Experience on Building a Lock-Free B+-Tree on Persistent Memory

Tianzheng Wang Xiangpeng Hao

Persistent Multi-Word Compare-and-Swap (PMwCAS)

Easy Lock-Free Indexing in Non-Volatile Memory, ICDE 2018

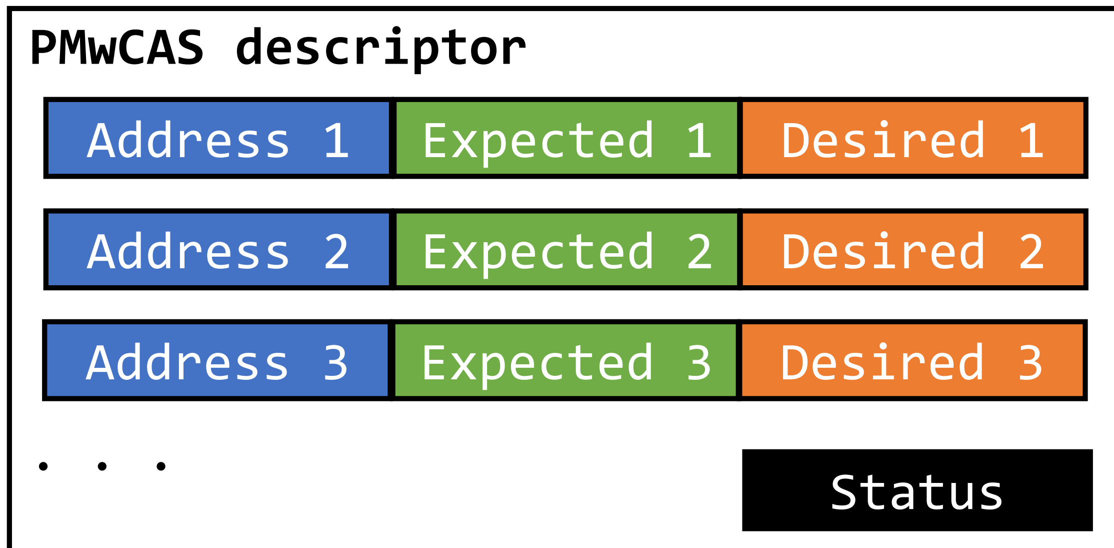
- Atomically changing *multiple* 8-byte words *with persistence guarantee*
  - Either all specified updates succeed, or none of them
- Software-only, lock-free
- Based on a volatile MwCAS design*
  - We made it work on persistent memory, plus
    - Persistence guarantees (with PMDK)
    - Recovery
    - Persistent memory management
- Open sourced at https://github.com/Microsoft/pmwcas

Goal: simplify lock-free programming on PM, maintain good performance

* A Practical Multi-word Compare-and-Swap Operation, DISC 2002

PMwCAS Descriptor

- Application specifies words to change atomically, in a descriptor
  - Following CAS interface for each word
  - Issue (launch) the operation after adding all words
- Final result: either all words changed, or none of them



Fixed size: four words maximum

Lock-Free Doubly-Linked List using Single-Word CAS

```
94  Status CASDList::InsertBefore(DListNode* next, DListNode* node, bool) {
95    if (next == &head_) {
96      return InsertAfter(next, node);
97    }
98
99    DListNode* prev = nullptr;
100   // Persist the payload
101   NVRAM::Flush(node->payload_size, node->GetPayload());
102
103   while (true) {
104     prev = DereferenceNodePointer(&next->prev);
105     // If the guy supposed to be behind me got deleted, fast
106     // forward to its next node and retry
107     auto* next_next = ReadPersist(&next->next);
108     if ((uint64_t)next_next & kNodeDeleted) {
109       next = GetNext(next);
110       prev = CorrectPrev(prev, next);  // using the new next
111       continue;
112     }
113
114     node->prev = (DListNode*)((uint64_t)prev & ~kNodeDeleted);
115     node->next = (DListNode*)((uint64_t)next & ~kNodeDeleted);
116     // Flush node.prev and node.next before installing
117     NVRAM::Flush(sizeof(node->prev) + sizeof(node->next), &node->prev);
118
119     // Install [node] on prev->next
120     DListNode* expected = (DListNode*)((uint64_t)next & ~kNodeDeleted);
121     if (expected == CompareExchange64Ptr(&prev->next, node, expected)) {
122       break;
123     }
124     // Failed, get a new hopefully-correct prev
125     prev = CorrectPrev(prev, next);
126     backoff();
127   }
128   RAW_CHECK(prev, "invalid prev pointer");
129   CorrectPrev(prev, next);
130   return Status::OK();
131 }
```

```
244 DListNode* CASDList::CorrectPrev(DListNode* prev, DListNode* node) {
245   DListNode* last_link = nullptr;
246   while (true) {
247     auto* link1 = ReadPersist(&node->prev);
248     if ((uint64_t)link1 & kNodeDeleted) {
249       break;
250     }
251
252     DListNode* prev_cleared = (DListNode*)((uint64_t)prev & ~kNodeDeleted);
253     auto* prev_next = ReadPersist(&prev_cleared->next);
254     if ((uint64_t)prev_next & kNodeDeleted) {
255       if (last_link) {
256         MarkNodePointer(&prev_cleared->prev);
257         auto* desired = (DListNode*)(((uint64_t)prev_next & ~kNodeDeleted) | kDirtyFlag);
258         CompareExchange64Ptr(&last_link->next, desired, prev);
259         prev = last_link;
260         last_link = nullptr;
261         continue;
262       }
263       prev_next = ReadPersist(&prev_cleared->prev);
264       prev = prev_next;
265       continue;
266     }
267
268     if (prev_next != node) {
269       last_link = prev_cleared;
270       prev = prev_next;
271       continue;
272     }
273
274     DListNode* p = (DListNode*)(((uint64_t)prev & ~kNodeDeleted) | kDirtyFlag);
275     if (link1 == CompareExchange64Ptr(&node->prev, p, link1)) {
276       auto* prev_cleared_prev = ReadPersist(&prev_cleared->prev);
277       if ((uint64_t)prev_cleared_prev & kNodeDeleted) {
278         continue;
279       }
280       break;
281     }
282     backoff();
283   }
284   return prev;
285 }
```

Lock-Free Doubly-Linked List using PMwCAS

```
320 Status MwCASDList::InsertBefore(DListNode* next, DListNode* node, bool prot) {
321   if (next == &head_) {
322     return InsertAfter(next, node, prot);
323   }
324   EpochGuard guard(GetEpoch(), !prot);
325
326   // Persist the payload
327   NVRAM::Flush(node->payload_size, node->GetPayload());
328
329   while (true) {
330     auto* prev = ResolveNodePointer(&next->prev, true);
331
332     auto* next_next = ResolveNodePointer(&next->next, true);
333     if (((uint64_t)prev & kNodeDeleted) || (uint64_t)next_next & kNodeDeleted) {
334       next = GetNext(next, true);
335       continue;
336     }
337
338     auto* prev_next = ResolveNodePointer(&prev->next, true);
339     if ((uint64_t)prev_next & kNodeDeleted) {
340       continue;
341     }
342
343     node->next = next;
344     node->prev = prev;
345     // Persist node
346     NVRAM::Flush(sizeof(node->prev) + sizeof(node->next), &node->prev);
347
348     auto* desc = descriptor_pool_->AllocateDescriptor();
349     desc->AddEntry((uint64_t*)&node->prev->next, (uint64_t)node->next, (uint64_t)node);
350     desc->AddEntry((uint64_t*)&next->prev, (uint64_t)node->prev, (uint64_t)node);
351     if(desc->MwCAS()) {
352       return Status::OK();
353     }
354   }
355   return Status::OK();
356 }
```

Application specifies:

- Which words to change (address)
- What are the expected values
- What are the new desired values
- + more (later)

Easier implementation with PMwCAS

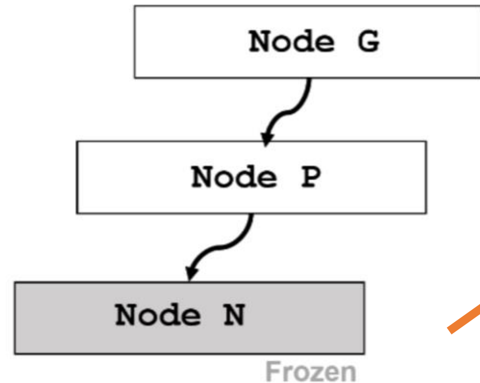
PMwCAS in Action: Building the BzTree*

- A B+-tree variant by Microsoft Research using PMwCAS
  - Data in leaf nodes
  - Internal nodes only guides search
  - Copy-on-write for internal nodes: always sorted (binary search)
  - Unsorted leaf nodes (linear search), in-place update allowed
- Designed before Optane DCPMM came out
  - Targeted more on NVDIMM-N in original paper
- What we did
  - Faithful implementation following the paper, in order to evaluate it
  - In collaboration with SAP and TU Dresden

* BzTree: A High-Performance Latch-free Range Index for Non-Volatile Memory, VLDB 2018

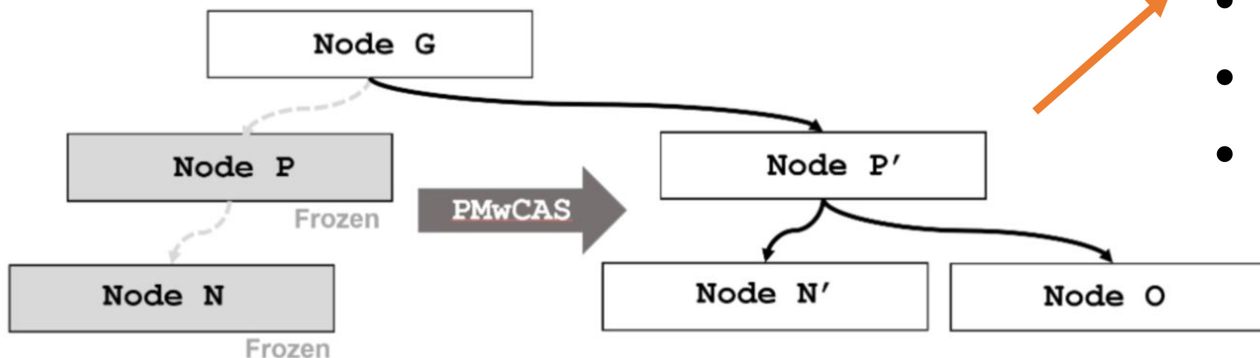
BzTree Split

Step 1: “freeze” node
(preparation phase)



1-word PMwCAS: toggle “frozen” bit

Step 2: install new nodes (installation phase)



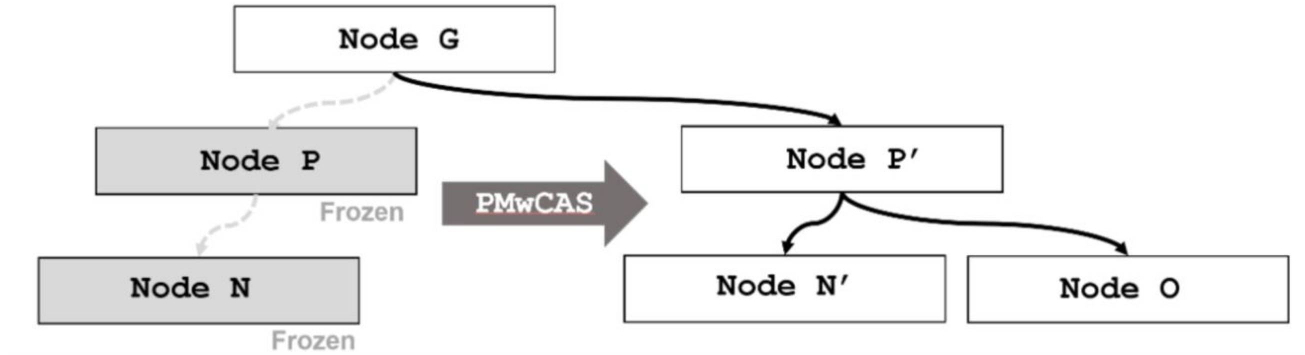
3-word PMwCAS:

- Freeze parent P
- Update grandparent to point to new P’
- Ensure grandparent’s status didn’t change

Sounds simple, “how hard can it be?”

Warming-up Problem: Persistent Memory Leaks

- New nodes allocated
  - Not installed until PMwCAS succeeds
- Old nodes need to be freed
  - Only if PMwCAS succeeds



Solution: PMwCAS got it covered

PMwCAS descriptor

G.P’	P	P’	FreeOne
Address 2	Expected 2	Desired 2	MemPolicy 2
Address 3	Expected 3	Desired 3	MemPolicy 3

Status

Free old (new) value if succeeded (failed)

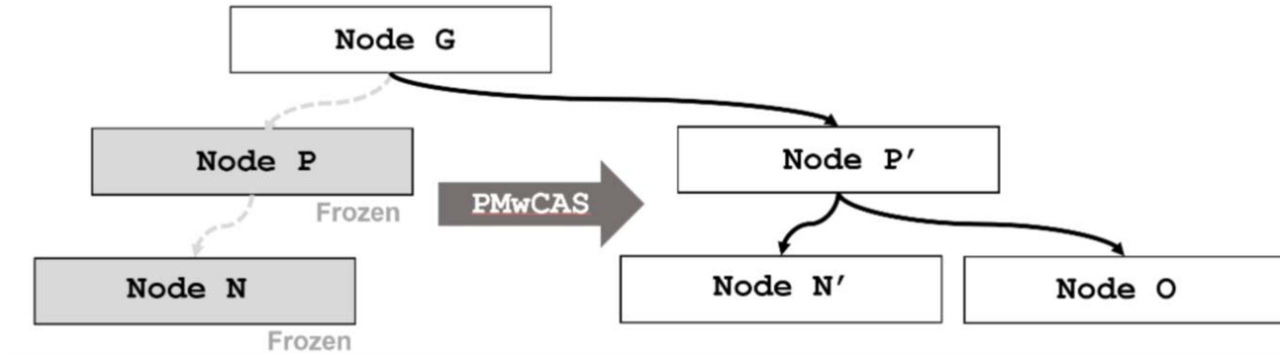
Memory Policies in Action

```
1541     auto *pd = GetPMWCASPool()->AllocateDescriptor();
1542     // TODO(hao): should implement a cascading memory recycle callback
1543     pd->ReserveAndAddEntry(reinterpret_cast<uint64_t *>(pmwcas::Descriptor::kAllocNullAddress),
1544                            reinterpret_cast<uint64_t>(nullptr),
1545                            pmwcas::Descriptor::kRecycleOnRecovery);
1546     pd->ReserveAndAddEntry(reinterpret_cast<uint64_t *>(pmwcas::Descriptor::kAllocNullAddress),
1547                            reinterpret_cast<uint64_t>(nullptr),
1548                            pmwcas::Descriptor::kRecycleOnRecovery);
1549     pd->ReserveAndAddEntry(reinterpret_cast<uint64_t *>(pmwcas::Descriptor::kAllocNullAddress),
1550                            reinterpret_cast<uint64_t>(nullptr),
1551                            pmwcas::Descriptor::kRecycleOnRecovery);
1552     uint64_t *ptr_r = pd->GetNewValuePtr(0);
1553     uint64_t *ptr_l = pd->GetNewValuePtr(1);
1554     uint64_t *ptr_parent = pd->GetNewValuePtr(2);
```

- Reserve entry first in descriptor for safe memory ownership transfer
  - posix_memalign-like interface
  - PMwCAS “owns” the memory temporarily

A Real Problem: Tree Growth

- This is easy for a single level



- But split can propagate up, and grow the tree
  - Descriptor size scales with data size!
- Alternative: split and propagate up one level at a time
- Current: Limit and pre-allocate space in descriptor
  - Had to modify PMwCAS descriptor size

Not so simple, eh?

Beyond “Z”

- “Easy” vs. “hard”
  - Up to the design even with a library that can potentially simplify things
  - Need careful consideration about interactions with PM programming library
- “Something-after-Z” Tree
  - Basic principles and more insights covered in Lucas’ talk yesterday
  - No/less CoW
  - Careful interactions with persistent memory allocator
- Implementations
  - BzTree: https://github.com/wangtzh/bztree/
  - PMwCAS with PMDK: https://github.com/Microsoft/pmwcas
  - Pull requests/bug report/fixes welcome

Thank you!