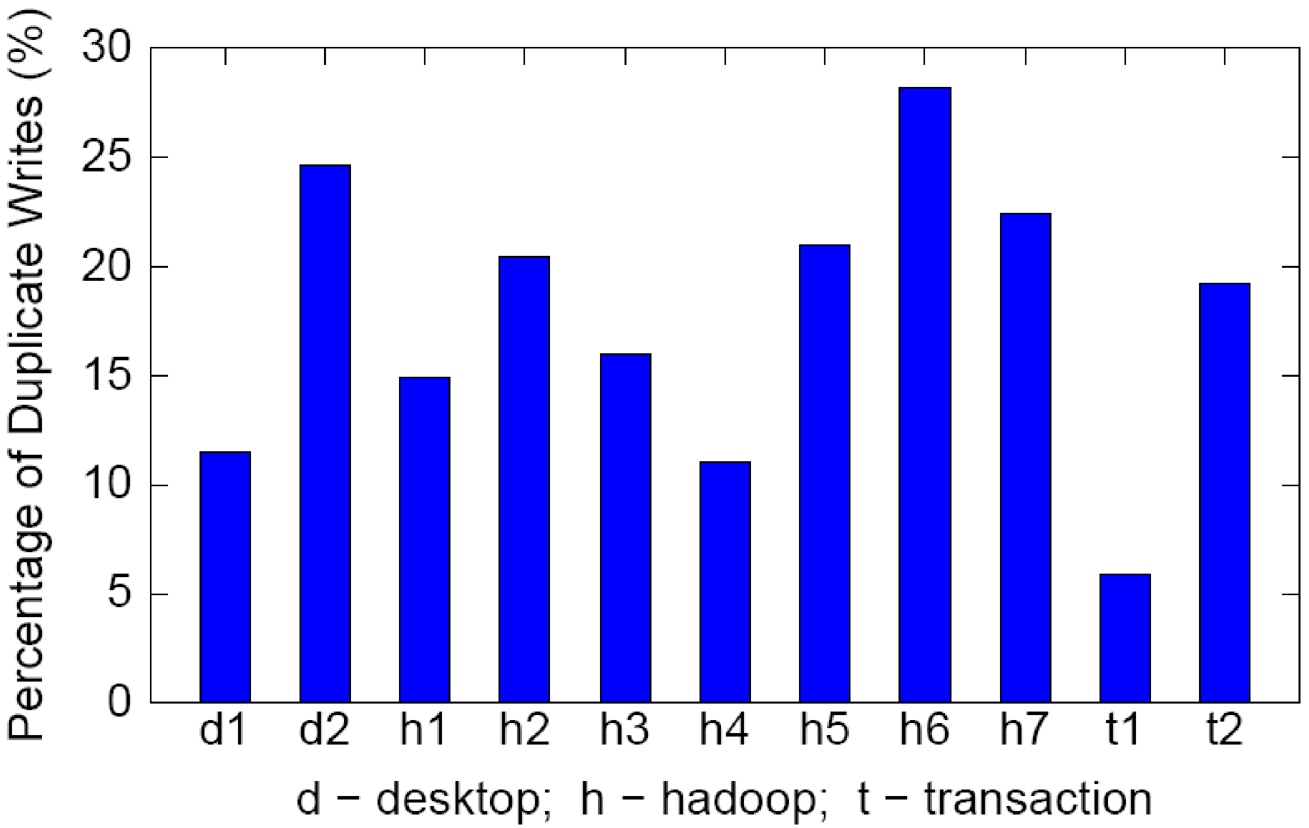
- Duplicate data rate – up to 85.9% over 15 disks in CSE/OSU
- A good extension to over-provisioned space (only 6~25%)



Data Duplication is Common

Write redundancy in workloads

- Duplicate writes – 5.8 ~ 28.1% in 11 workloads



Making FTL Content Aware

Flash Translation Layer (FTL)

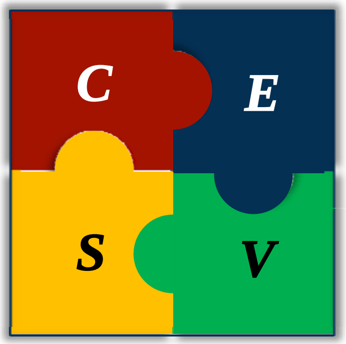
- Emulating a hard drive with an LBA interface at the device level

Content-aware Flash Translation Layer (CAFTL)

- Eliminating duplicate writes
- Coalescing redundant data

Potential benefits

- Removing duplicate writes into flash memory → ***reducing V***
- Extending available flash memory space → ***increasing S***



Technical Challenges

Information constraint

- Block-level information only → no file-level semantic hints can be used

Resource constraint

- Limited on-device resource → resource usage must be minimized

Workload constraint

- Regular file system workloads → relatively low duplication level

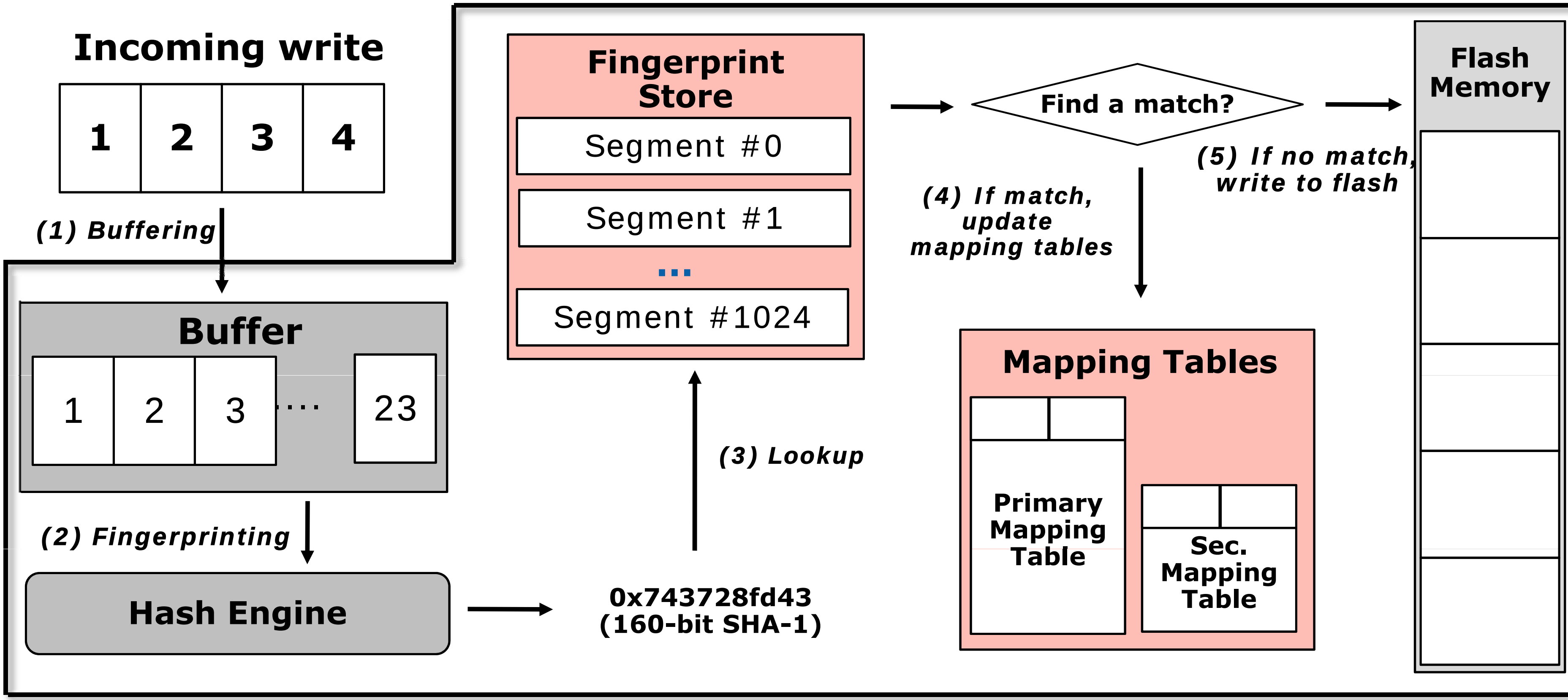
Overhead constraint

- Stringent requirement on runtime latencies → high access performance

Outline

- Introduction
- Hashing and Fingerprint Store
- Indirect mapping
- Acceleration methods
- Evaluation
- Conclusion

Overview of CAFTL



An incoming write arrives ...

- Dirty data is temporarily cached in an on-device buffer
- Computing a SHA-1 hash value (fingerprint) for each page
- Lookup against a fingerprint store to search for a match
- If a match is found → update the mapping tables, drop the write
- If no match is found → dispatch the write to flash memory

Outline

- Introduction
- Hashing and Fingerprint Store
- Indirect mapping
- Acceleration methods
- Evaluation
- Conclusion

Hash Function and Fingerprints

Fixed-sized chunking

- Basic hash unit size – a flash page (e.g. 4KB)

A cryptographic hash function

- SHA-1 hash function – low collision probability

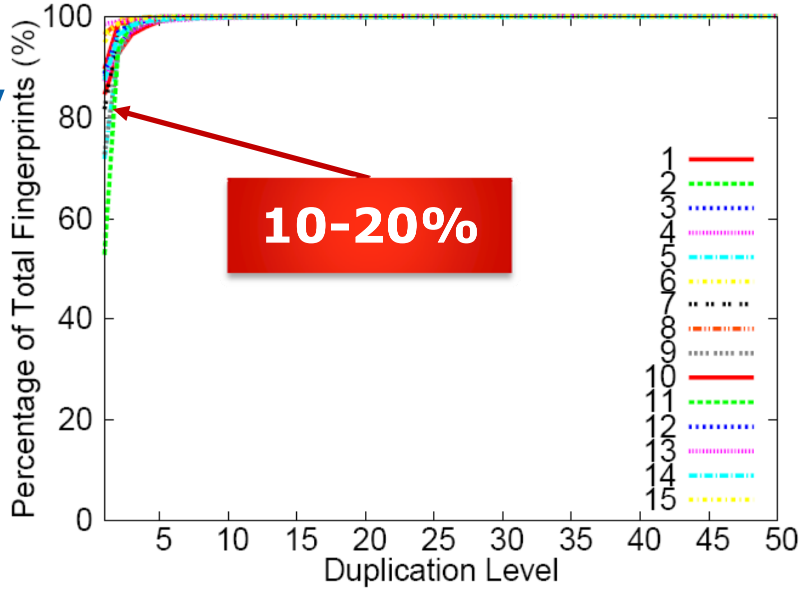
Fingerprints

- A 160-bit SHA-1 hash value for a page
- Identifying duplicate data by comparing fingerprints

Fingerprint Store

Fingerprint Store

- Maintaining fingerprints in memory



Challenges

- Memory overhead (25 bytes each)
- Fingerprint store lookup overhead

Observations & indications

- Skewed duplicate fingerprint distribution – only 10~20%
  - Most fingerprints are *NOT* duplicate → a waste of memory space
  - Most lookups *CANNOT* find a match → a waste of lookup latencies

We only need to store the most likely-to-be-duplicate fingerprints in memory and search them in the fingerprint store

Outline

- Introduction
- Hashing and Fingerprint Store
- Indirect mapping
- Acceleration methods
- Evaluation
- Conclusion

Indirect Mapping Mechanism

Existing Mapping Structure

- Essentially ***1-to-1*** mapping
  - Forward mapping: LBA → PBA (1:1)
  - Reverse mapping: PBA → LBA (1:1)

Indirect mapping in CAFTL

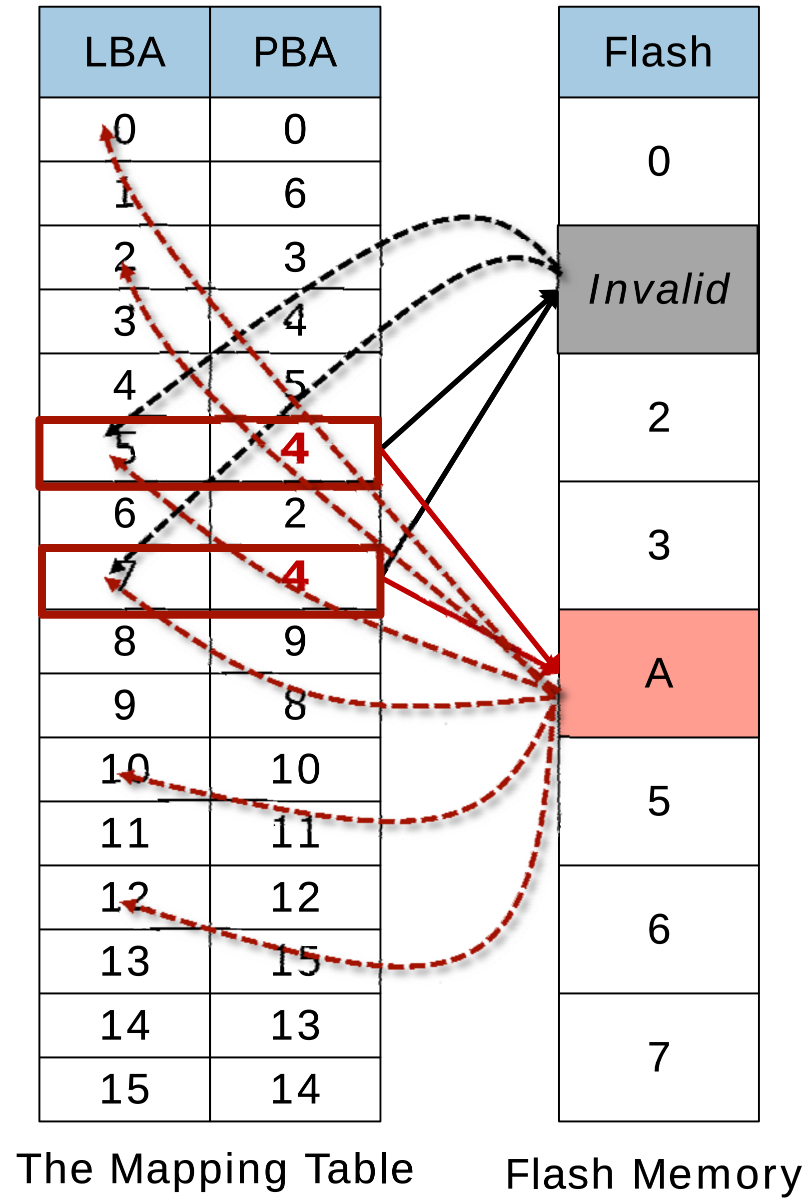
- Essentially ***N-to-1*** mapping
  - Forward mapping: LBA → PBA (N:1)
  - Reverse mapping: PBA →LBA (**1:N**)

Challenges – Reverse Mapping

- # of sharing LBAs can be large/variable
- LBAs sharing a PBA can change on the fly

How to keep reverse-mapping info?

- Array, list, exhaustive scanning – high cost
- Keep/updating info in flash – slow/complex



Two-level Indirect Mapping

Virtual block address (VBA)

- A pseudo address – sharing LBAs

Primary mapping table

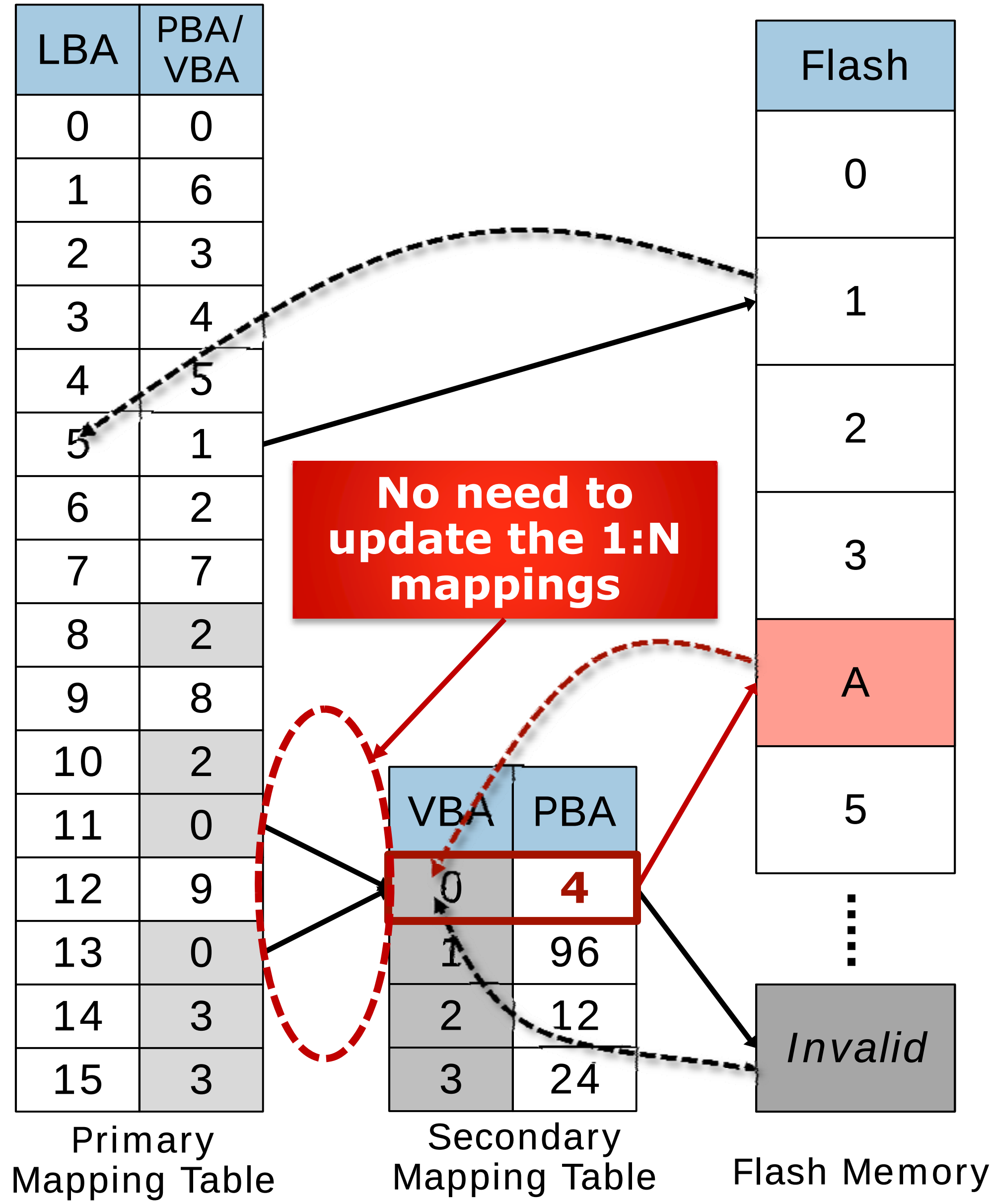
- Unique pages – LBA → PBA (1:1)
- Shared pages – LBA → VBA (**N:1**)

Secondary mapping table

- VBA → PBA (1:1)

Reverse mapping

- Unique pages – PBA → LBA (1:1)
- **Shared pages** – PBA → VBA (**1:1**)



Outline

- Introduction
- Hashing and Fingerprint Store
- Indirect mapping
- Acceleration methods
- Evaluation
- Conclusion

Acceleration Methods

Overhead of fingerprinting

- SHA-1 hash function incurs high overhead
- On-device buffer size is limited and can be overfilled
- Dedicated hash engine increases production cost

Acceleration methods

- Sampling for hashing
- Light-weight pre-hashing
- Dynamic Switch

Sampling for Hashing

Principle – Speeding up the common case

- Most writes are *unique* → most hashing operations turn out useless eventually

Intuition

- If a page in a write is a duplicate page, the other pages are likely to be duplicate too

Sampling

- Select one page in a write request as a **sample**
- If the sample page is duplicate, hash and examine the other pages
- Otherwise, we stop fingerprinting the whole request at the earliest time

Technical Challenges

- No file-level info available → e.g. we cannot use the first page in a file
- Overhead concerns → e.g. we cannot rely on hashing to select samples

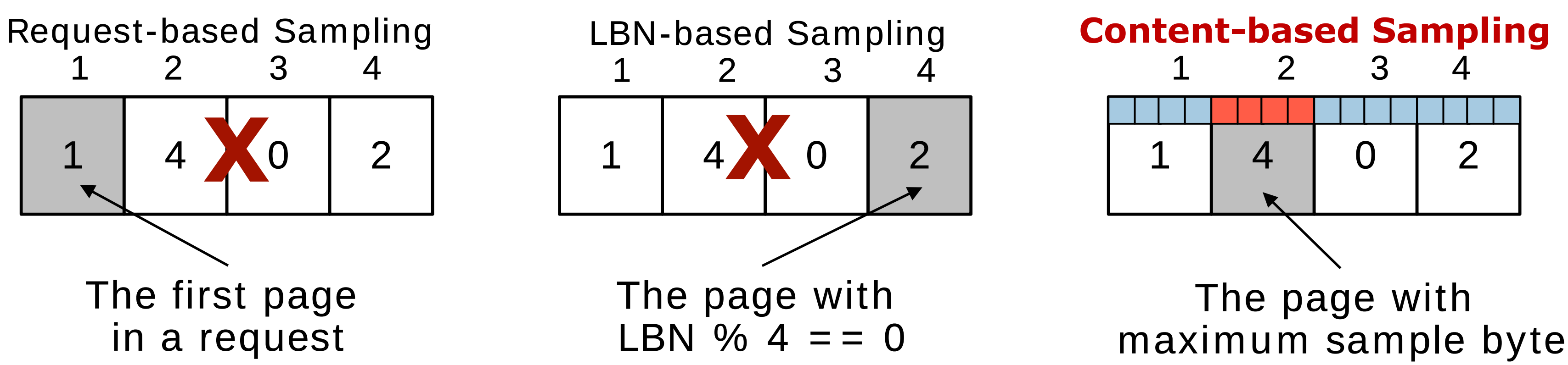
Selecting Sample Pages

Potential candidate solutions

- Request-based Sampling → requests may not repeat
- LBN-based Sampling → written locations may not repeat

Content-based Sampling

- Selecting/comparing first four bytes in each page
- The page with the largest sample bytes is the sample page
- Sample bytes – the first four bytes are the best choice



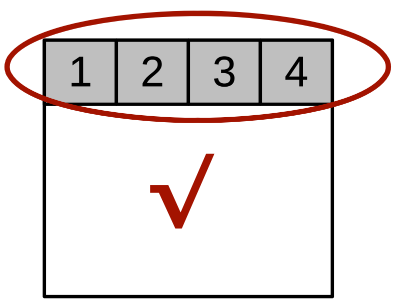
Selecting Sample Pages

Potential candidate solutions

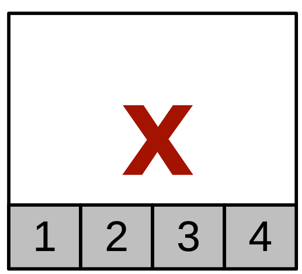
- Request-based Sampling → requests may not repeat
- LBN-based Sampling → written locations may not repeat

Content-based Sampling

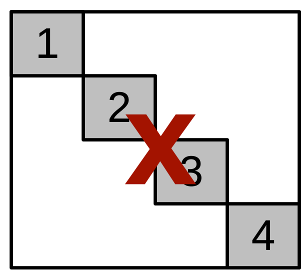
- Selecting/comparing first four bytes in each page
- The page with the largest sample bytes is the sample page
- Sample bytes – the first four bytes are the best choice



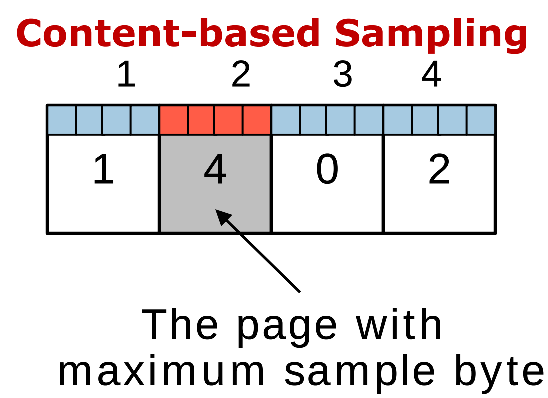
First 4 bytes



Last 4 bytes



Sparse 4 bytes



Outline

- Introduction
- Hashing and Fingerprint Store
- Indirect mapping
- Acceleration methods
- Evaluation
- Conclusion

Performance Evaluation

SSD simulator

- Microsoft® Research SSD extension for *DiskSim* simulator*
  - Indirect mapping, wear-leveling, garbage collection, etc.
- Simulator augmented with CAFTL design and an on-device buffer

SSD configurations

- Default configuration numbers
- Estimated latencies of hashing code on ARM simulator

Description	Configurations
Flash page size	4KB
Pages / block	64
Blocks / plane	2048
Num of pkgs	10
Over-provisioning	15%

Description	Latency
Flash Read	25µs
Flash write	200µs
Flash Erase	1.5ms
SHA-1 hashing	47,548 cycles
CRC32 hashing	4,120 cycles

* MSR, http://research.microsoft.com/en-us/downloads/b41019e2-1d2b-44d8-b512-ba35ab814cd4/

Workloads

Desktop (*d1, d2*)

- Office workloads – Web surfing, emailing, word editing (12 and 19 hours)
- Workloads feature irregular idle intervals and small read/writes

TPC-H queries (*h1-h7*)

- TPC-H queries – Query 1,6,14,15,16,20 (Scale factor of 1)
- Workloads run on Hadoop distributed system platform (2~40 min)
- Workloads feature intensive large writes of temp data

Transaction processing (*t1, t2*)

- TCP-C workloads – Transaction processing on PostgreSQL 8.4.3 database systems (1,3 warehouses, 10 terminals)
- Workloads run for 30 min and 4 hours with intensive write operations

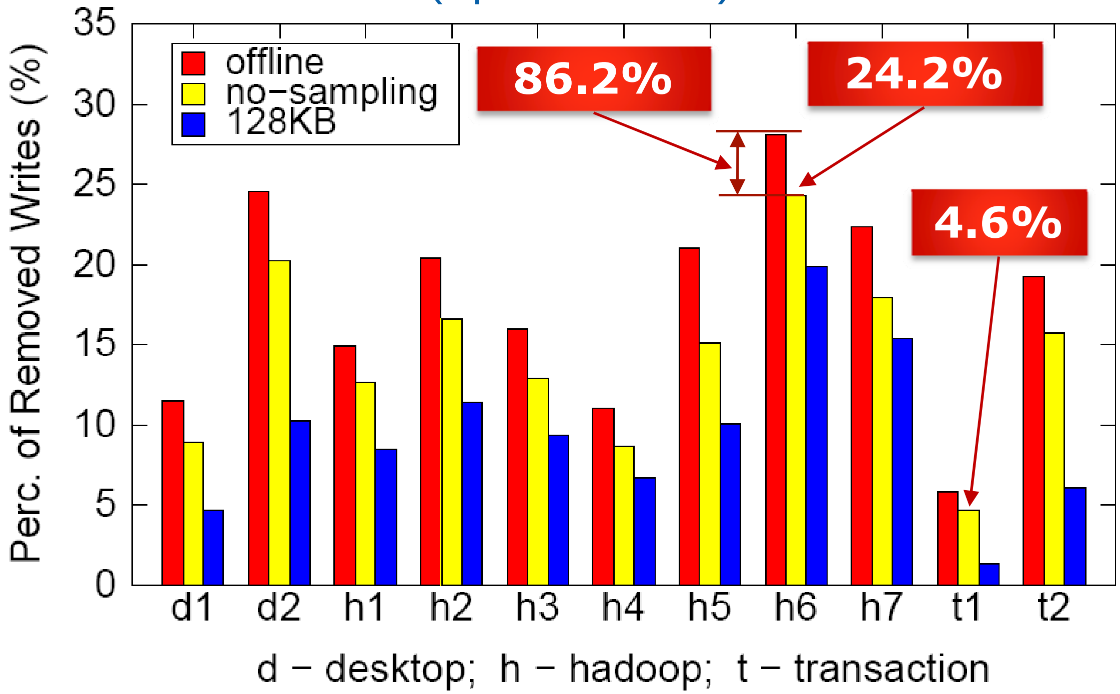
Effectiveness of De-duplication

Removing duplicate writes

- Deduplication Rate: $(n-m)/n$
  - n – total number of pages of incoming write requests
  - m – total number of pages being actually written into flash memory

Experimental Results

- Deduplication Rate: 4.6% (*t1*) ~ 24.2% (*h6*)
- Up to 86.2% of the duplicate writes in *offline* (optimal case)



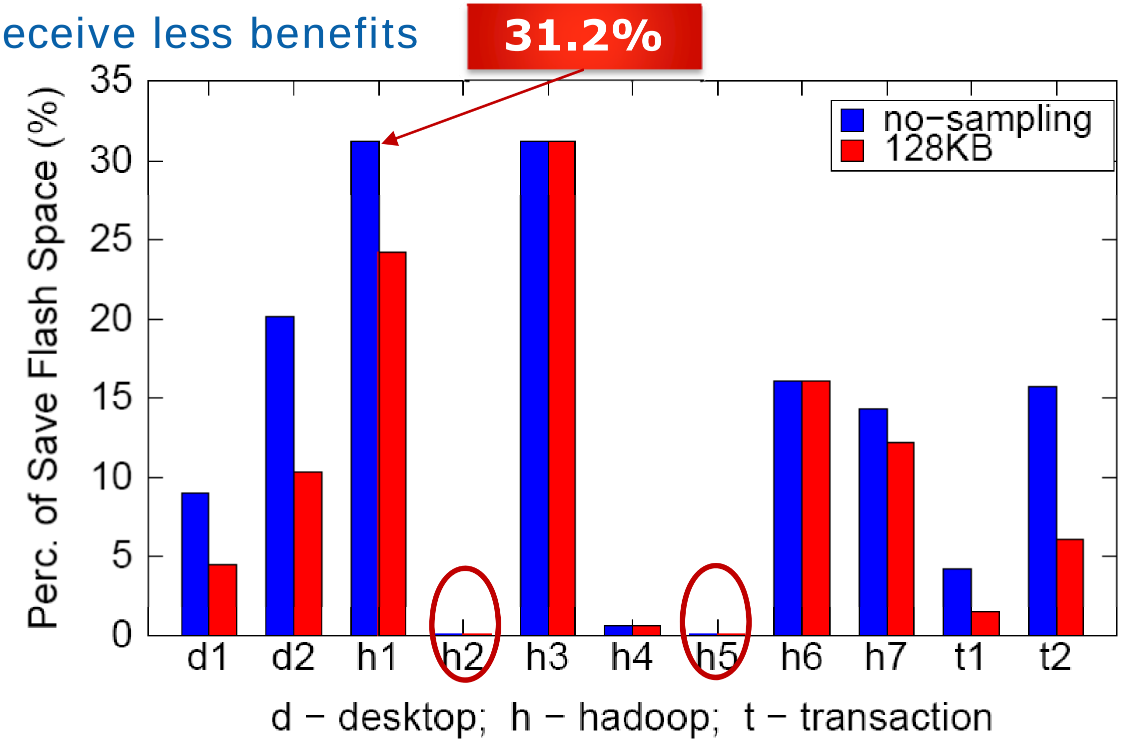
Effectiveness of De-duplication

Extending flash space

- Space Saving Rate: $(n-m)/n$
  - n –total number of occupied erase blocks of flash memory w/o CAFTL
  - m –total number of occupied erase blocks of flash memory w/ CAFTL

Experimental Results

- Space Saving Rate: up to 31.2% (*h1*)
- Small workloads (*h2*, *h5*) receive less benefits



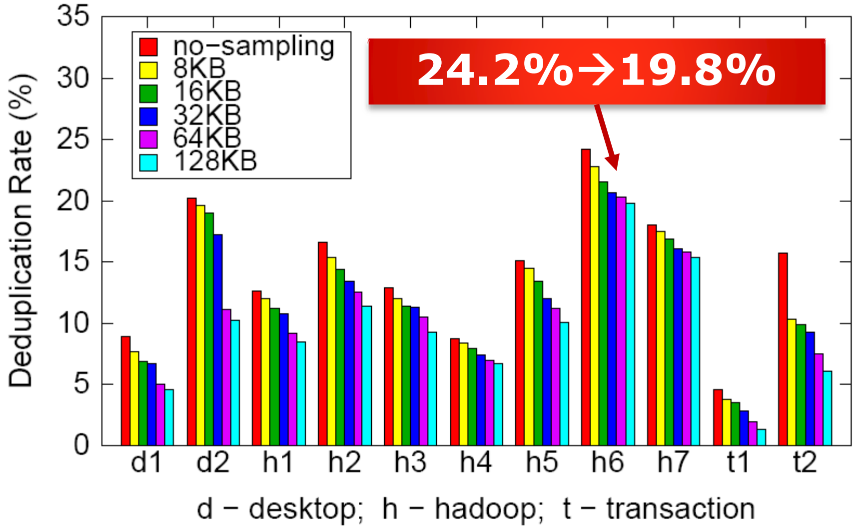
Effectiveness of Sampling

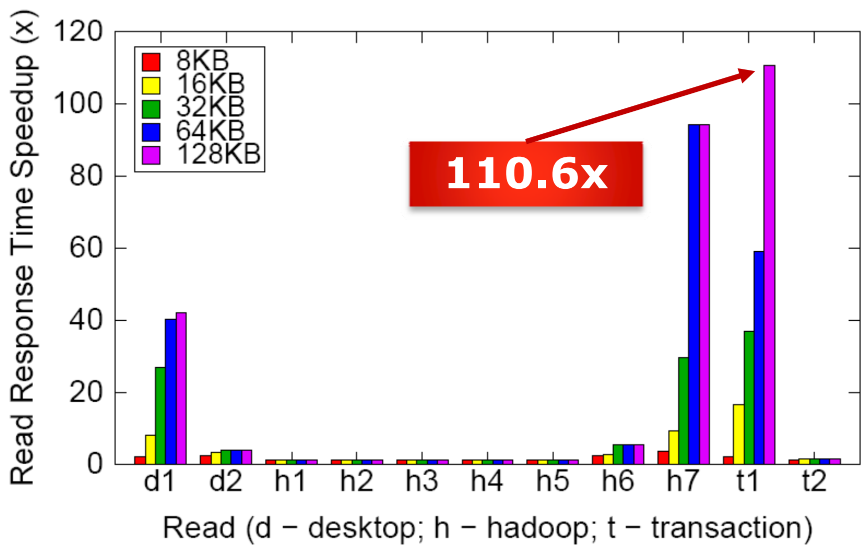
Response Time Speedup

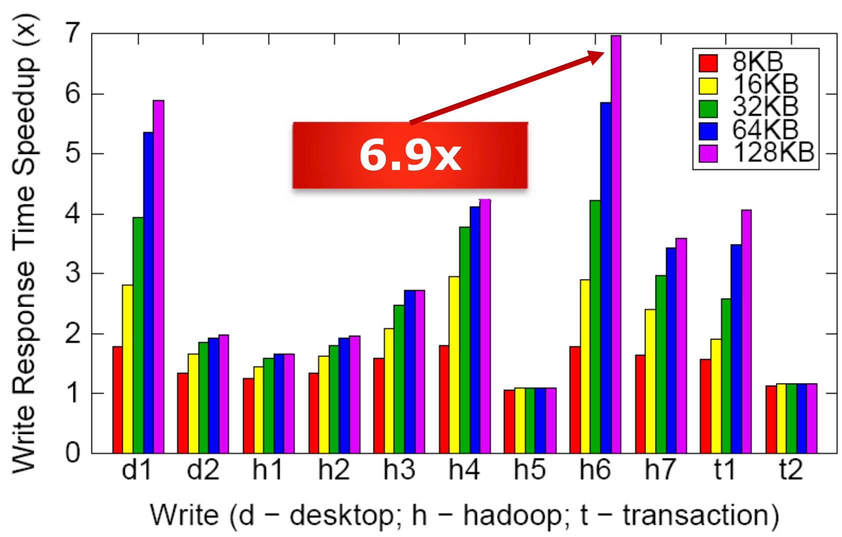
- Read – up to 110.6x
- Write – up to 6.9x

Deduplication Rate Reduction

- Dedup Rate – 24.2% → 19.8% (*h6*)







Outline

- Introduction
- Hashing and Fingerprint Store
- Indirect mapping
- Acceleration methods
- Evaluation
- Conclusion

Conclusion

- SSD endurance would remain a serious concern in reality
- Data duplication is common in regular file systems, which provides unique opportunities for improving SSD lifespan via deduplication on the device
- We present a unique Content-Aware Flash Translation Layer (CAFTL) to remove duplicate writes and coalesce redundant data in SSDs on the fly
- We show that CAFTL can effectively improve SSD lifespan via on-device deduplication while retaining low performance overhead

Thank You !

Feng Chen

✉: *feng.a.chen@intel.com*