ELECTRICAL & COMPUTER ENGINEERING
TEXAS A&M UNIVERSITY

Virtualize and Share Non-Volatile Memory in User Space

Chih Chieh Chou, Jaemin Jung, A. L. Narasimha Reddy, Paul Gratz, and Doug Voigt

May 23, 2019

Outline

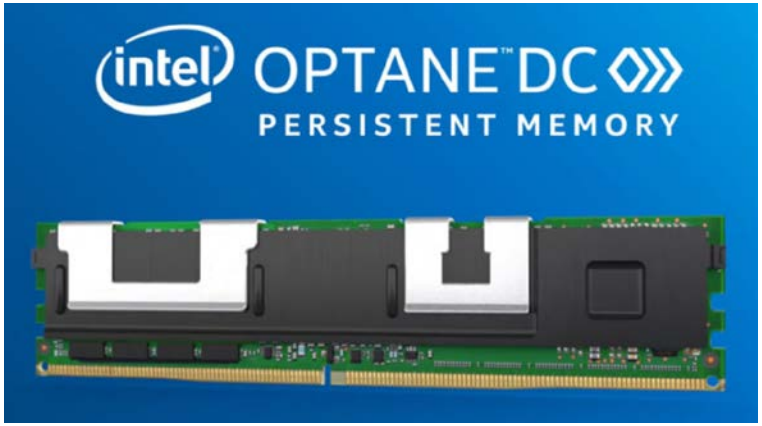
- Introduction
- Motivation and Goal
- Architecture
- Conclusions
- Acknowledgements

Introduction

ELECTRICAL & COMPUTER ENGINEERING
TEXAS A&M UNIVERSITY

- The non-volatile memory has becomes promising storage device because of some amazing properties
  - Byte-addressability
  - Non-volatility
  - Low latency
  - Low power in idle (except for NVDIMM)

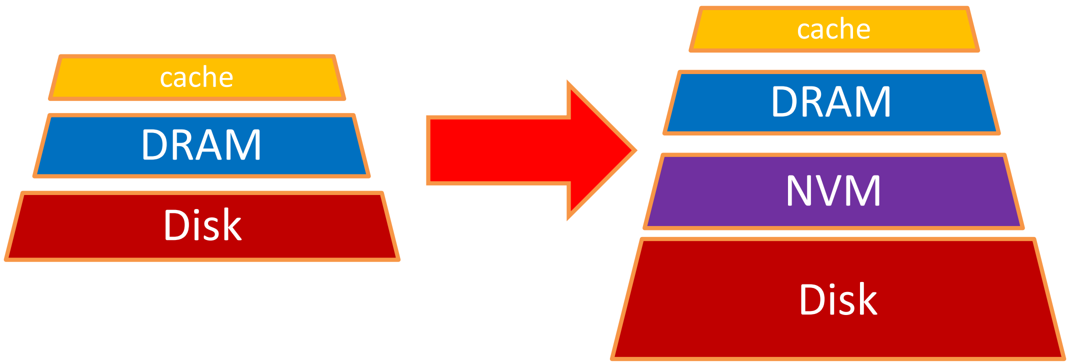
HPE 8GB NVDIMM single Rank x4 DDR4-2133 Module

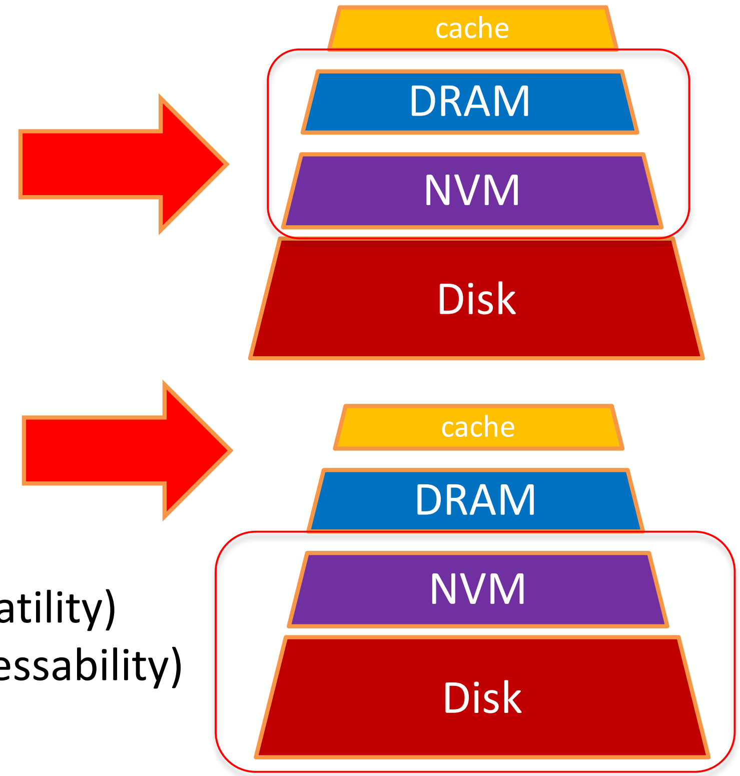


Introduction

ELECTRICAL & COMPUTER ENGINEERING
TEXAS A&M UNIVERSITY

- Unlike DRAM and disk, how to deploy NVM (put in which layer of memory hierarchy) does not have an agreement so far





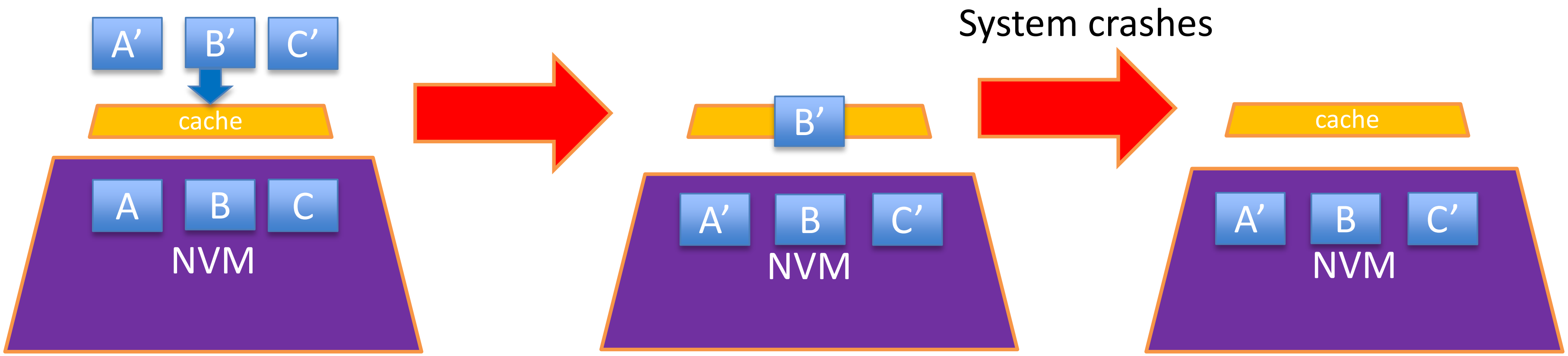
1. Use DRAM as cache of NVM (w/o non-volatility)
2. Use NVM as cache of disk (w/o byte-addressability)

Can we do more?

Challenge

ELECTRICAL & COMPUTER ENGINEERING
TEXAS A&M UNIVERSITY

- Directly attach to memory bus as DIMM under cache are "not persistent" after power cycling
- Need write ordering! (sol: logs and transactions)



Motivations

- Several prior work focusing on building a specific file system tailed for NVM
- Scmfs (SC’11), NOVA (FAST’16, MSST’17), Strata (SOSP’17)
  - Limit users to use their file systems
  - No concurrency
  - System calls are too expensive and will squander the low latency provided by NVM
    - Handling almost everything in user space provides much better performance
    - Intel SPDK (https://spdk.io): user space, polling-based, NVMe driver
      - ULL SSD: Intel Optane SSD/Samsung Z-NAND

Motivations

- SNIA NVM Programming Model/Intel PMDK(https://pmem.io/pmdk/)
  - Use mmap interface to access NVM
- Virtualize and share NVM (between processes), like virtual memory (mmap)
  - Virtual NVM capacity more than physical available capacity
    - Leveraging storage device as data final destination
- Leveraging DRAM as cache
  - Performance: better latency; avoid log searching
  - Write lifetime issue of PCM: reduce write to NVM

Our Goals

- User space
  - library
- Transactional interface
  - Log
- mmap-like access form
- Virtualization and sharing of NVM
  - Leverage storage device
- DRAM cache

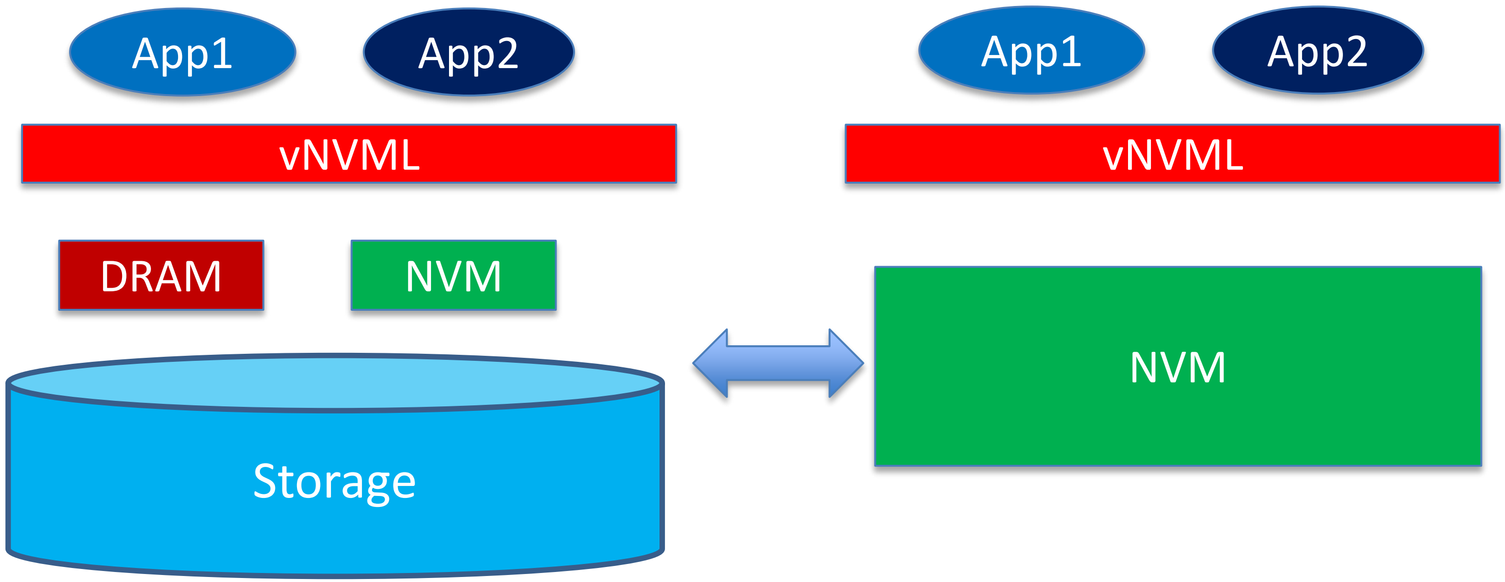
Methodology

- Leveraging the existing mmap function
- Integrate DRAM, NVM, and SSD to provide virtual NVM
  - Treat (DRAM + NVM + SSD) as a huge NVM pool
  - Its performance is very close to that of NVM (or DRAM)

Methodology

ELECTRICAL & COMPUTER ENGINEERING
TEXAS A&M UNIVERSITY

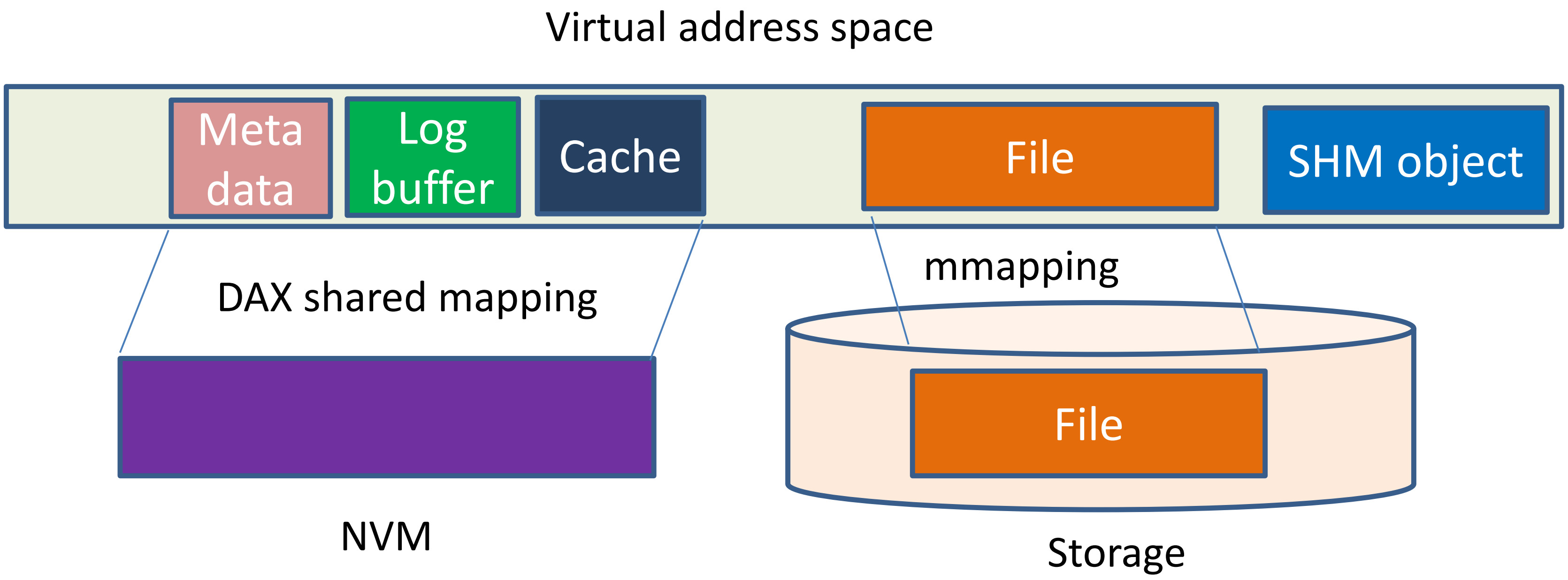
- User space library: vNVML
  - Access NVM only through vNVML
  - Support concurrently (processes) access
  - Allocate (virtual) NVM regardless of actual NVM size



Example

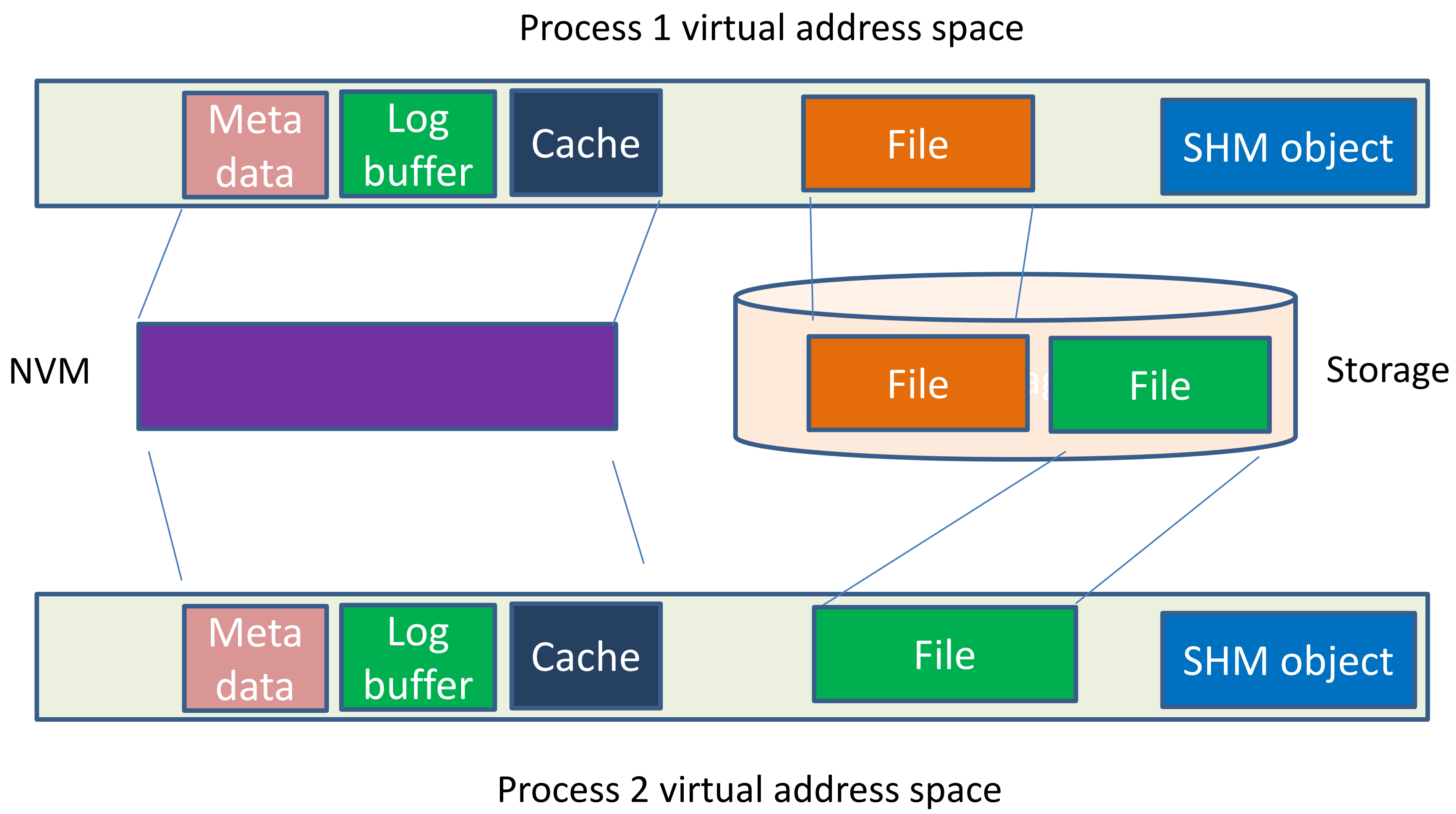
```
ptr = nv_allocate(filepath, filesize, mode);
tid = nv_txbegin();                                  // TX starts
x = *ptr;                                            // read
y = *(ptr + sizeof(x));                              // read
x = 1;
y = 2;
nv_write(tid, ptr, &x, sizeof(x));                   //write
nv_write(tid, ptr+sizeof(x), &y, sizeof(y));         //write
nv_commit(tid);                                      //TX commits
```

Components



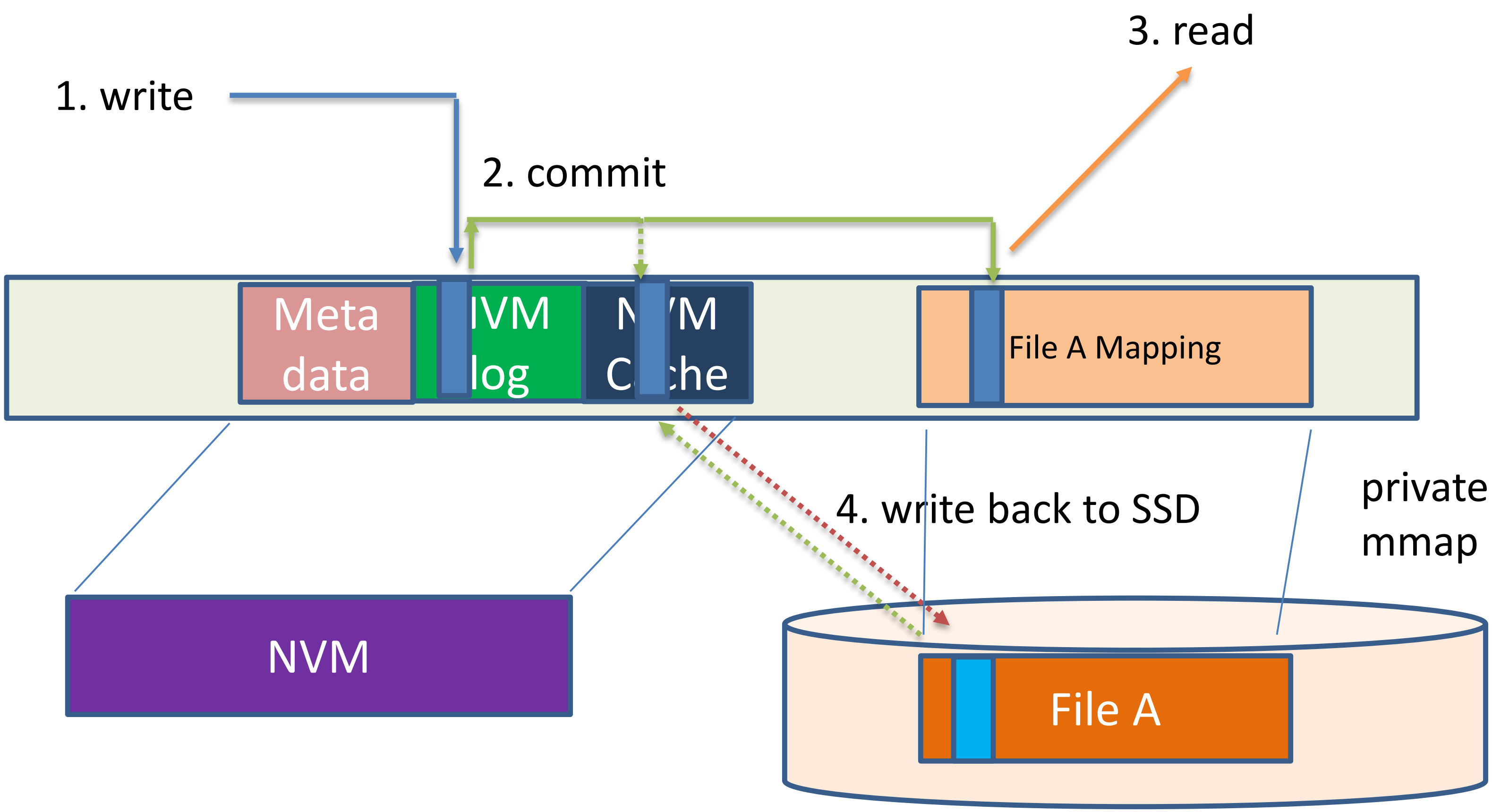
- Limitations/challenge:
- 1. File system must support mmap
- 2. Virtual addressed cannot be stored in NVM

Components


Process 1 virtual address space
Meta data
Log buffer
Cache
File
SHM object
NVM
File
File
Storage
Meta data
Log buffer
Cache
File
SHM object
Process 2 virtual address space

Data access flow

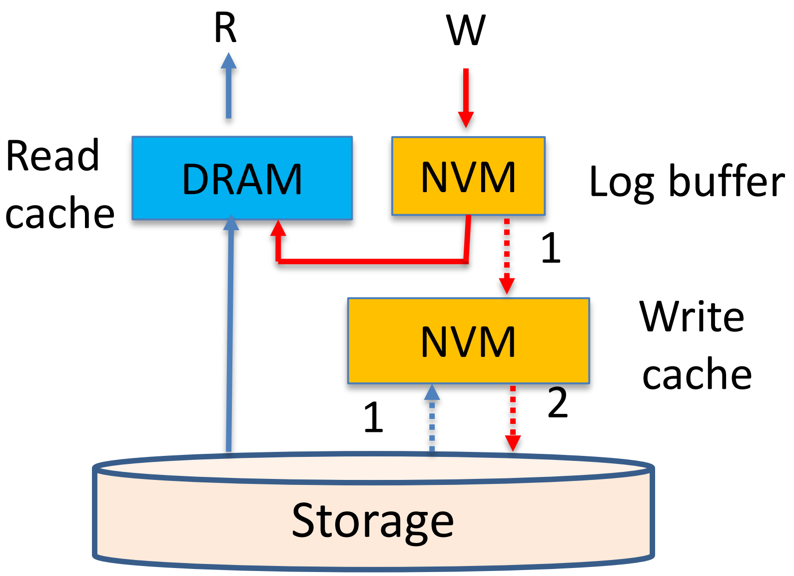
ELECTRICAL & COMPUTER ENGINEERING
TEXAS A&M UNIVERSITY



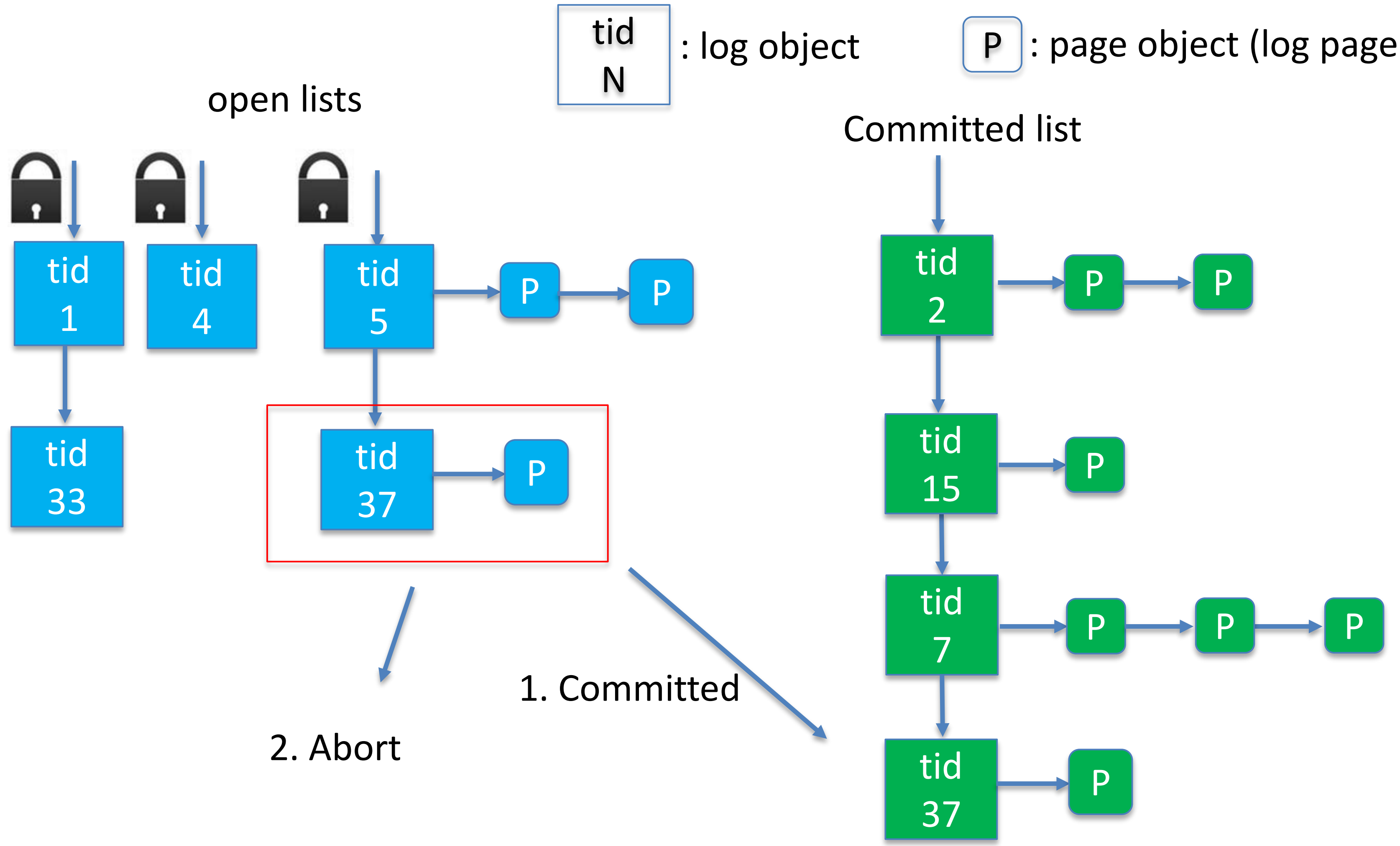
R/W flows

ELECTRICAL & COMPUTER ENGINEERING
TEXAS A&M UNIVERSITY

- DRAM as read only cache
- Limitation: Read committed TX
- NVM as log buffer and
- Write only cache
- Two background threads
  - Update the logs to write cache
  - Update the write cache to storage



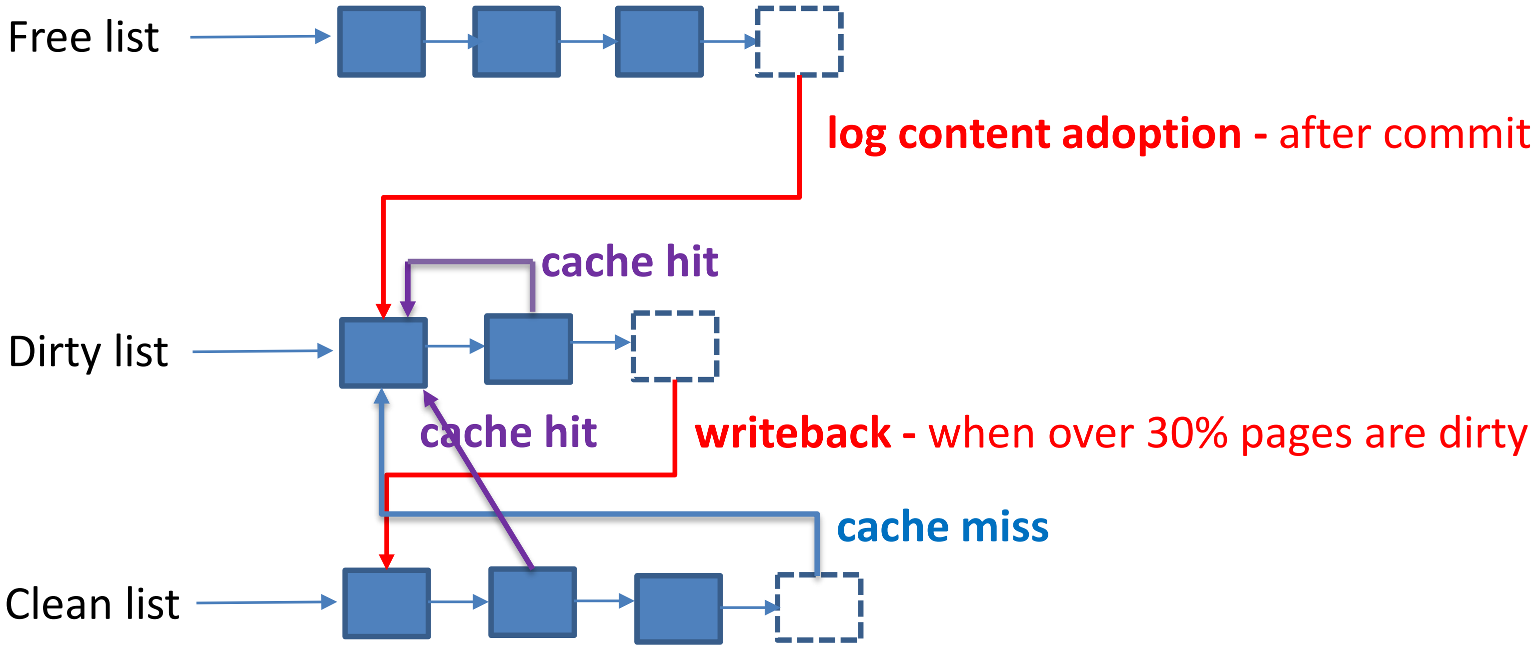
Log structure



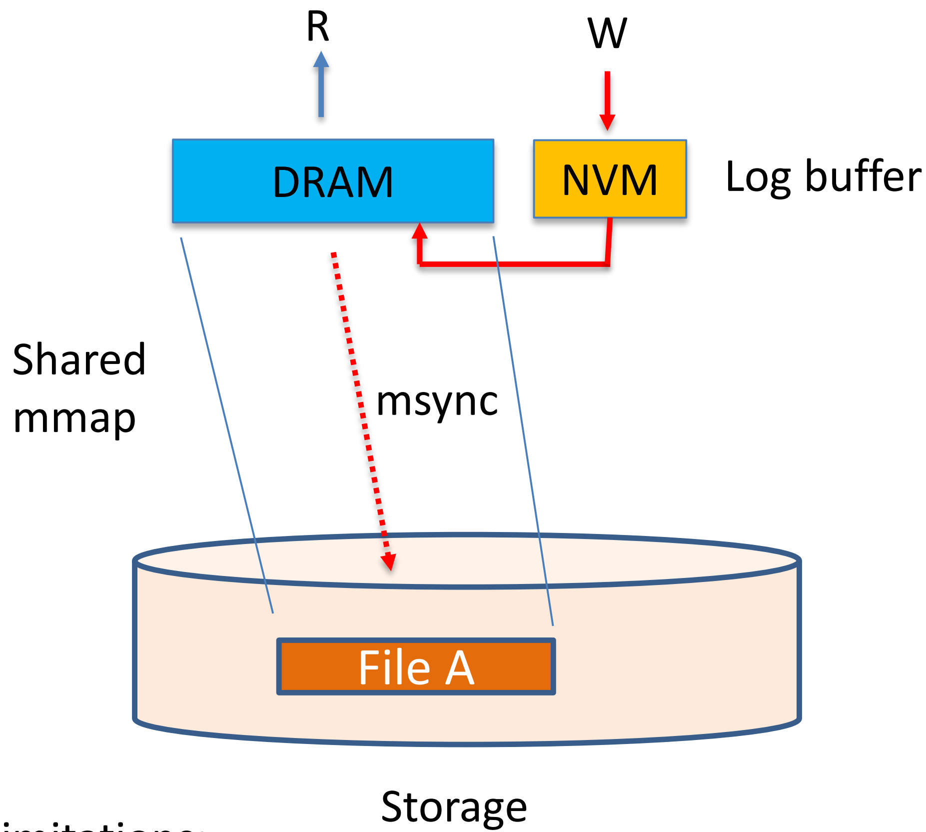
Limitation: write first should commit first (only when writing the same object)

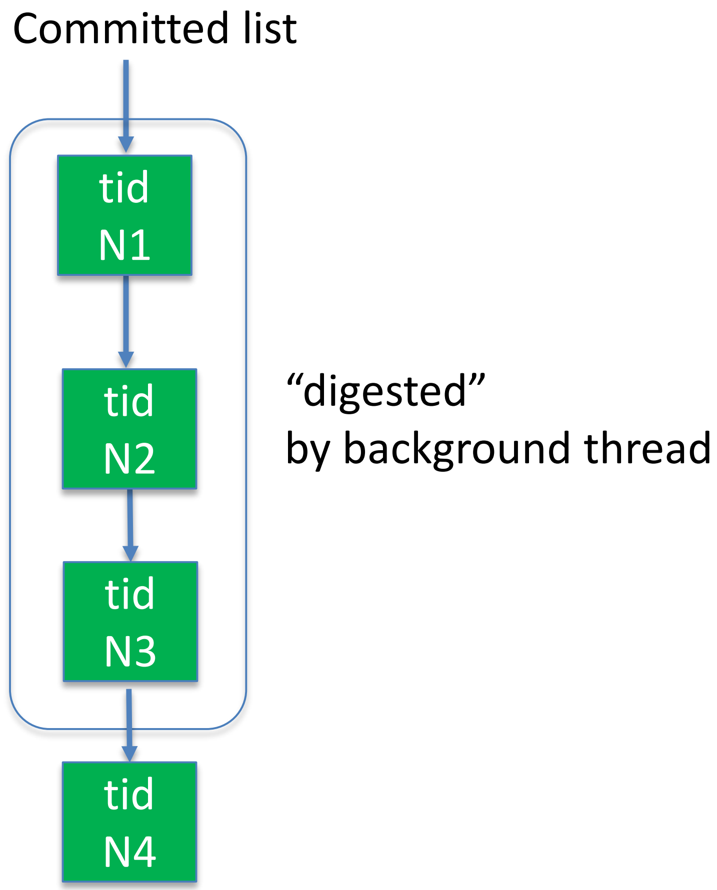
NVM Cache Management

ELECTRICAL & COMPUTER ENGINEERING
TEXAS A&M UNIVERSITY



Shared files



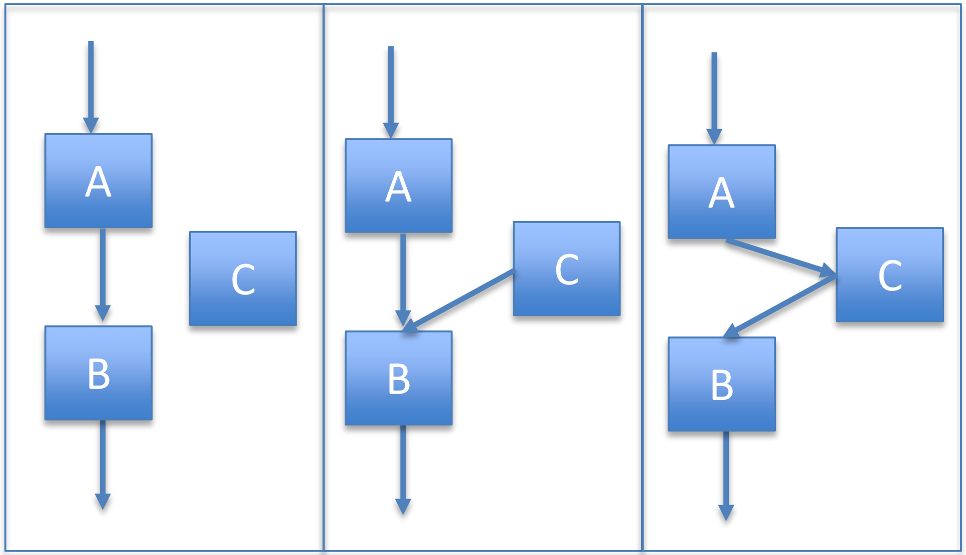


Limitations:

1. write first should commit first (only when writing the same object)
2. All writes of a TX must write to the same shared file

Recovery after crashing

- Dirty pages: check dirty bits
- Logs of committed list: leverage 8-byte atomicity (pointer) of cpu
- Insert: (a) => (b) => (c)
- Delete: (c) => (b) => (a)

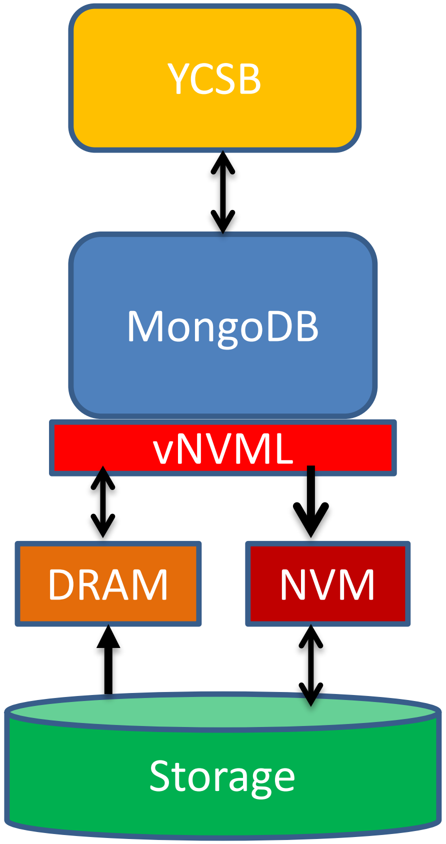


(a) (b) (c)

Experiment methodology

ELECTRICAL & COMPUTER ENGINEERING
TEXAS A&M UNIVERSITY

- YCSB + MongoDB + Library
  - YCSB generates read/write traffic (workload) to MongoDB
    - Fixed size record: 64KB
    - Run 100K records for each experiments
  - MongoDB accesses the NVM through library
  - Baseline: MongoDB generates files directly to NVM, and disables journaling/msync
- Platform setting:
  - 12GB DRAM, 12GB emulated NVM
  - CPU: 4 cores
  - 4 MongoDB instances run concurrently



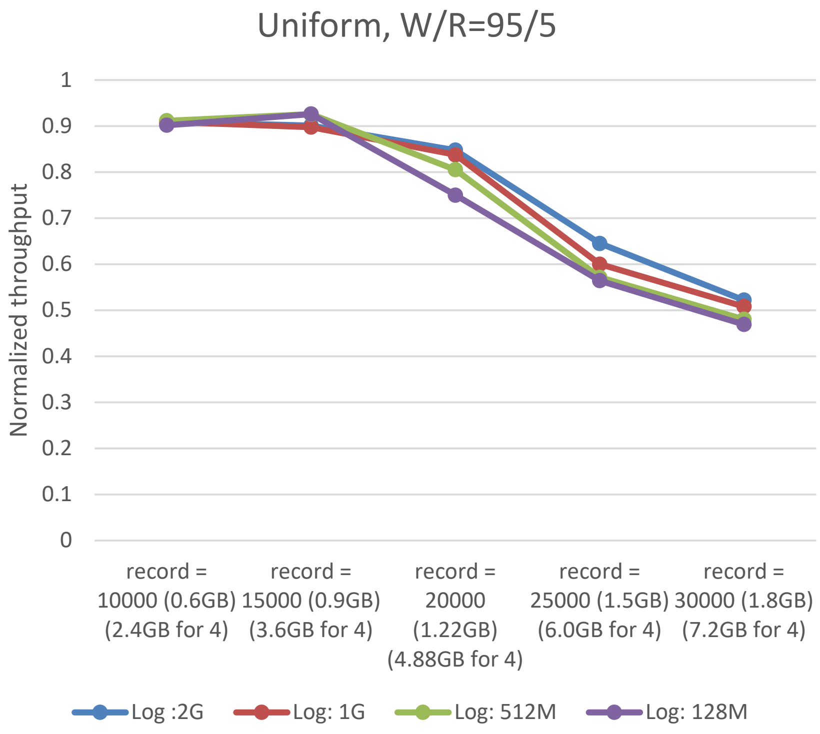
Evaluation

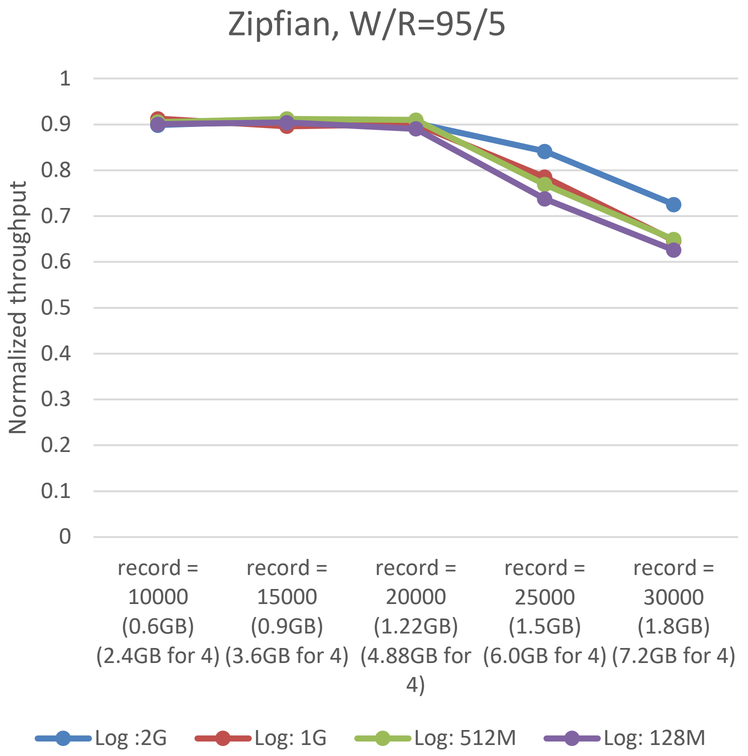
ELECTRICAL & COMPUTER ENGINEERING
TEXAS A&M UNIVERSITY

- Assume NVM size is fixed, how to partition the log buffer size and cache size?
- How does vNVML perform compared to other libraries?

Results of fixed cache size

- NVM cache size is 4GB, record number is the size of data set in the MongoDB

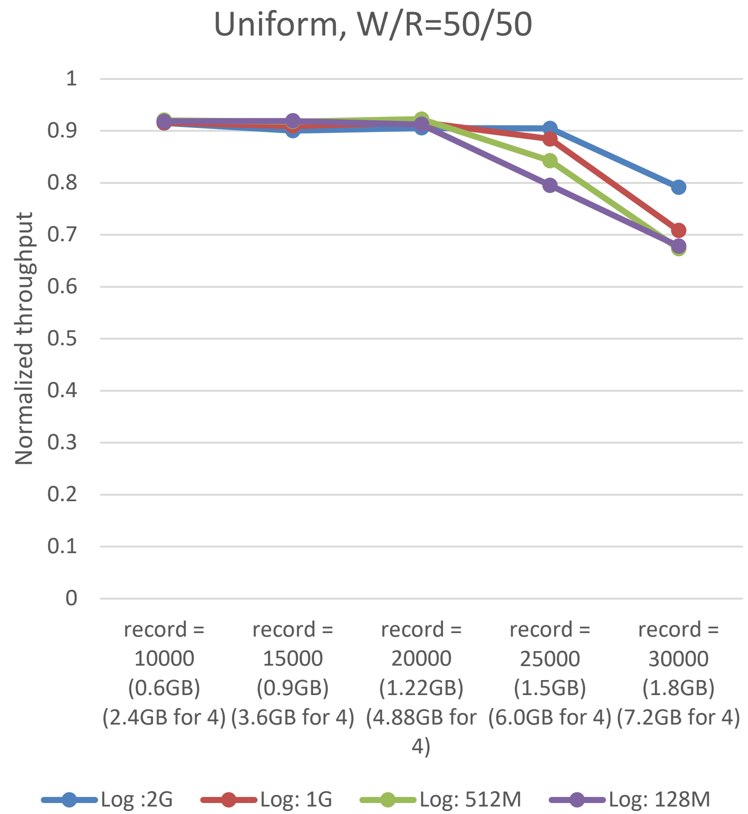


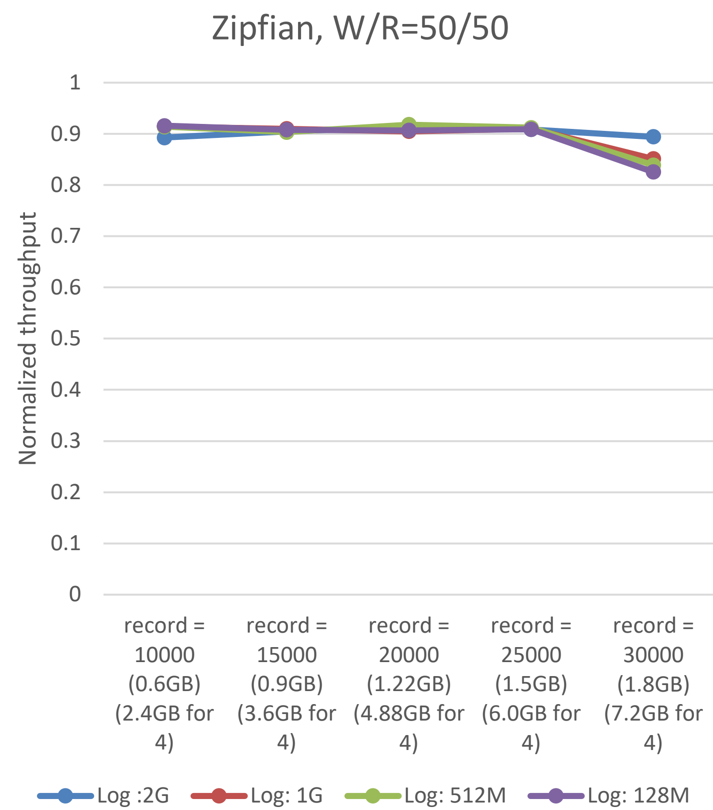


Results of fixed cache size

ELECTRICAL & COMPUTER ENGINEERING
TEXAS A&M UNIVERSITY

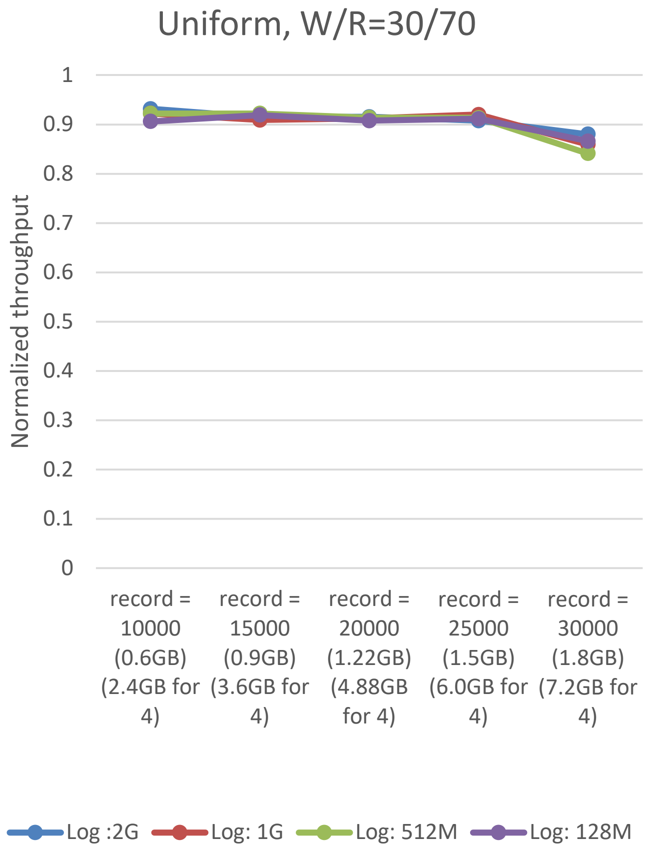
- NVM cache size is 4GB

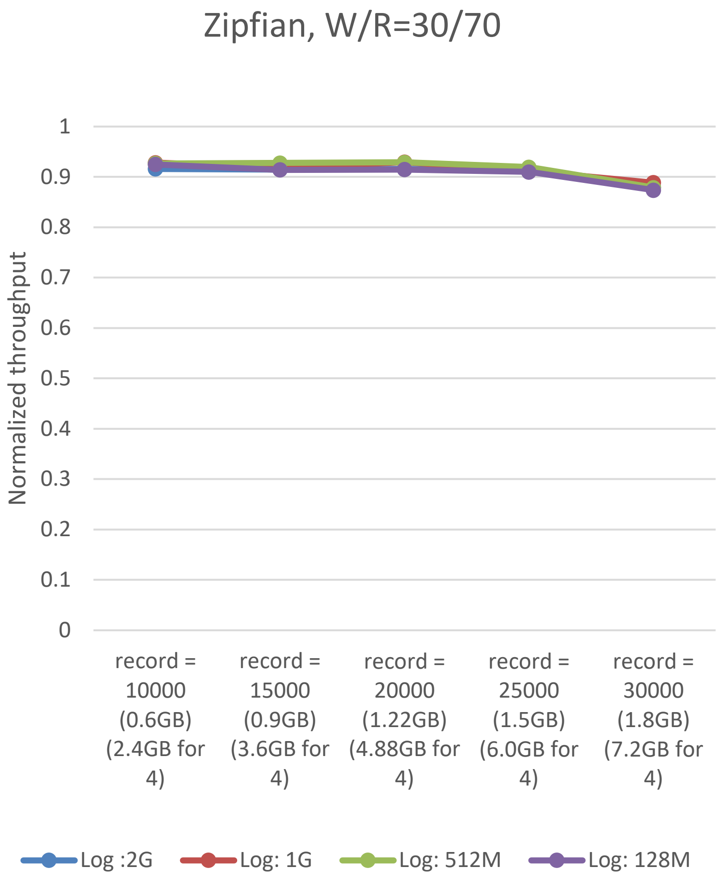




Results of fixed cache size

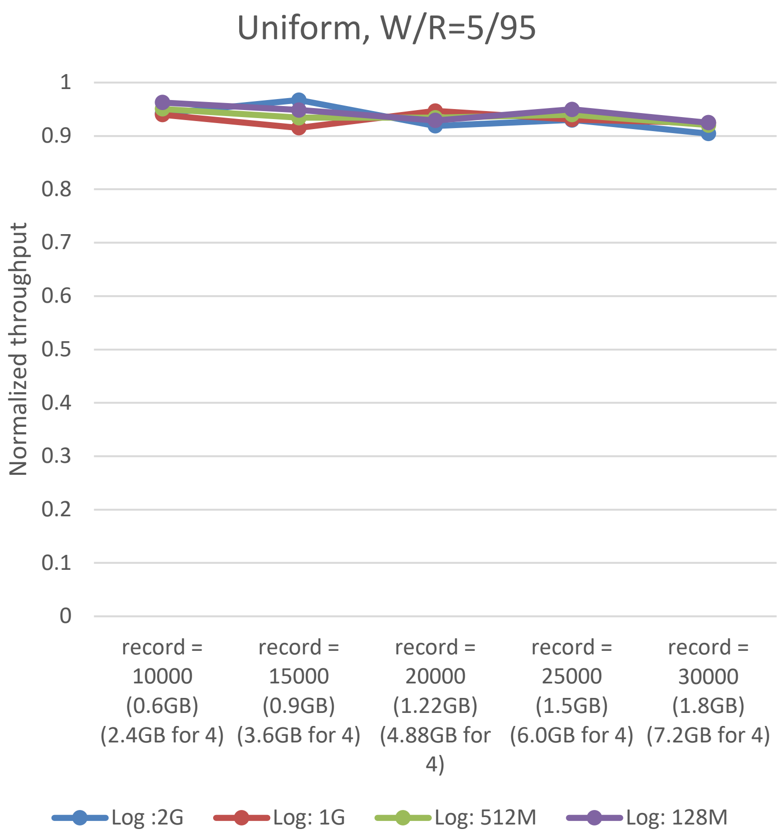
- NVM cache size is 4GB

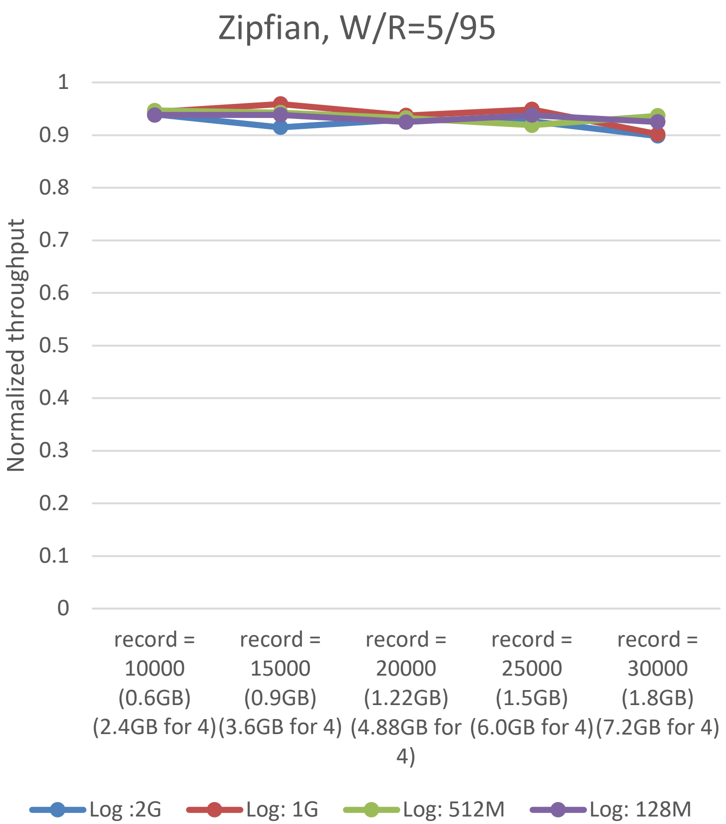




Results of fixed cache size

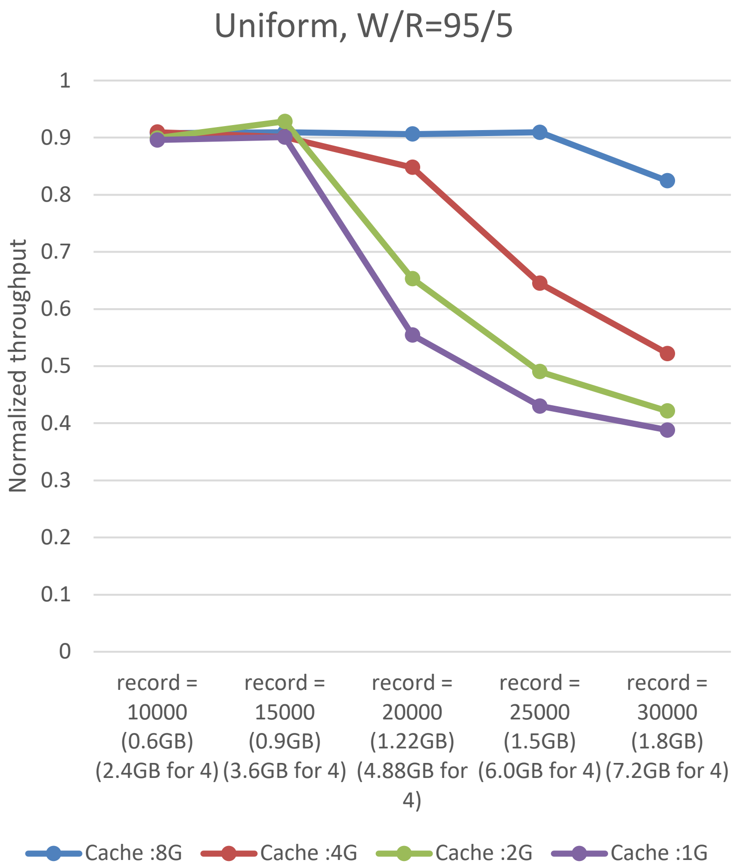
- NVM cache size is 4GB

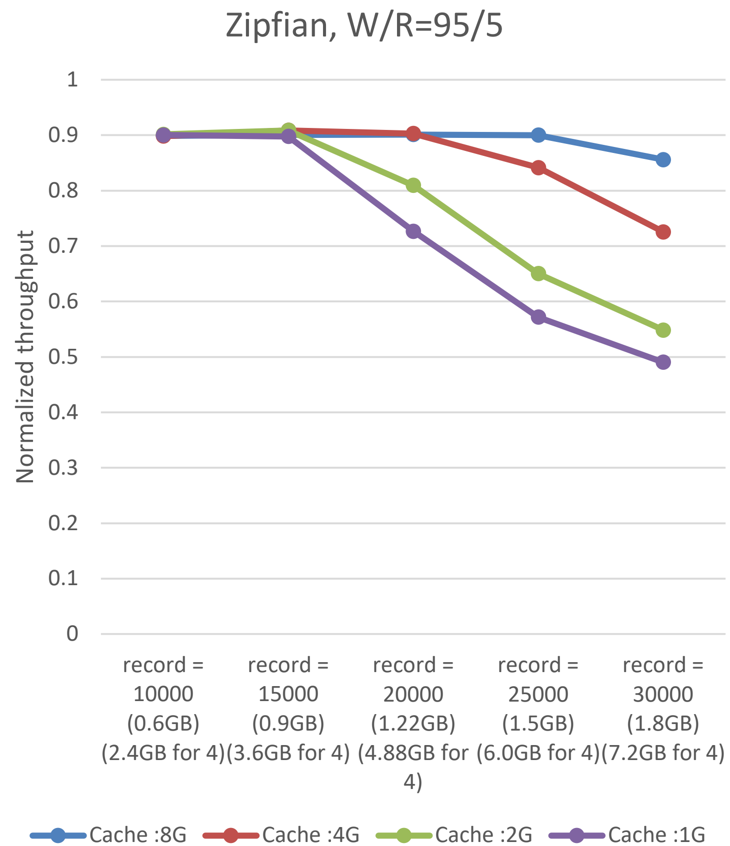




Results of fixed log buffer size

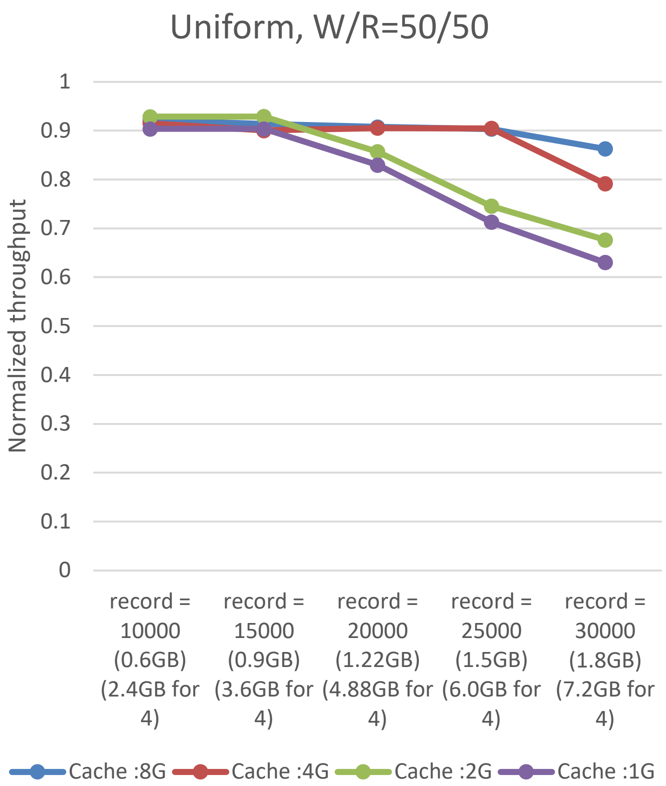
- NVM log buffer size is 2GB

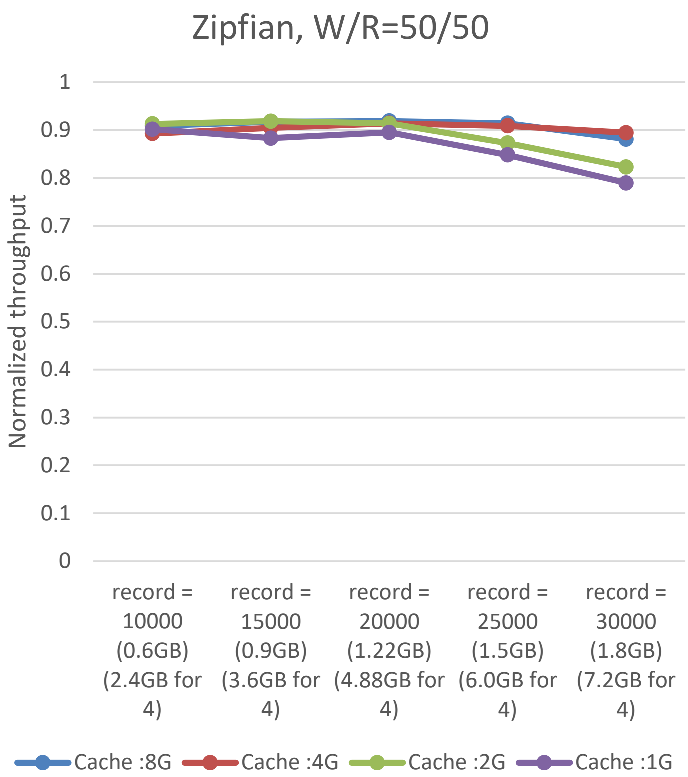




Results of fixed log buffer size

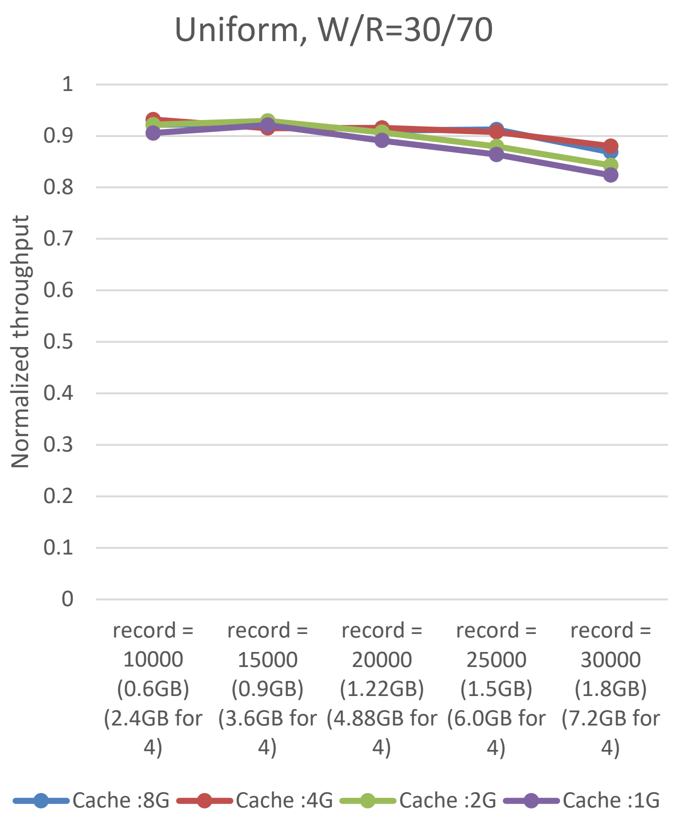
- NVM log buffer size is 2GB

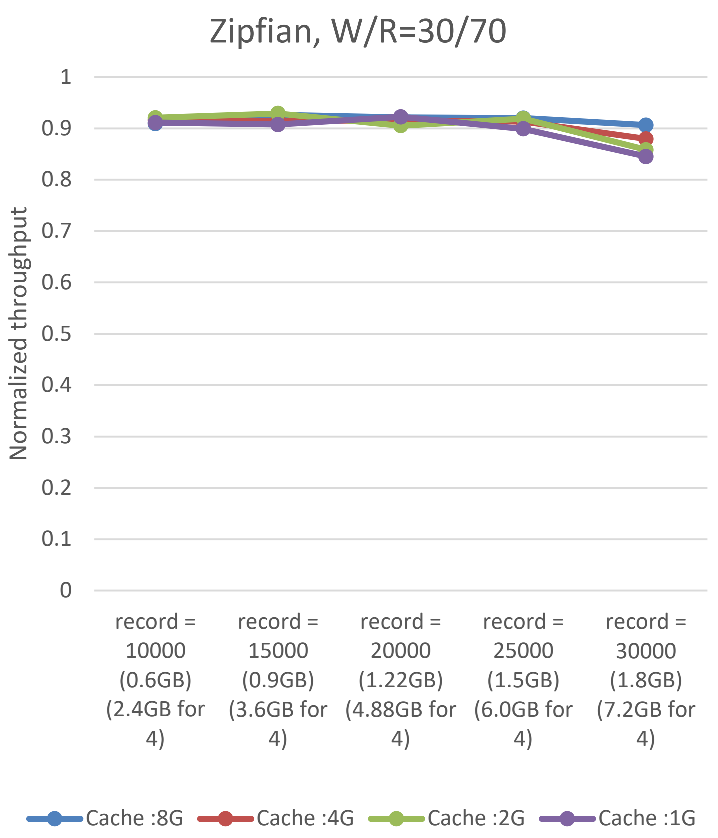




Results of fixed log buffer size

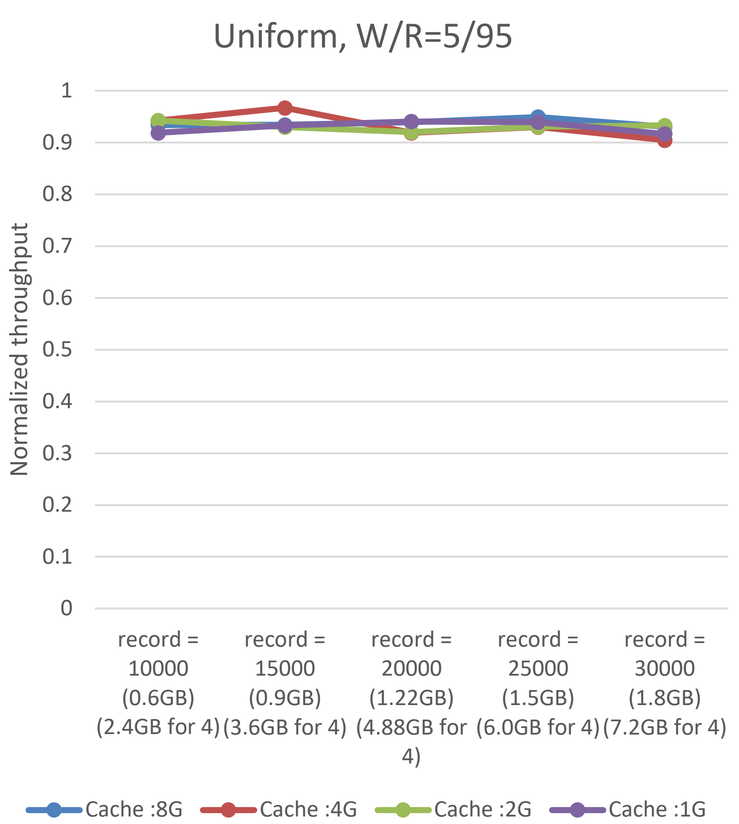
- NVM log buffer size is 2GB

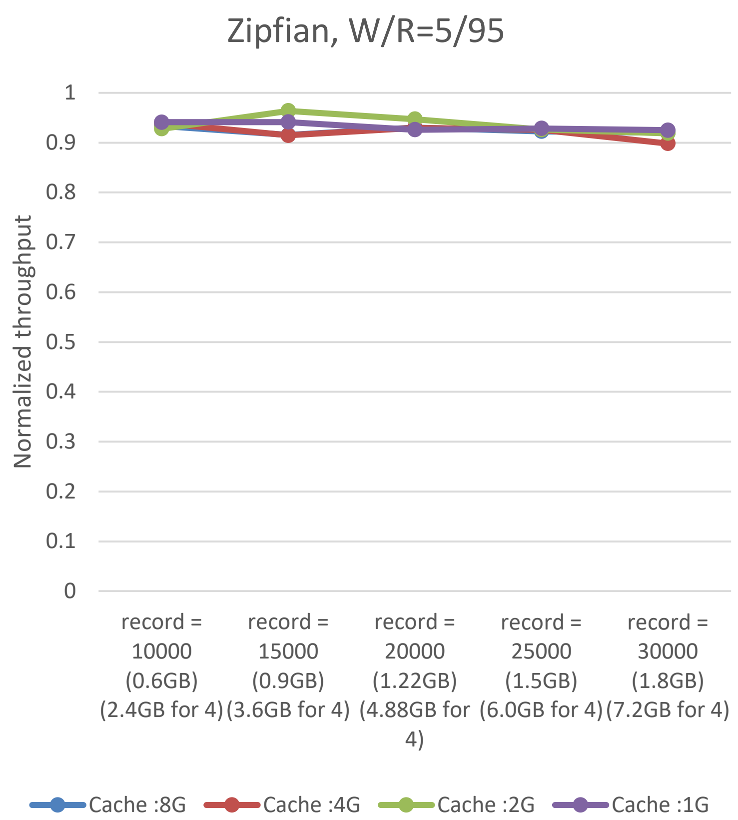




Results of fixed log buffer size

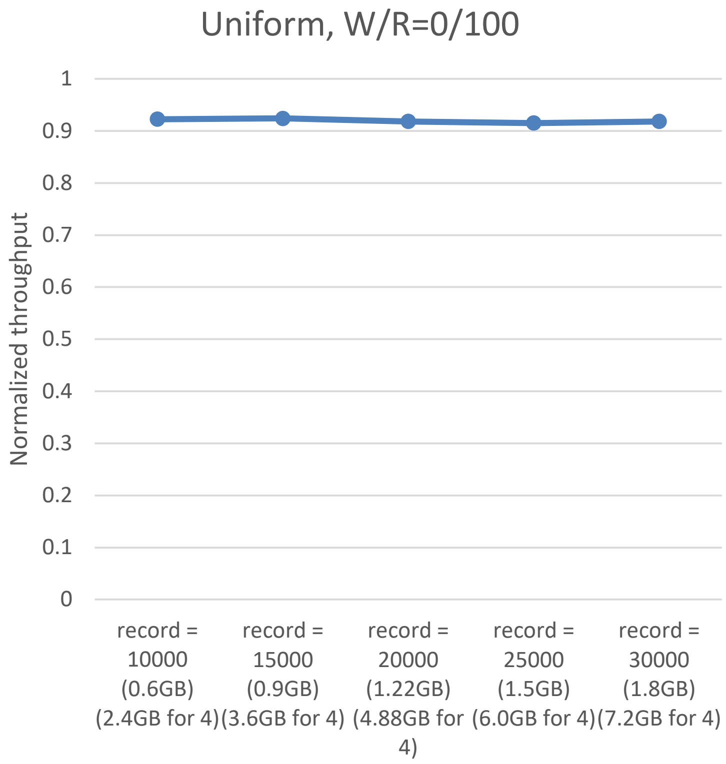
- NVM log buffer size is 2GB

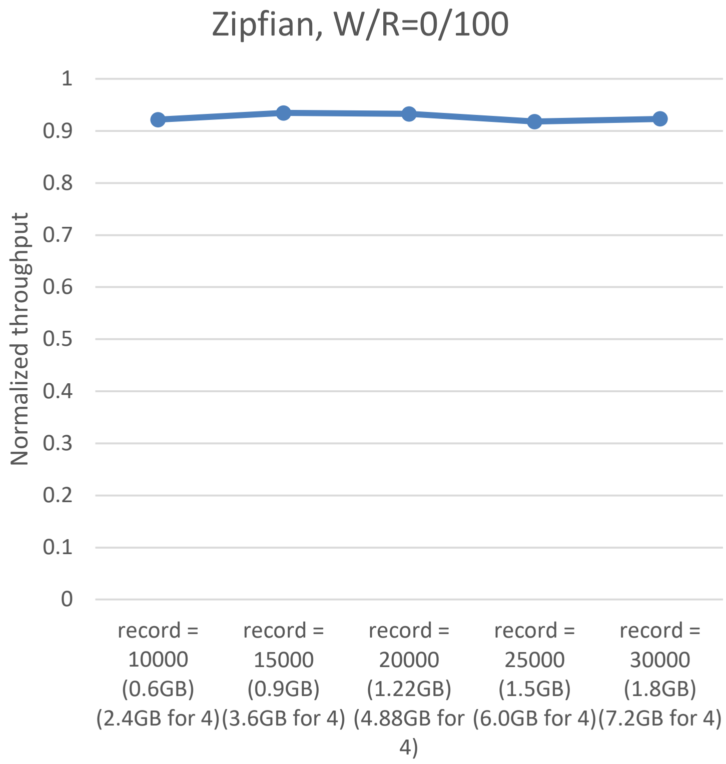




Results of read only case

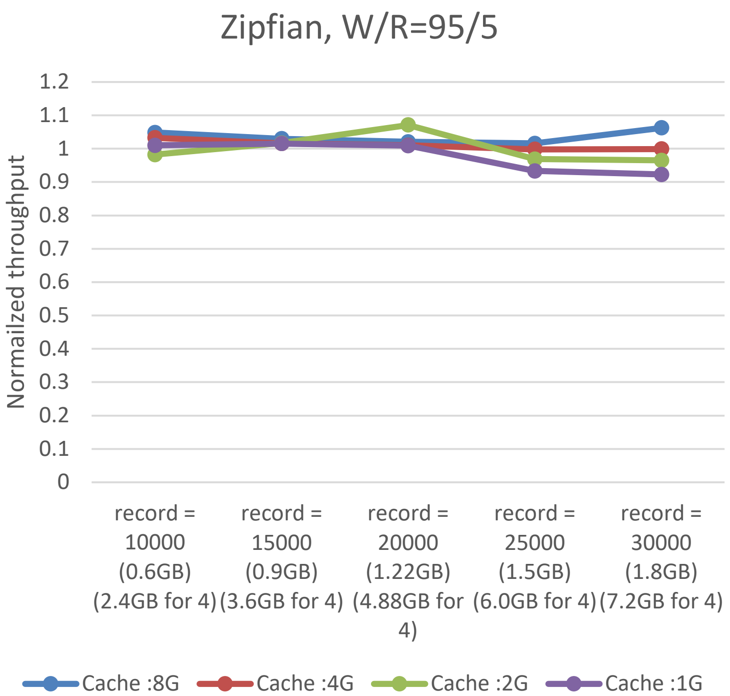
- NVM log buffer size is 128MB, cache size is 4GB

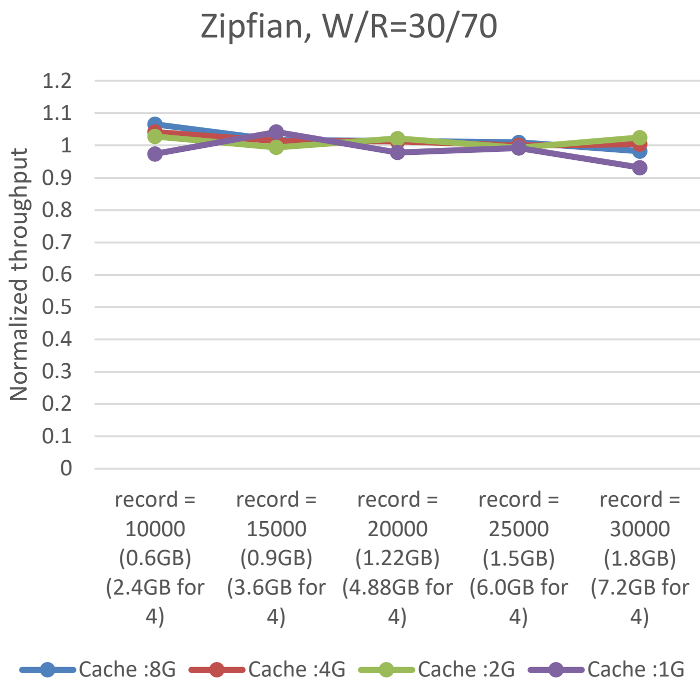




Results of docker container using bind mount

- NVM log buffer size is 2GB
- Baseline: access library from normal processes with the same setting





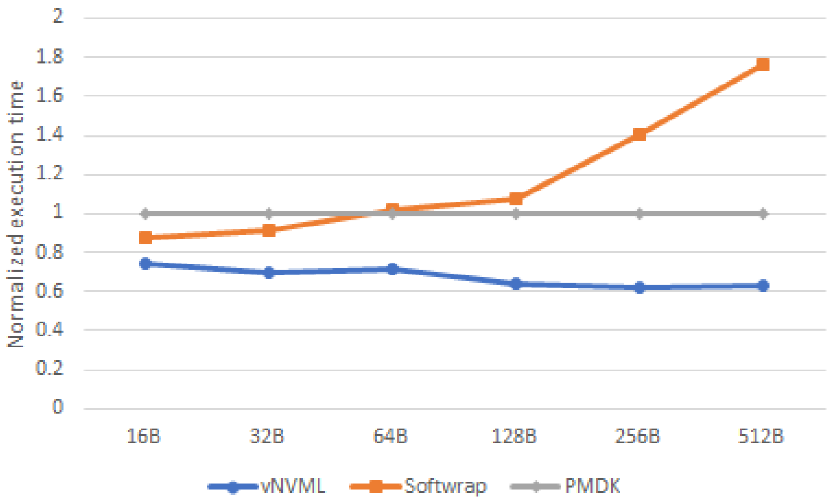
Comparison of other libraries

- We use a microbenchmark to compare three libraries: vNVML, PMDK, and SoftWrAP (MSST'15)
- We allocate a 2GB array in NVM, and write certain amount of data to each 4K page until we have written all pages in the 2GB NVM array

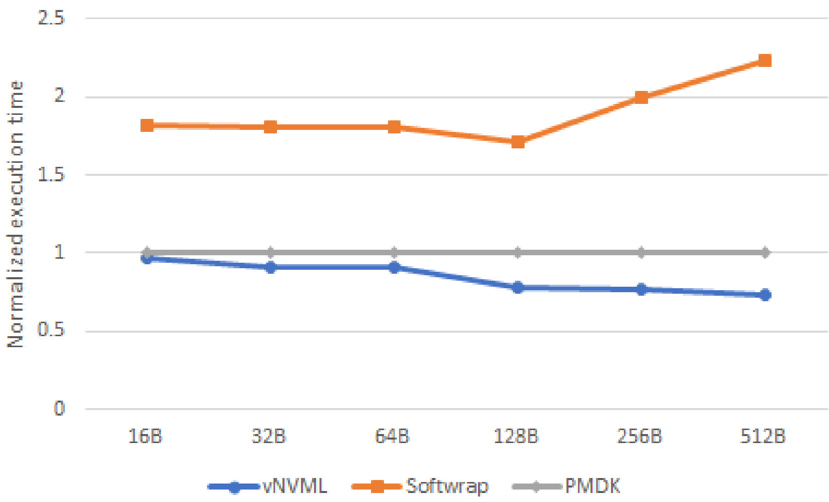
Results

ELECTRICAL & COMPUTER ENGINEERING
TEXAS A&M UNIVERSITY

X-axis stands for the written data of each page; Y-axis is total execution time



Write 2GB NVM array once



Write 2GB NVM array 16 times

Conclusions

- The log buffer size does not affect the performance a lot (less than 10%) when we shrink the size of log buffer from 2GB to 128MB
- The vNVML can provide over 90% throughput compared to that of baseline if the NVM cache system can handle the write traffic well
- The performance between accessed vNVML from normal processes and from docker container has no much difference

Acknowledgements

ELECTRICAL & COMPUTER ENGINEERING
TEXAS A&M UNIVERSITY

- Thank the generous support from Hewlett Packard Enterprise and National Science Foundation through IUCRC (Industry–University Cooperative Research Centers) Program

ELECTRICAL & COMPUTER ENGINEERING

TEXAS A&M UNIVERSITY

Thank You