libvmemcache

Piotr Balcer
<piotr.balcer@intel.com>
Intel® Data Center Group

Problem statement

Local LRU cache

Support for large capacities available with persistent memory (many terabytes per server)

Lightweight, efficient and embeddable

In-memory

Scalable

Existing solutions

In-memory databases tend to rely on malloc() in some form for allocating memory for entries

- Which means allocating anonymous memory

Persistent Memory is exposed by the operating system through normal file-system operations

- Which means allocating byte-addressable PMEM needs to use file memory mapping (fsdax).

We could modify the allocator of an existing in-memory database and be done with it, right? ☺

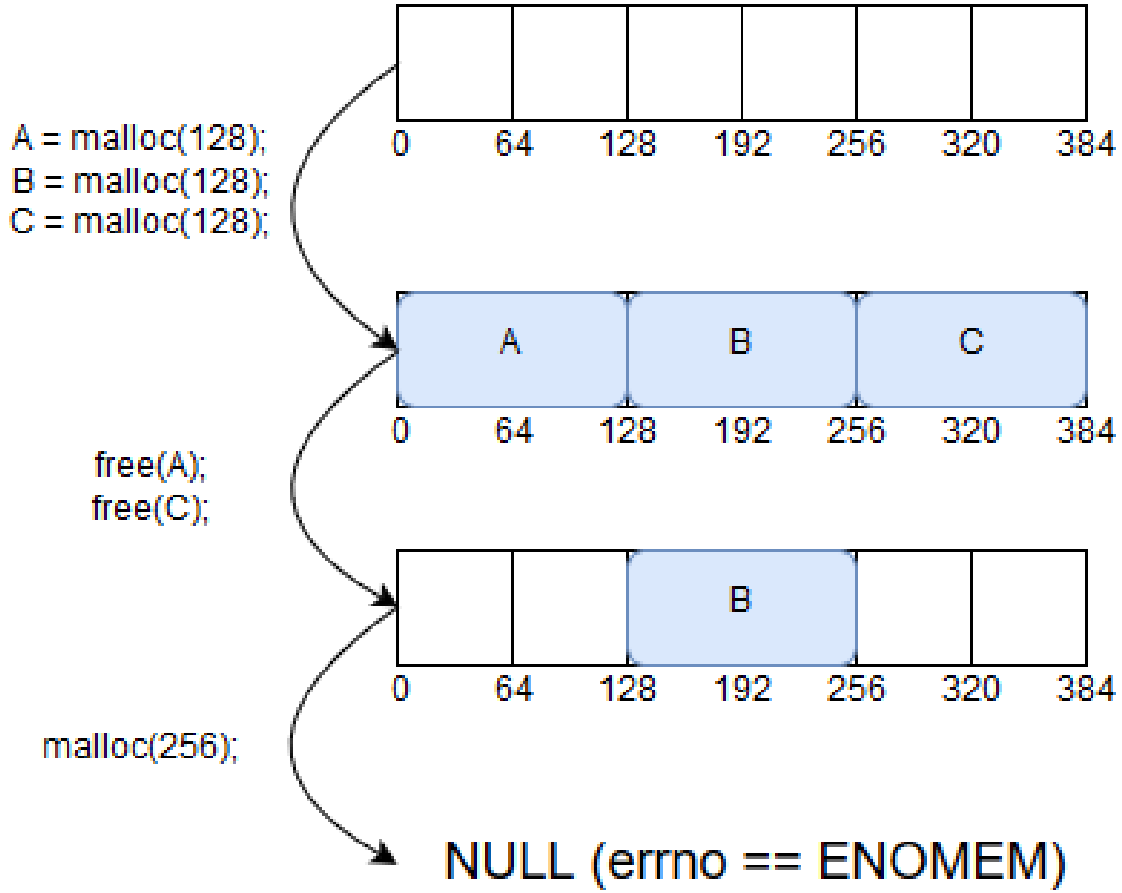
Fragmentation

Manual dynamic memory management a'la dlmalloc/jemalloc/tcmalloc/palloc causes fragmentation

Applications with substantial expected runtime durations need a way to combat this problem

- Compacting GC (Java, .NET)
- Defragmentation (Redis, Apache Ignite)
- Slab allocation (memcached)

Especially so if there's substantial expected variety in allocated sizes

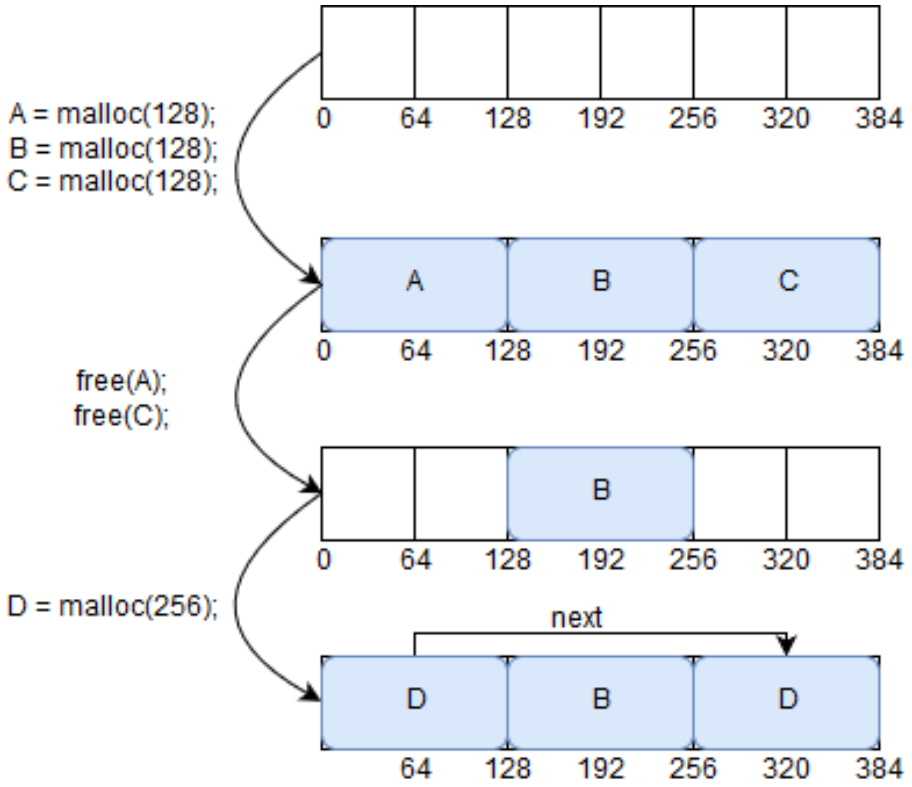


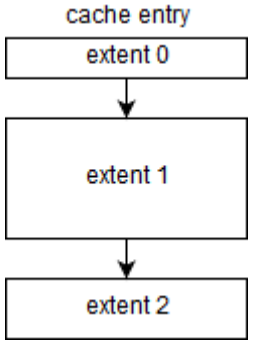
Extent allocation

If fragmentation is unavoidable, and defragmentation/compacting is CPU and memory bandwidth intensive, let's embrace it!

Usually only done in relatively large blocks in file-systems.

But on PMEM, we are no longer restricted by large transfer units (sectors, pages etc)



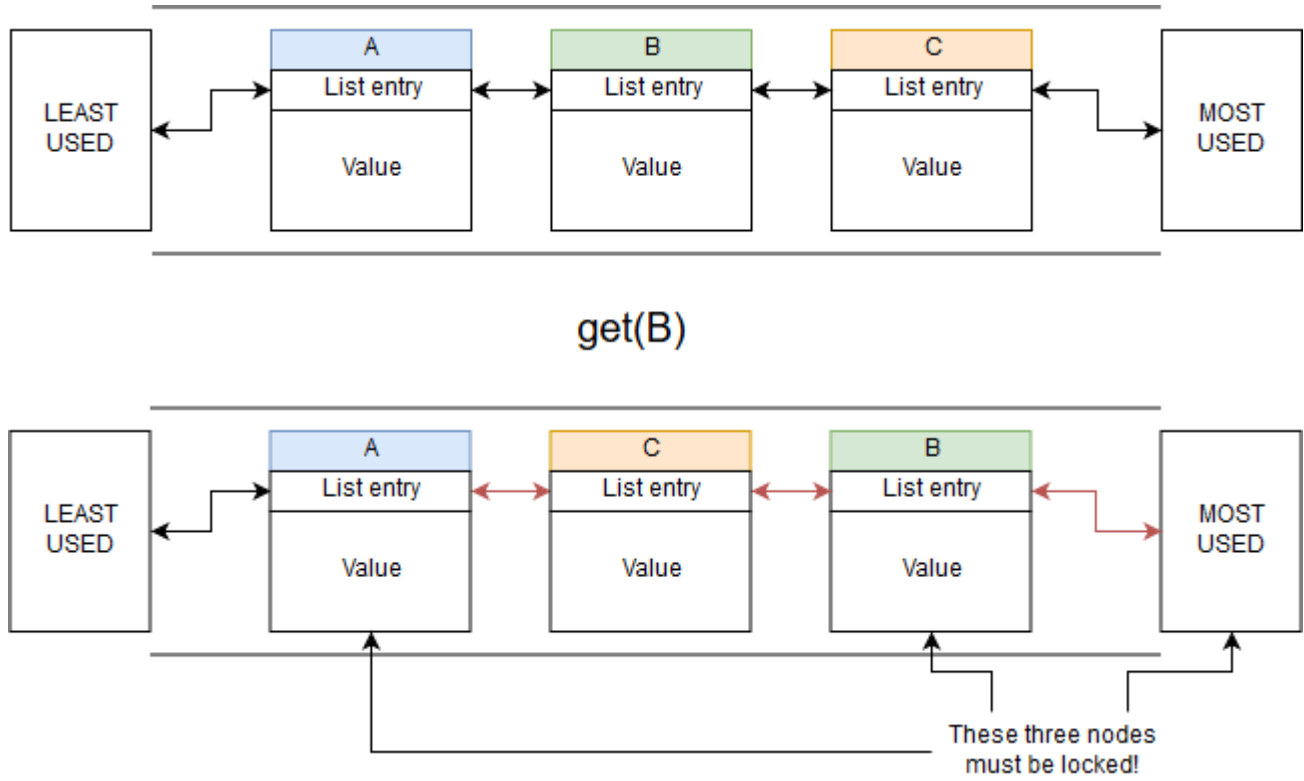


Scalable replacement policy

Performance of libvmemcache was bottlenecked by naïve implementation of LRU based on a doubly-linked list.

With 100st of threads, most of the time of any request was spent waiting on a list lock...

Locking per-node doesn't solve the problem...

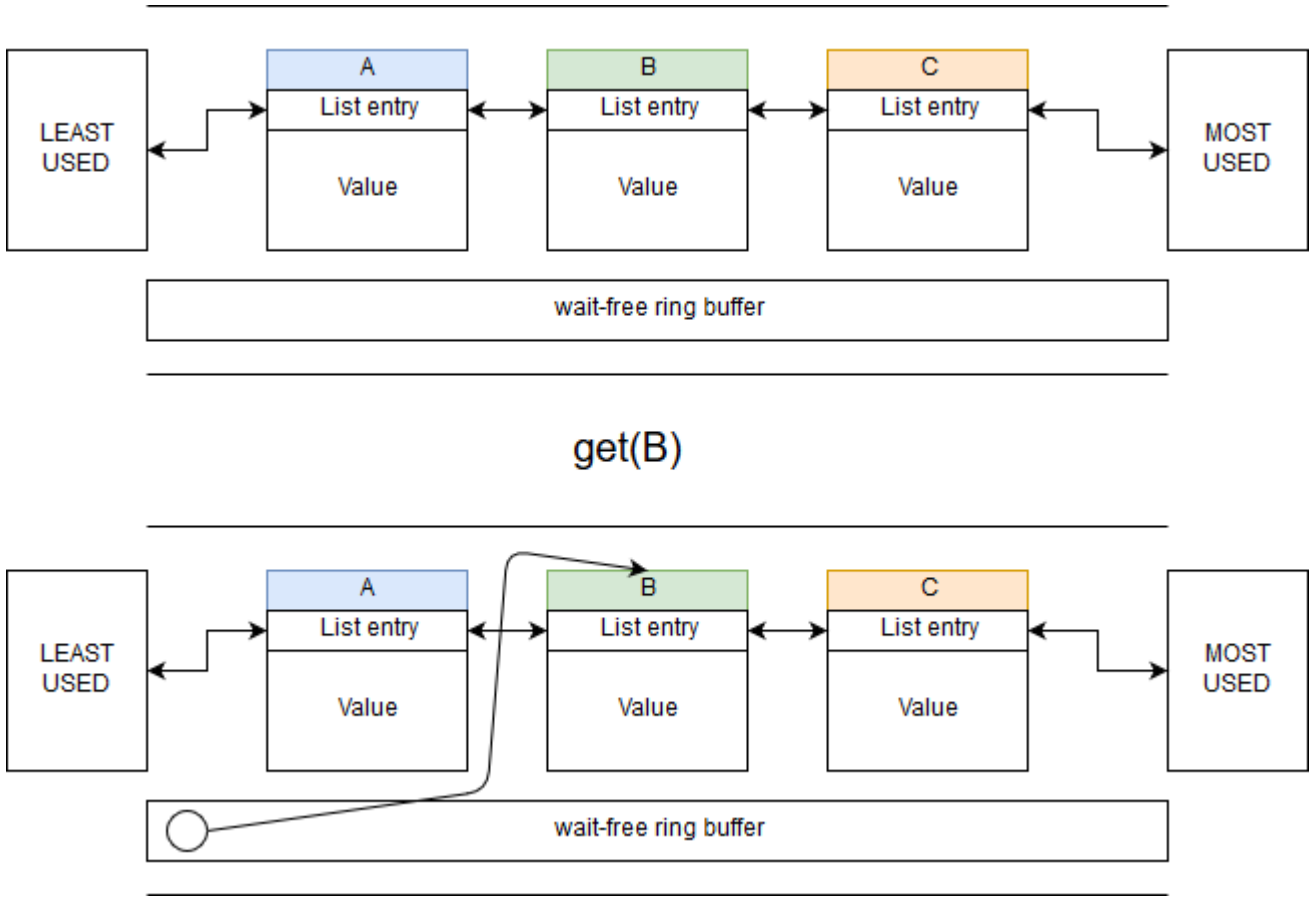


Buffered LRU

Our solution was quite simple.

We've added a wait-free ringbuffer which buffers the list-move operations

This way, the list only needs to get locked during eviction or when the ringbuffer is full.



Lightweight, embeddable, in-memory caching

```
VMEMcache *cache = vmemcache_new("/tmp", VMEMCACHE_MIN_POOL,
VMEMCACHE_MIN_EXTENT, VMEMCACHE_REPLACEMENT_LRU);

const char *key = "foo";
vmemcache_put(cache, key, strlen(key), "bar", sizeof("bar"));

char buf[128];
ssize_t len = vmemcache_get(cache, key, strlen(key),
        buf, sizeof(buf), 0, NULL);

vmemcache_delete(cache);
```

libvmemcache has normal get/put APIs, optional replacement policy, and configurable extent size. Works with terabyte-sized in-memory workloads without a sweat, with very high space utilization. Also works on regular DRAM.

https://github.com/pmem/vmemcache

谢谢

- Questions?

- Questions?