针对用户态定长申请的内存分配器

星火十六期终审答辩

刘般若

清华大学

2022 年 5 月 22 日

Motivation

- Modern volatile memory allocators over DRAM are quiet optimized.(tcmalloc[1] jemalloc[2])
- However, emerging NVM(non-volatile memory, or persistent memory) applications require allocators to be crash-consistent.
- To achieve that, general-purpose NVM allocator has to use transcation-like method, which devastates its performance.
- For low-level applications, a fast, scalable persistent allocator is in dire need.

[1] https://github.com/google/tcmalloc
[2] https://github.com/jemalloc/jemalloc

Observation

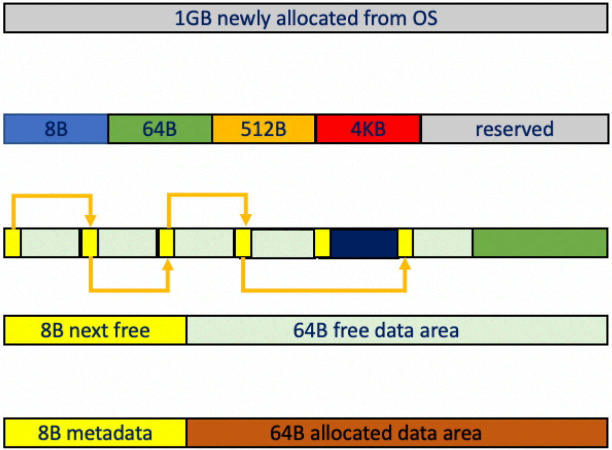
- Since programmers endeavour to cater hardware characteristics, most of alloc requests are confined in a few size types.
- For indexes, such requests are fixed and highly skewed, usually related with values below.
  - 64 Bytes, typical CPU cacheline size
  - 256 Bytes, Optane DIMM write buffer size
  - 4K Bytes, page size for most X86 systems
- Thus, we only need to deal with a few fixed sizes.

User-awared Fixed Size Memory Allocator

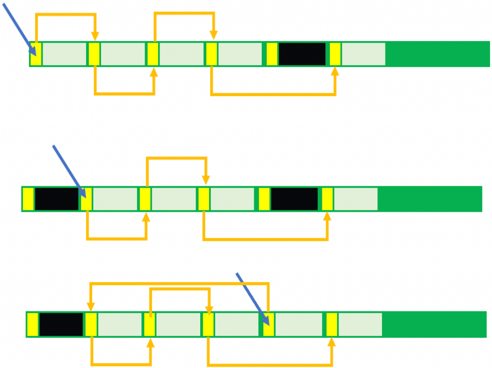
- In this project, we present a memory pool, which accepts a size vector from user-space, provides alloc utility for such sizes, achieving:
- High throughput, low latency
- Near-linear scalability with thread number
- Low internal fragmentation
- Crash consistency (non-volatile)
- Outperform current mainstream persistent allocator. (pool in PMDK, Persistent Memory Development Kit [3])

[3] https://pmem.io/pmdk/

Overview


1GB newly allocated from OS
8B
64B
512B
4KB
reserved
8B next free
64B free data area
8B metadata
64B allocated data area

Alloc and Free



- initial state
- alloc(64)
- free(addr), size is inferred from header

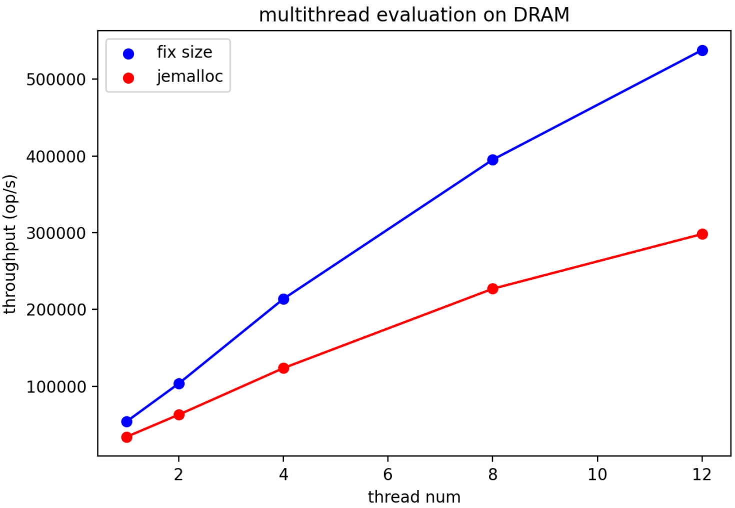
Configuration

OS	Ubuntu 18.04 LTS, 4.15.0-169
CPU	Xeon Gold 5220 @ 2.20GHz, 36 logical cores with hyperthreading
DRAM	2 × 32G DDR4 2933 MHz
NVM	128G Intel Optane 100

Performance as Volatile Allocator

- We compared our allocator with current industrial-level general purpose allocator: jemalloc from Facebook
- Data are randomly generated alloc-free sequence of size in $\{8, 64, 512, 4K, 2M\}$
  About 50000 operations per thread. Borrowing rarely occurs.

Performance as Volatile Allocator



Performance as Persistent Allocator

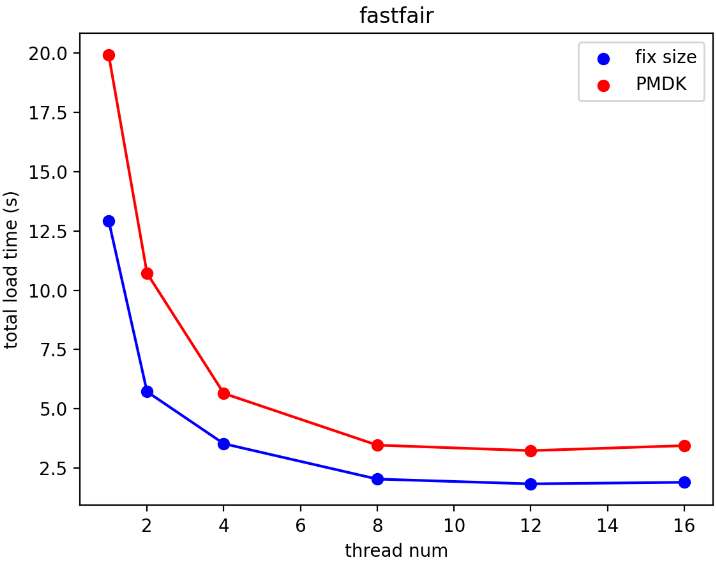
- We compared with PMDK::pool on real workloads.
- We changed allocator of non-volatile index from PDMK to fixed size pool. And test the load phase duration, which consists about $2 * 10^6$ insert operation.[4]
- We tested our project on three indexes, FastFair[5], p-masstree[6] and CCEH[7]

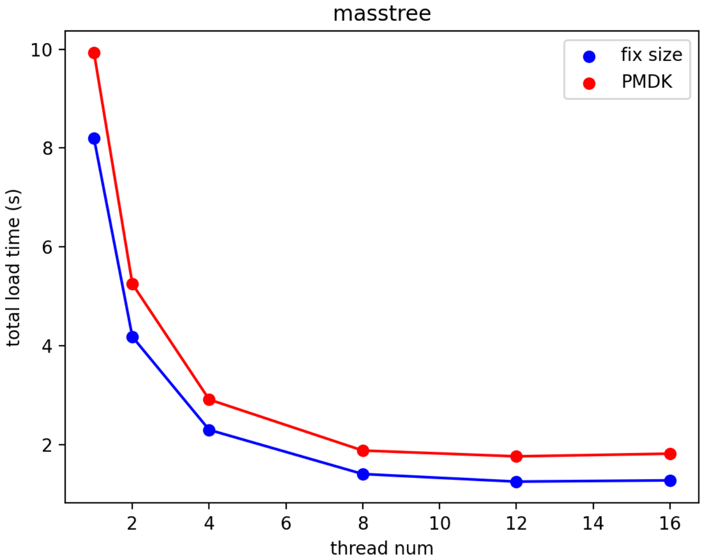
[4] Wang, Qing, et al. "Nap: A Black-Box Approach to NUMA-Aware Persistent Memory Indexes."

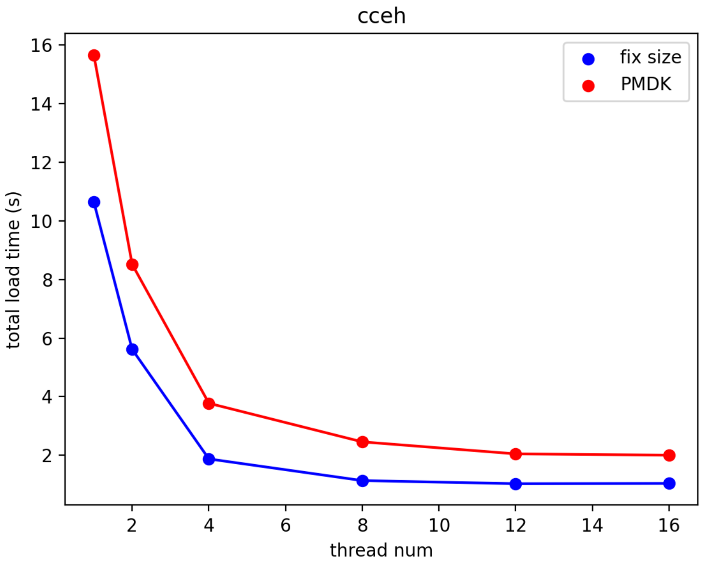
[5] Hwang, Deukyeon, et al. "Endurable Transient Inconsistency in Byte-Addressable Persistent B+-Tree."

[6] Lee, Se Kwon, et al. "Recipe: Converting concurrent DRAM indexes to persistent-memory indexes."

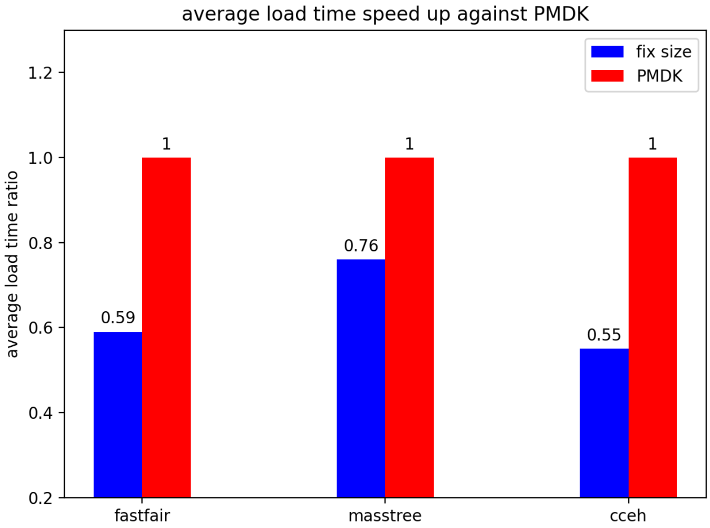
[7] Nam, Moohyeon, et al. "Write-Optimized Dynamic Hashing for Persistent Memory."


fastfair
fix size
PMDK
total load time (s)
thread num
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
2
4
6
8
10
12
14
16


masstree
fix size
PMDK
total load time (s)
thread num
10
8
6
4
2
2
4
6
8
10
12
14
16


cceh
fix size
PMDK
total load time (s)
thread num
16
14
12
10
8
6
4
2
2
4
6
8
10
12
14
16

Result

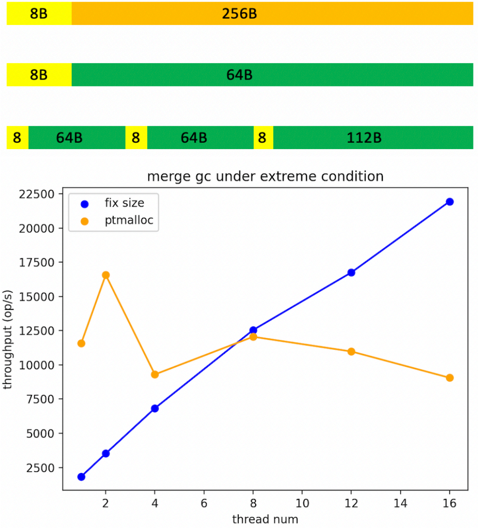

average load time speed up against PMDK
average load time ratio
1.2
1.0
0.8
0.6
0.4
0.2
fix size
PMDK
0.59
1
0.76
1
0.55
1
fastfair
masstree
cceh

Consistency

For a general purpose allocator, it uses refined tricks and techniques to deal with arbitrary sizes. As a result, it is much more complex to persist its structure on NVM.

However, since our goal is inherently simple, our design is much concise, and we can use techniques like write-ahead logging to guarantee consistency.

Reduce Internal Fragmentation



- Through rarely happens, we have invented practical plan for borrowing and lending to ensure correctness. We use header-division or redundant allocation.
- We also use heuristic segmented merge for efficiency and parallelizability under extreme workloads.

Small Size Optimization

- For small allocations i.e. 8 Bytes, using header will introduce about 50% internal fragmentation.
- A self-decode pointer is easy to implement and has decent performance. But linked-list is not cache-favorable. Performance is compromised when there is successive requests for small size.
- Using multi-level bitmap and random magic number checksum. We can fit the available slots in L1 cache, which improves the performance.

Future Plan

- Do more experiments on other NVM applications, i.e. key-value storage, filesystem...
- Optimize recovery strategy, using techniques like checkpoint to make the allocator more durable.
- Hopefully, make it a library that can actually be integrated into other projects.

Thanks!