Understanding and Detecting Deep Memory Persistency Bugs in NVM Programs with DeepMC

Benjamin Reidys  
Jian Huang

University of Illinois Urbana-Champaign
Adoption of Intel Optane persistent memory picks up in 2020

Intel provides an update on adoption trends for its Optane persistent memory modules, showing that SAP HANA, virtualization and high-performance computing are top use cases.
Non-Volatile Memory is a Promising Technology

Adoption of Intel Optane persistent memory picks up in 2020

Intel provides an update on adoption trends for its Optane persistent memory modules, showing that SAP HANA, virtualization and high-performance computing are top use cases.

By Carol Sliwa  
Published: 30 Oct 2020
Non-Volatile Memory is a Promising Technology

Adoption of Intel Optane persistent memory picks up in 2020

Intel provides an update on adoption trends for its Optane persistent memory modules, showing that SAP HANA, virtualization and high-performance computing are top use cases.

By Carol Sliwa
Published: 30 Oct 2020
Adoption of Intel Optane persistent memory picks up in 2020

Intel provides an update on adoption trends for its Optane persistent memory modules, showing that SAP HANA, virtualization and high-performance computing are top use cases.
Using Non-Volatile Memory is Not Easy

Volatile Caches

NVM
Using Non-Volatile Memory is Not Easy

Volatile Processor Caches

CPU

Volatile Caches

NVM
Using Non-Volatile Memory is Not Easy

Volatile Caches

Volatile Processor Caches

Out-of-Order Execution

NVM
Using Non-Volatile Memory is Not Easy

Volatile Processor Caches

Out-of-Order Execution

Persistence/Performance Tradeoff
Persistency Models for Non-Volatile Memory

Strict Persistency

Writes must be persisted in program order!
Persistency Models for Non-Volatile Memory

Strict Persistency

 Writes can be concurrent within an epoch!

Epoch Persistency
Persistency Models for Non-Volatile Memory

Strict Persistency

Epoch Persistency

Strand Persistency

Writes can be concurrent within and across strands!
A
clwb(A)
mfence
B
clwb(B)
mfence
C

Dependency

False dependency

 Writes must be persisted in program order!
Persistency Models – Strict Persistency

A
clwb(A)
mfence

B
clwb(B)
mfence

C

Writes must be persisted in program order!
Persistency Models – Strict Persistency

A
clwb(A)
mfence

B
clwb(B)
mfence

C

Insert barriers between operations!

Dependency

False dependency

Writes must be persisted in program order!
Persistency Models – Strict Persistency

A
clwb(A)
mfence

B
clwb(B)
mfence

C

Insert barriers between operations!

Strong Persistency Guarantees

 Writes must be persisted in program order!
Persistency Models – Strict Persistency

Writes must be persisted in program order!
Persistency Models – Strict Persistency

A
\texttt{clwb(A)}
\texttt{mfence}

B
\texttt{clwb(B)}
\texttt{mfence}

C

Insert barriers between operations!

Strong Persistency Guarantees

Easy to Use

Low Performance!

Writes must be persisted in program order!
begin_epoch
A
B
end_epoch
begin_epoch
C
end_epoch

Writes can be concurrent within an epoch!
Persistency Models – Epoch Persistency

Writes can be concurrent within an epoch!
Persistency Models – Epoch Persistency

```
begin_epoch
A
B
end_epoch
begin_epoch
C
end_epoch
```

Ordering enforced between epochs!

Writes can be concurrent within an epoch!
Persistency Models – Epoch Persistency

- begin_epoch
  - A
  - B
- end_epoch
- begin_epoch
  - C
- end_epoch

Concurrency within an epoch!

Ordering enforced between epochs!

Writes can be concurrent within an epoch!
Persistency Models – Epoch Persistency

begin_epoch
A
B
end_epoch
begin_epoch
C
end_epoch

Concurrency within an epoch!
Ordering enforced between epochs!

Relaxed Persistency Guarantees

Writes can be concurrent within an epoch!
Persistency Models – Epoch Persistency

- **begin_epoch**
  - A
  - B
- **end_epoch**
- **begin_epoch**
  - C
- **end_epoch**

Concurrence within an epoch!

Ordering enforced between epochs!

Relaxed Persistency Guarantees

Enables Higher Concurrency

Writes can be concurrent within an epoch!
Persistency Models – Epoch Persistency

**begin_epoch**

A

B

**end_epoch**

**begin_epoch**

C

**end_epoch**

Concurrence within an epoch!

Ordering enforced between epochs!

Relaxed Persistency Guarantees

Improved Performance!

**Writes can be concurrent within an epoch!**
begin_strand
B
begin_strand
A
barrier
C

Writes can be reordered within and across strands!
Persistency Models – Strand Persistency

```
begin_strand
B
begin_strand
A
barrier
C
```

Writes can be reordered within and across strands!
Persistency Models – Strand Persistency

begin_strand
B
begin_strand
A
barrier
C

A

B

C

 Writes can be reordered within and across strands!
Persistency Models – Strand Persistency

begin_strand
B
begin_strand
A
barrier
C

Order within strand!

A

B

C

Writes can be reordered within and across strands!
Persistency Models – Strand Persistency

begin_strand

No false dependencies!

begin_strand

Order within strand!

<C

barrier

A

B

C

Writes can be reordered within and across strands!
writes can be reordered within and across strands!
Persistency Models – Strand Persistency

```
begin_strand A
begin_strand B
barrier C
```

No false dependencies!
Order within strand!

Highest Possible Performance
Difficult to program correctly

Writes can be reordered within and across strands!
Implementing Persistency Models Properly is Challenging

- Strict Persistency
- Epoch Persistency
- Strand Persistency
Implementing Persistency Models Properly is Challenging

- Strict Persistency
- Epoch Persistency
- Strand Persistency

Guide developers in NVM programming

NVM Programs
Implementing Persistency Models Properly is Challenging

Strict Persistency

Epoch Persistency

Strand Persistency

Guide developers in NVM programming

Does my program correctly follow the model specifications?

NVM Programs
## Understanding Persistency Bugs in NVM Programs

<table>
<thead>
<tr>
<th>NVM Library</th>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMDK</td>
<td>bstree_map.c</td>
</tr>
<tr>
<td></td>
<td>rbtree_map.c</td>
</tr>
<tr>
<td></td>
<td>pminvaders.c</td>
</tr>
<tr>
<td></td>
<td>obj_pmemlog.c</td>
</tr>
<tr>
<td></td>
<td>hash_map.c</td>
</tr>
<tr>
<td>PMFS</td>
<td>journal.c</td>
</tr>
<tr>
<td></td>
<td>symlink.c</td>
</tr>
<tr>
<td></td>
<td>xips.c</td>
</tr>
<tr>
<td></td>
<td>files.c</td>
</tr>
<tr>
<td>NVM-Direct</td>
<td>nvm_region.c</td>
</tr>
<tr>
<td></td>
<td>nvm_heap.c</td>
</tr>
</tbody>
</table>
### Understanding Persistency Bugs in NVM Programs

<table>
<thead>
<tr>
<th>NVM Library</th>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMDK</td>
<td>btree_map.c</td>
</tr>
<tr>
<td></td>
<td>rbtree_map.c</td>
</tr>
<tr>
<td></td>
<td>rbtree_map.c</td>
</tr>
<tr>
<td></td>
<td>pminvaders.c</td>
</tr>
<tr>
<td></td>
<td>pminvaders.c</td>
</tr>
<tr>
<td></td>
<td>obj_pmemlog.c</td>
</tr>
<tr>
<td></td>
<td>hash_map.c</td>
</tr>
<tr>
<td>PMFS</td>
<td>journal.c</td>
</tr>
<tr>
<td></td>
<td>symlink.c</td>
</tr>
<tr>
<td></td>
<td>xips.c</td>
</tr>
<tr>
<td></td>
<td>files.c</td>
</tr>
<tr>
<td>NVM-Direct</td>
<td>nvm_region.c</td>
</tr>
<tr>
<td></td>
<td>nvm_heap.c</td>
</tr>
</tbody>
</table>

Select programs from open source framework PMDK, PMFS, and NVM-Direct
## Understanding Persistency Bugs in NVM Programs

<table>
<thead>
<tr>
<th>NVM Library</th>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMDK</td>
<td>btree_map.c</td>
</tr>
<tr>
<td></td>
<td>rbtree_map.c</td>
</tr>
<tr>
<td></td>
<td>pminvaders.c</td>
</tr>
<tr>
<td></td>
<td>obj_pmemlog.c</td>
</tr>
<tr>
<td></td>
<td>hash_map.c</td>
</tr>
<tr>
<td>PMFS</td>
<td>journal.c</td>
</tr>
<tr>
<td></td>
<td>symlink.c</td>
</tr>
<tr>
<td></td>
<td>xips.c</td>
</tr>
<tr>
<td></td>
<td>files.c</td>
</tr>
<tr>
<td>NVM-Direct</td>
<td>nvm_region.c</td>
</tr>
<tr>
<td></td>
<td>nvm_heap.c</td>
</tr>
</tbody>
</table>

Select programs from open source framework PMDK, PMFS, and NVM-Direct

Manually study 19 representative persistency bugs
Understanding Persistency Bugs in NVM Programs

<table>
<thead>
<tr>
<th>NVM Library</th>
<th>File</th>
<th>Location (#Line)</th>
<th>File Location</th>
<th>Bug Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMDK</td>
<td>btree_map.c</td>
<td>201</td>
<td>EP</td>
<td>[V] Modify tree node without making it durable</td>
</tr>
<tr>
<td></td>
<td>rbtree_map.c</td>
<td>197, 231</td>
<td>EP</td>
<td>[P] Log unmodified fields of a tree node</td>
</tr>
<tr>
<td></td>
<td>rbtree_map.c</td>
<td>379</td>
<td>EP</td>
<td>[V] Modified object not made durable</td>
</tr>
<tr>
<td></td>
<td>pminvaders.c</td>
<td>256, 301</td>
<td>EP</td>
<td>[P] Durable transaction without persistent writes</td>
</tr>
<tr>
<td></td>
<td>pminvaders.c</td>
<td>246, 143</td>
<td>EP</td>
<td>[P] Flush unmodified fields of an object</td>
</tr>
<tr>
<td></td>
<td>obj_pmemlog.c</td>
<td>91</td>
<td>LIB</td>
<td>[V] Multiple epochs writing to different fields of an object</td>
</tr>
<tr>
<td></td>
<td>hash_map.c</td>
<td>120, 264</td>
<td>EP</td>
<td>[V] Multiple epochs writing to different fields of an object</td>
</tr>
<tr>
<td>PMFS</td>
<td>journal.c</td>
<td>632</td>
<td>LIB</td>
<td>[P] Flush redundant data when committing</td>
</tr>
<tr>
<td></td>
<td>symlink.c</td>
<td>38</td>
<td>LIB</td>
<td>[V] Missing persistent barrier</td>
</tr>
<tr>
<td></td>
<td>xips.c</td>
<td>207, 262</td>
<td>LIB</td>
<td>[P] Flush the same buffer multiple times</td>
</tr>
<tr>
<td></td>
<td>files.c</td>
<td>232</td>
<td>LIB</td>
<td>[P] Flush unmodified object</td>
</tr>
<tr>
<td>NVM-Direct</td>
<td>nvm_region.c</td>
<td>614, 933</td>
<td>LIB</td>
<td>[V] Missing persist barrier between epoch transactions</td>
</tr>
<tr>
<td></td>
<td>nvm_heap.c</td>
<td>1965</td>
<td>LIB</td>
<td>[P] Redundant flushes of persistent object</td>
</tr>
</tbody>
</table>
## Understanding Persistency Bugs in NVM Programs

We analyze each bug and discover they fall into two categories: Model Violations [V] or Performance Bugs [P].

<table>
<thead>
<tr>
<th>NVM Library</th>
<th>File</th>
<th>Location (#Line)</th>
<th>File Location</th>
<th>Bug Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMDK</td>
<td>btree_map.c</td>
<td>201</td>
<td>EP</td>
<td>[V] Modify tree node without making it durable</td>
</tr>
<tr>
<td></td>
<td>rbtree_map.c</td>
<td>197, 231</td>
<td>EP</td>
<td>[P] Log unmodified fields of a tree node</td>
</tr>
<tr>
<td></td>
<td>rbtree_map.c</td>
<td>379</td>
<td>EP</td>
<td>[V] Modified object not made durable</td>
</tr>
<tr>
<td></td>
<td>pminvaders.c</td>
<td>256, 301</td>
<td>EP</td>
<td>[P] Durable transaction without persistent writes</td>
</tr>
<tr>
<td></td>
<td>pminvaders.c</td>
<td>246, 143</td>
<td>EP</td>
<td>[P] Durable transaction without persistent writes</td>
</tr>
<tr>
<td></td>
<td>obj_pmemlog.c</td>
<td>91</td>
<td>LIB</td>
<td>[V] Multiple epochs writing to different fields of an object</td>
</tr>
<tr>
<td></td>
<td>hash_map.c</td>
<td>120, 264</td>
<td>EP</td>
<td>[V] Multiple epochs writing to different fields of an object</td>
</tr>
<tr>
<td>PMFS</td>
<td>journal.c</td>
<td>632</td>
<td>LIB</td>
<td>[P] Flush redundant data when committing</td>
</tr>
<tr>
<td></td>
<td>symlink.c</td>
<td>38</td>
<td>LIB</td>
<td>[V] Missing persistent barrier</td>
</tr>
<tr>
<td></td>
<td>xips.c</td>
<td>207, 262</td>
<td>LIB</td>
<td>[P] Flush the same buffer multiple times</td>
</tr>
<tr>
<td></td>
<td>files.c</td>
<td>232</td>
<td>LIB</td>
<td>[P] Flush unmodified object</td>
</tr>
<tr>
<td>NVM-Direct</td>
<td>nvm_region.c</td>
<td>614, 933</td>
<td>LIB</td>
<td>[V] Missing persist barrier between epoch transactions</td>
</tr>
<tr>
<td></td>
<td>nvm_heap.c</td>
<td>1965</td>
<td>LIB</td>
<td>[P] Redundant flushes of persistent object</td>
</tr>
</tbody>
</table>
Classifying Persistency Bugs in NVM Programs

Persistency Model Violations

- Semantic Mismatch
- Unflushed/Unlogged Writes
- Missing Persist Barrier
Classifying Persistency Bugs in NVM Programs

Persistency Model Violations
- Semantic Mismatch
- Unflushed/Unlogged Writes
- Missing Persist Barrier

Performance Bugs
- Flushing Unmodified Data
- Redundant Write-backs of Data
- Durable Transactions without Updates
Classifying Persistency Bugs in NVM Programs

Persistency Model Violations
- Semantic Mismatch
- Unflushed/Unlogged Writes
- Missing Persist Barrier

Performance Bugs
- Flushing Unmodified Data
- Redundant Write-backs of Data
- Durable Transactions without Updates
Persistency Model Violations: Semantic Mismatch

```c
static int create_buckets (PMEMobjpool *pop, void *ptr, void *arg) {
    struct buckets *b = (struct buckets *) ptr;
    b->nbuckets = * ((size_t *) arg);
    pmemobj_memset_persist (pop, &b->bucket, 0,
                            b->nbuckets * sizeof (b->bucket[0]));
    pmemobj_persist (pop, &b->nbuckets, sizeof (b->nbuckets));
    return 0;
}
```

hashmap from PMDK using strict persistency
Persistency Model Violations: Semantic Mismatch

```c
static int create_buckets (PMEMobjpool *pop, void *ptr, void *arg) {
    struct buckets *b = (struct buckets *) ptr;
    b->nbuckets = *(size_t *) arg;
    pmemobj_memset_persist (pop, &b->bucket, 0,
                            b->nbuckets * sizeof (b->bucket[0]));
    pmemobj_persist (pop, &b->nbuckets, sizeof (b->nbuckets));
    return 0;
}
```

hashmap from PMDK using strict persistency
Persistency Model Violations: Semantic Mismatch

```c
static int create_buckets (PMEMobjpool *pop, void *ptr, void *arg) {
    struct buckets *b = (struct buckets *) ptr;
    b->nbuckets = * ((size_t *) arg);
    pmemobj_memset_persist (pop, &b->bucket, 0,
                           b->nbuckets * sizeof (b->bucket[0]));
    pmemobj_persist (pop, &b->nbuckets, sizeof (b->nbuckets));
    return 0;
}
```

hashmap from PMDK using strict persistency

nbuckets initialized on line 3

nbuckets is not persisted until line 6
Persistency Model Violations: Semantic Mismatch

1. static int `create_buckets` (PMEMobjpool *pop, void *ptr, void *arg) {
2.     struct buckets *b = (struct buckets *) ptr;
3.     b->nbuckets = * ((size_t *) arg);
4.     pmemobj_memset_persist (pop, &b->bucket, 0,
5.         b->nbuckets * sizeof (b->bucket[0]));
6.     pmemobj_persist (pop, &b->nbuckets, sizeof (b->nbuckets));
7.     return 0;
8. }

hashmap from PMDK using strict persistency

nbuckets initialized on line 3

nbuckets is not persisted until line 6

Strict persistency requires persists to occur in program order!
Persistency Model Violations: Semantic Mismatch

1. static int create_buckets (PMEMobjpool *pop, void *ptr, void *arg) {
2.   struct buckets *b = (struct buckets *) ptr;
3.   b->nbuckets = * ((size_t *) arg);
4.   pmemobj_memset_persist (pop, &b->bucket, 0,
5.                        b->nbuckets * sizeof (b->bucket[0]));
6.   pmemobj_persist (pop, &b->nbuckets, sizeof (b->nbuckets));
7.   return 0;
8. }

Crash between lines 4 and 6 results in inconsistency!

hashmap from PMDK using strict persistency
Persistency Model Violations: Unflushed/Unlogged Writes

```
1 static struct tree_map_node *
2 btree_map_create_split_node (struct tree_map_node *node,
3                        struct tree_map_node_item *m) {
4               ..........  
5
6       node->items[c - 1] = EMPTY_ITEM;
7       ..........  
8       return 0;
9 } // This function is executed in a transaction.
```

btree_map from PMDK
using epoch persistency
Persistency Model Violations: Unflushed/Unlogged Writes

```c
1 static struct tree_map_node *
2 btree_map_create_split_node (struct tree_map_node *node,
3                              struct tree_map_node _item *m) {
4       ........
5
6       node->items[c - 1] = EMPTY_ITEM;
7       ........
8       return 0;
9   } // This function is executed in a transaction.
```

- `items` is not logged in the transaction.
Persistency Model Violations: Unflushed/Unlogged Writes

```
static struct tree_map_node *
btree_map_create_split_node (struct tree_map_node *node,
    struct tree_map_node _item *m) {

        ........

6     node->items[c - 1] = EMPTY_ITEM;
7     ........
8     return 0;
9 }    // This function is executed in a transaction.

```

Object is updated without logging and is not persisted!
Persistency Model Violations: Missing Persist Barrier

```c
nvm_desc nvm_create_region (nvm_desc desc, const char* pathname,
const char *regionname, void *attach, size_t vspace, size_t pspace, mode_t mode) {
    ........
    nvm_flush (region, sizeof (*region));
    ...
    nvm_app_data *ad = nvm_get_app_data ();
    nvm_txbegin (desc);
    ........
    nvm_txend ();
    return desc;
}
```

*nvm_create_region from NVM-Direct using strict persistency*
Persistency Model Violations: Missing Persist Barrier

nvm_desc nvm_create_region (nvm_desc desc, const char* pathname,
const char *regionname, void *attach, size_t vspace, size_t pspace, mode_t mode) {

........

nvm_flush (region, sizeof (*region));
...

nvm_app_data *ad = nvm_get_app_data ();
nvm_txbegin (desc);
........
nvm_txend ();
return desc;
}

nvm_create_region from NVM-Direct using strict persistency

No persist barrier to enforce ordering
nvm_desc nvm_create_region (nvm_desc desc, const char* pathname,
const char *regionname, void *attach, size_t vspace, size_t pspace, mode_t mode) {

........

nvm_flush (region, sizeof (*region));
...

nvm_app_data *ad = nvm_get_app_data ();
nvm_txbegin (desc);
........
nvm_txend ();
return desc;
}

nvm_create_region from NVM-Direct using strict persistency

Object is flushed but ordering is not enforced with persist barrier

No persist barrier to enforce ordering
# Persistency Model Violations: Checking Rules

<table>
<thead>
<tr>
<th>Model</th>
<th>Persistency Model Violation</th>
<th>Checking Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Strict</strong></td>
<td>Unflushed/unlogged write</td>
<td>An operation $W$ writing to addr $A_1$, should be followed by a flush $F$ at addr $A_2$, where $A_1 = A_2$.</td>
</tr>
<tr>
<td></td>
<td>Multiple writes made durable at once</td>
<td>A persist barrier $P$ should be preceded by only one write $W$.</td>
</tr>
<tr>
<td><strong>Epoch</strong></td>
<td>Missing persist barriers between epochs</td>
<td>For any consecutive disjoint epochs $E_1$ and $E_2$, there should be a persist barrier $P$ at the end $E_1$.</td>
</tr>
<tr>
<td></td>
<td>Missing persist barriers in nested transactions</td>
<td>For any epoch $E_1$ inside of epoch $E_2$, there should be a persist barrier $P$ at the end $E_1$.</td>
</tr>
<tr>
<td></td>
<td>Unflushed/unlogged write</td>
<td>A $W$ writing to addr $A_1$, should be followed by a flush $F$ at addr $A_2$, where $A_1 \cap A_2 = A_1$.</td>
</tr>
<tr>
<td></td>
<td>Mismatch between program semantics and real implementation of persistent operations</td>
<td>For any consecutive epochs $E_1$ and $E_2$ writing to addresses $A_1$ and $A_2$ respectively, where $A_1 \in O_1$ and $A_2 \in O_2$, then $O_1 \neq O_2$.</td>
</tr>
<tr>
<td><strong>Strand</strong></td>
<td>Having data dependencies between strands</td>
<td>For any concurrent strands $S_1$ and $S_2$, operating on addrs $A_1$ and $A_2$ respectively, $A_1 \cap A_2 = \emptyset$.</td>
</tr>
</tbody>
</table>
# Persistency Model Violations: Checking Rules

<table>
<thead>
<tr>
<th>Model</th>
<th>Persistency Model Violation</th>
<th>Checking Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Strict</strong></td>
<td>Unflushed/unlogged write</td>
<td>An operation $W$ writing to addr $A_1$, should be followed by a flush $F$ at addr $A_2$, where $A_1 = A_2$.</td>
</tr>
<tr>
<td></td>
<td>Multiple writes made durable at once</td>
<td>A persist barrier $P$ should be preceded by only one write $W$.</td>
</tr>
<tr>
<td><strong>Epoch</strong></td>
<td>Missing persist barriers between epochs</td>
<td>For any consecutive disjoint epochs $E_1$ and $E_2$, there should be a persist barrier $P$ at the end $E_1$.</td>
</tr>
<tr>
<td></td>
<td>Missing persist barriers in nested transactions</td>
<td>For any epoch $E_1$ inside of epoch $E_2$, there should be a persist barrier $P$ at the end $E_1$.</td>
</tr>
<tr>
<td></td>
<td>Unflushed/unlogged write</td>
<td>A $W$ writing to addr $A_1$, should be followed by a flush $F$ at addr $A_2$, where $A_1 \cap A_2 = A_1$.</td>
</tr>
<tr>
<td></td>
<td>Mismatch between program semantics and real implementation of persistent operations</td>
<td>For any consecutive epochs $E_1$ and $E_2$ writing to addresses $A_1$ and $A_2$ respectively, where $A_1 \in O_1$ and $A_2 \in O_2$, then $O_1 \neq O_2$.</td>
</tr>
<tr>
<td><strong>Strand</strong></td>
<td>Having data dependencies between strands</td>
<td>For any concurrent strands $S_1$ and $S_2$, operating on addr $A_1$ and $A_2$ respectively, $A_1 \cap A_2 = \emptyset$.</td>
</tr>
</tbody>
</table>

**Strict Persistency**

- Every write is followed by a flush.
- Every flush is preceded by a single write.
Persistency Model Violations: Checking Rules

<table>
<thead>
<tr>
<th>Model</th>
<th>Persistency Model Violation</th>
<th>Checking Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strict</td>
<td>Unflushed/unlogged write</td>
<td>An operation $W$ writing to addr $A_1$, should be followed by a flush $F$ at addr $A_2$, where $A_1 = A_2$.</td>
</tr>
<tr>
<td></td>
<td>Multiple writes made durable at once</td>
<td>A persist barrier $P$ should be preceded by only one write $W$.</td>
</tr>
<tr>
<td>Epoch</td>
<td>Missing persist barriers between epochs</td>
<td>For any consecutive disjoint epochs $E_1$ and $E_2$, there should be a persist barrier $P$ at the end $E_1$.</td>
</tr>
<tr>
<td></td>
<td>Missing persist barriers in nested transactions</td>
<td>For any epoch $E_1$ inside of epoch $E_2$, there should be a persist barrier $P$ at the end $E_1$.</td>
</tr>
<tr>
<td></td>
<td>Unflushed/unlogged write</td>
<td>A $W$ writing to addr $A_1$, should be followed by a flush $F$ at addr $A_2$, where $A_1 \cap A_2 = A_1$.</td>
</tr>
<tr>
<td></td>
<td>Mismatch between program semantics and real implementation of persistent operations</td>
<td>For any consecutive epochs $E_1$ and $E_2$ writing to addresses $A_1$ and $A_2$ respectively, where $A_1 \in O_1$ and $A_2 \in O_2$, then $O_1 \neq O_2$.</td>
</tr>
<tr>
<td>Strand</td>
<td>Having data dependencies between strands</td>
<td>For any concurrent strands $S_1$ and $S_2$, operating on addrs $A_1$ and $A_2$ respectively, $A_1 \cap A_2 = \emptyset$.</td>
</tr>
</tbody>
</table>

**Strict Persistency**
- Every write is followed by a flush.
- Every flush is preceded by a single write.

**Epoch Persistency**
- Every epoch is followed by a flush.
- Consecutive or nested epochs have barriers between them.
## Persistency Model Violations: Checking Rules

<table>
<thead>
<tr>
<th>Model</th>
<th>Persistency Model Violation</th>
<th>Checking Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Strict</strong></td>
<td>Unflushed/unlogged write</td>
<td>An operation $W$ writing to addr $A_1$, should be followed by a flush $F$ at addr $A_2$, where $A_1 = A_2$.</td>
</tr>
<tr>
<td></td>
<td>Multiple writes made durable at once</td>
<td>A persist barrier $P$ should be preceded by only one write $W$.</td>
</tr>
<tr>
<td><strong>Epoch</strong></td>
<td>Missing persist barriers between epochs</td>
<td>For any consecutive disjoint epochs $E_1$ and $E_2$, there should be a persist barrier $P$ at the end $E_1$.</td>
</tr>
<tr>
<td></td>
<td>Missing persist barriers in nested transactions</td>
<td>For any epoch $E_1$ inside of epoch $E_2$, there should be a persist barrier $P$ at the end $E_1$.</td>
</tr>
<tr>
<td></td>
<td>Unflushed/unlogged write</td>
<td>A $W$ writing to addr $A_1$, should be followed by a flush $F$ at addr $A_2$, where $A_1 \cap A_2 = A_1$.</td>
</tr>
<tr>
<td></td>
<td>Mismatch between program semantics and real implementation of persistent operations</td>
<td>For any consecutive epochs $E_1$ and $E_2$ writing to addresses $A_1$ and $A_2$ respectively, where $A_1 \subseteq O_1$ and $A_2 \subseteq O_2$, then $O_1 \neq O_2$.</td>
</tr>
<tr>
<td><strong>Strand</strong></td>
<td>Having data dependencies between strands</td>
<td>For any concurrent strands $S_1$ and $S_2$, operating on addr $A_1$ and $A_2$ respectively, $A_1 \cap A_2 = \emptyset$.</td>
</tr>
</tbody>
</table>

### Strict Persistency
- Every write is followed by a flush.
- Every flush is preceded by a single write.

### Epoch Persistency
- Every epoch is followed by a flush.
- Consecutive or nested epochs have barriers between them.

### Strand Persistency
- Different strands should write to different addresses.
**Persistency Model Violations: Checking Rules**

<table>
<thead>
<tr>
<th>Model</th>
<th>Persistency Model Violation</th>
<th>Checking Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Strict</strong></td>
<td>Unflushed/unlogged write</td>
<td>An operation $W$ writing to addr $A_1$, should be followed by a flush $F$ at addr $A_2$, where $A_1 = A_2$.</td>
</tr>
<tr>
<td></td>
<td>Multiple writes made durable at once</td>
<td>A persist barrier $P$ should be preceded by only one write $W$.</td>
</tr>
<tr>
<td><strong>Epoch</strong></td>
<td>Missing persist barriers between epochs</td>
<td>For any consecutive disjoint epochs $E_1$ and $E_2$, there should be a persist barrier $P$ at the end $E_1$.</td>
</tr>
<tr>
<td></td>
<td>Missing persist barriers in nested transactions</td>
<td>For any epoch $E_1$ inside of epoch $E_2$, there should be a persist barrier $P$ at the end $E_1$.</td>
</tr>
<tr>
<td></td>
<td>Unflushed/unlogged write</td>
<td>A $W$ writing to addr $A_1$, should be followed by a flush $F$ at addr $A_2$, where $A_1 \cap A_2 = A_1$.</td>
</tr>
<tr>
<td></td>
<td>Mismatch between program semantics and real implementation of persistent operations</td>
<td>For any consecutive epochs $E_1$ and $E_2$ writing to addresses $A_1$ and $A_2$ respectively, where $A_1 \in O_1$ and $A_2 \in O_2$, then $O_1 \neq O_2$.</td>
</tr>
<tr>
<td><strong>Strand</strong></td>
<td>Having data dependencies between strands</td>
<td>For any concurrent strands $S_1$ and $S_2$, operating on addr $A_1$ and $A_2$ respectively, $A_1 \cap A_2 = \emptyset$.</td>
</tr>
</tbody>
</table>

**Strict Persistency**

- Every write is followed by a flush.
- Every flush is preceded by a single write.

**Epoch Persistency**

- Every epoch is followed by a flush.
- Consecutive or nested epochs have barriers between them.

**Strand Persistency**

- Different strands should write to different addresses.
Classifying Persistency Bugs in NVM Programs

Persistency Model Violations
- Semantic Mismatch
- Unflushed/Unlogged Writes
- Missing Persist Barrier

Performance Bugs
- Flushing Unmodified Data
- Redundant Write-backs of Data
- Durable Transactions without Updates
Classifying Persistency Bugs in NVM Programs

Persistency Model Violations
- Semantic Mismatch
- Unflushed/Unlogged Writes
- Missing Persist Barrier

Performance Bugs
- Flushing Unmodified Data
- Redundant Write-backs of Data
- Durable Transactions without Updates
static int pi_task_construct (PMEMobjpool *pop, void *ptr, void *arg) {
    struct pi_task *t = (struct pi_task *) ptr;
    struct pi_task_proto *p = (struct pi_task_proto *) arg;
    t->proto = *p;
    pmemobj_persist (pop, t, sizeof(*t));
    return 0;
}

pi_task_construct from PMDK
Performance Bugs: Flushing Unmodified Data

```
1  static int pi_task_construct (PMEMobjpool *pop, void *ptr, void *arg) {
2       struct pi_task *t = (struct pi_task *) ptr;
3       struct pi_task_proto *p = (struct pi_task_proto *) arg;
4       t->proto = *p;
5       pmemobj_persist (pop, t, sizeof(*t));
6       return 0;
7  }
```

Persist entire object when only one field is modified.

*pi_task_construct* from PMDK
Performance Bugs: Flushing Unmodified Data

Persist entire object when only one field is modified.

```c
1 static int pi_task_construct (PMEMobjpool *pop, void *ptr, void *arg) {
2     struct pi_task *t = (struct pi_task *) ptr;
3     struct pi_task_proto *p = (struct pi_task_proto *) arg;
4     t->proto = *p;
5     pmemobj_persist (pop, t, sizeof(*t));
6     return 0;
7 }
```

Flush unmodified data hurts performance!
Performance Bugs: Redundant Write-Backs of Data

```c
1 void nvm_free_callback (nvm_free_ctx *ctx) {
2       .......
3       nvm_free_blk (heap, nvb);
4       nvm_flushl (nvb);
5 }

6 void nvm_free_blk (nvm_heap *heap, nvm_blk *nvb) {
7       .......
8       nvm_flushl (nvb);
9 }
```

nvm_free from NVM-Direct
Performance Bugs: Redundant Write-Backs of Data

```c
1    void nvm_free_callback (nvm_free_ctx *ctx) {
2        .......
3    nvm_free_blk (heap, nvb);
4    nvm_flushl (nvb);
5 }
6    void nvm_free_blk (nvm_heap *heap, nvm_blk *nvb) {
7        .......
8    nvm_flushl (nvb);
9 }
```

nvm_free from NVM-Direct
Performance Bugs: Redundant Write-Backs of Data

1: void nvm_free_callback (nvm_free_ctx *ctx) {
2:     .......
3:     nvm_free_blk (heap, nvb);
4:     nvm_flushl (nvb);
5: }

6: void nvm_free_blk (nvm_heap *heap, nvm_blk *nvb) {
7:     .......
8:     nvm_flushl (nvb);
9: }

Flushing twice in a row.

nvm_free from NVM-Direct
Performance Bugs: Redundant Write-Backs of Data

void nvm_free_callback (nvm_free_ctx *ctx) {
    .......
    nvm_free_blk (heap, nvb);
    nvmflushl (nvb);
}

void nvm_free_blk (nvm_heap *heap, nvm_blk *nvb) {
    .......
    nvmflushl (nvb);
}

flushing twice in a row.

Redundant flushing does not affect correctness and hurts performance!
static int timer_tick (uint32_t *timer) {
    int ret = *timer == 0 || (*((timer)--) == 0;
    pmemobj_persist (pop, timer, sizeof (*timer));
    return ret;
}

static void process_set (void) {
    ..........  // Example code
    if (timer_tick (&iter->timer)) {
        iter->timer = MAX_ALIEN_TIMER;
        iter->y++;
    }
    pmemobj_persist (pop, iter, sizeof (struct alien));
    ..........  // Example code
}
static int timer_tick (uint32_t *timer) {
    int ret = *timer == 0 || (*((timer)--) == 0;
    pmemobj_persist (pop, timer, sizeof (*timer));
    return ret;
}

static void process_aliens (void) {
    ........
    if (timer_tick (&iter->timer)) {
        iter->timer = MAX_ALIEN_TIMER;
        iter->y++; 
    }
    pmemobj_persist (pop, iter, sizeof (struct alien));
    ........
}
Performance Bugs: Transactions without Updates

```c
1  static int timer_tick (uint32_t *timer) {
2      int ret = *timer == 0 || (*((timer)--)) == 0;
3      pmemobj_persist (pop, timer, sizeof (*timer));
4      return ret;
5  }
6  static void process_aliens (void) {
7      ........
8      if (timer_tick (&iter->timer)) {
9          iter->timer = MAX_ALIEN_TIMER;
10         iter->y++;
11      }
12      pmemobj_persist (pop, iter, sizeof (struct alien));
13      ........
14  }
```

Object is unmodified if condition is false!
Persist alien object

pm_invaders from PMDK examples
Performance Bugs: Transactions without Updates

Transactions without updates enforce unnecessary orderings!

```
1  static int timer_tick (uint32_t *timer) {
2      int ret = *timer == 0 ll ((*timer)--) == 0;
3  pmemobj_persist (pop, timer, sizeof (*timer));
4      return ret;
5  }
6  static void process_aliens (void) {
7       ........
8      if (timer_tick (&iter->timer)) {
9          iter->timer = MAX_Alien_TIMER;
10         iter->y++;
11      }
12     pmemobj_persist (pop, iter, sizeof (struct alien));
13   ........
14   }
```

Object is unmodified if condition is false!
Persist alien object

pm_invaders from PMDK examples
Performance Bugs: Checking Rules

- Flushing Unmodified Data
- Redundant Write-Backs of Updated Data
- Durable Transactions Without Updates
Performance Bugs: Checking Rules

- Flushing Unmodified Data
- Every flush should have a preceding write
- Redundant Write-Backs of Updated Data
- Durable Transactions Without Updates
Performance Bugs: Checking Rules

- Flushing Unmodified Data
  - Every flush should have a preceding write

- Redundant Write-Backs of Updated Data
  - Consecutive flushes should not flush the same address

- Durable Transactions Without Updates
Performance Bugs: Checking Rules

- **Flushing Unmodified Data**: Every flush should have a preceding write.
- **Redundant Write-Backs of Updated Data**: Consecutive flushes should not flush the same address.
- **Durable Transactions Without Updates**: Every transaction should contain at least one write.
Classifying Persistency Bugs in NVM Programs

Persistency Model Violations
- Semantic Mismatch
- Unflushed/Unlogged Writes
- Missing Persist Barrier

Performance Bugs
- Flushing Unmodified Data
- Redundant Write-backs of Data
- Durable Transactions without Updates
Classifying Persistency Bugs in NVM Programs

Persistency Model Violations
- Semantic Mismatch
- Unflushed/Unlogged Writes
- Missing Persist Barrier

Performance Bugs
- Flushing Unmodified Data
- Redundant Write-backs of Data
- Durable Transactions without Updates
Persistency Models for Non-Volatile Memory

Strict Persistency

Epoch Persistency

Strand Persistency
Persistency Models for Non-Volatile Memory

Static Analysis

Epoch Persistency

Strand Persistency

Strict Persistency rules can be checked statically!
Persistency Models for Non-Volatile Memory

Static Analysis

Dynamic Analysis

Detecting data races between strands or epochs requires dynamic analysis!
Persistency Models for Non-Volatile Memory

The static and dynamic components combine to check **all rules**
Detecting Persistency Bugs in NVM Programs

- Semantic Mismatch
- Unflushed/Unlogged Writes
- Missing Persist Barrier
- Flushing Unmodified Data
- Redundant Write-backs of Data
- Durable Transactions without Updates
Detecting Persistency Bugs in NVM Programs

- Semantic Mismatch
- Unflushed/Unlogged Writes
- Missing Persist Barrier
- Flushing Unmodified Data
- Redundant Write-backs of Data
- Durable Transactions without Updates
Detecting Persistency Bugs in NVM Programs

- Semantic Mismatch
- Unflushed/Unlogged Writes
- Missing Persist Barrier
- Flushing Unmodified Data
- Redundant Write-backs of Data
- Durable Transactions without Updates

Can be detected statically!
Detecting Persistency Bugs in NVM Programs

Epoch and Strand dependencies require runtime information!

- Semantic Mismatch
- Unflushed/Unlogged Writes
- Missing Persist Barrier
- Flushing Unmodified Data
- Redundant Write-backs of Data
- Durable Transactions without Updates

Can be detected statically!
Detecting Persistency Bugs in NVM Programs

• Unflushed/Unlogged Writes
• Missing Persist Barrier
• Flushing Unmodified Data
• Redundant Write-backs of Data
• Durable Transactions without Updates

We introduce a static and dynamic component to check **all** rules!

Epoch and Strand dependencies require runtime information!

Can be detected statically!
Detecting Memory Persistency Bugs with DeepMC

NVM Program → LLVM → IR

Compile the program into LLVM IR
Detecting Memory Persistency Bugs with DeepMC

Apply data structure analysis!
Adapting Data Structure Analysis to Persistent Objects

Phase 1: Local Analysis

Create nodes for functions and new variables with edges for dependencies
Adapting Data Structure Analysis to Persistent Objects

- **Phase 1:** Local Analysis
- **Phase 2:** Bottom-Up Analysis

Resolve updates occurring in function calls with callee information
Adapting Data Structure Analysis to Persistent Objects

Phase 1: Local Analysis
Phase 2: Bottom-Up Analysis
Phase 3: Top-Down Analysis

Include caller information to finalize the data structure graph
Adapting Data Structure Analysis to Persistent Objects

```c
int nvm_lock (nvmutex *mutex, int excl, int timeout) {
    nvmutex *mutex = (nvmutex*)omutex;
    ...
    nvm_lock_rec *lk = nvm_add_lock_op(tx,td,mutex,sl);
    ...
    lk->state = nvm_lock_acquire_s;
    nvm_persist1(&lk->state);
    mutex->owners--;
    nvm_persist1(&mutex->owners);
    if (mutex->level > lk->new_level)
        lk->new_level = mutex->level;
    lk->state = nvm_lock_held_s;
    nvm_persist1(&lk->state);
}
```

nvm_lock from NVM-Direct

Phase 1: Local Analysis

Data structure graph for nvm_lock
Adapting Data Structure Analysis to Persistent Objects

1. int nvm_lock (nvm_mutex *omutex, int excl, int timeout) {
2.   nvm_amutex *mutex = (nvm_amutex*)omutex;
   ...
3.   nvm_lkrec *lk = nvm_add_lock_op(tx,td,mutex,st);
   ...
4.   lk->state = nvm_lock_acquire_s;
5.   nvm_persist1(&lk->state);
6.   mutex->owners--;
7.   nvm_persist1(&mutex->owners);
8.   if (mutex->level > lk->new_level)
9.      lk->new_level = mutex->level;
10.  lk->state = nvm_lock_held_s;
11.  nvm_persist1(&lk->state);
12. }

Data structure graph for nvm_lock

Phase 1: Local Analysis

nvm_lock from NVM-Direct
Adapting Data Structure Analysis to Persistent Objects

```c
1  int nvm_lock (nvm_mutex *omutex, int excl, int timeout) {
2    nvm_amutex *mutex = (nvm_amutex*)omutex;
3       ...
4    lk->state = nvm_lock_acquire_s;
5    nvm_persist1(&lk->state);
6    mutex->owners--;  
7    nvm_persist1(&mutex->owners);
8    if (mutex->level > lk->new_level)
9       lk->new_level = mutex->level;
10   lk->state = nvm_lock_held_s;
11   nvm_persist1(&lk->state);
12  }
```
Adapting Data Structure Analysis to Persistent Objects

```c
int nvm_lock (nvm_mutex *omutex, int excl, int timeout) {
    nvm_amutex *mutex = (nvm_amutex*)omutex;
    ...
    nvm_lkrec *lk = nvm_add_lock_op(tx,td,mutex,st);
    ...
    lk->state = nvm_lock_acquire_s;
    nvmPersist1(&lk->state);
    mutex->owners--;
    nvmPersist1(&mutex->owners);
    if (mutex->level > lk->new_level)
        lk->new_level = mutex->level;
    lk->state = nvm_lock_held_s;
    nvmPersist1(&lk->state);
}
```

**Phase 1: Local Analysis**

**nvm_lock from NVM-Direct**

**Data structure graph for nvm_lock**
Adapting Data Structure Analysis to Persistent Objects

```c
int nvm_lock (nvm_mutex *omutex, int excl, int timeout) {
    nvm_amutex *mutex = (nvm_amutex*)omutex;
    ...
    nvm_lkrec *lk = nvm_add_lock_op(tx,td,mutex,st);
    ...
    lk->state = nvm_lock_acquire_s;
    nvm_persist1(&lk->state);
    mutex->owners--;  
    nvm_persist1(&mutex->owners);
    if (mutex->level > lk->new_level)
        lk->new_level = mutex->level;
    lk->state = nvm_lock_held_s;
    nvm_persist1(&lk->state);
}
```

**nvm_lock from NVM-Direct**

**Data structure graph for nvm_lock**

**Phase 1: Local Analysis**
Adapting Data Structure Analysis to Persistent Objects

```
int nvm_lock (nvm_mutex *omutex, int excl, int timeout) {
    nvm_amutex *mutex = (nvm_amutex*)omutex;
    ...
    nvm_lkrec *lk = nvm_add_lock_op(tx,td,mutex,st);
    ...
    lk->state = nvm_lock_acquire_s;
    nvm_persist1(&lk->state);
    mutex->owners--;
    nvm_persist1(&mutex->owners);
    if (mutex->level > lk->new_level)
        lk->new_level = mutex->level;
    lk->state = nvm_lock_held_s;
    nvm_persist1(&lk->state);
}
```

Data structure graph for nvm_lock

nvm_lock from NVM-Direct

Phase 1: Local Analysis
Adapting Data Structure Analysis to Persistent Objects

```c
int nvm_lock (nvm_mutex *omutex, int excl, int timeout) {
    nvm_amutex *mutex = (nvm_amutex*)omutex;
    ...
    lk = nvm_add_lock_op(tx,td,mutex,st);
    ...
    lk->state = nvm_lock_acquire_s;
    nvmPersist1(&lk->state);
    mutex->owners--;
    nvmPersist1(&mutex->owners);
    if (mutex->level > lk->new_level)
        lk->new_level = mutex->level;
    lk->state = nvm_lock_held_s;
    nvmPersist1(&lk->state);
}
```

Phase 1: Local Analysis

Data structure graph for nvm_lock
Adapting Data Structure Analysis to Persistent Objects

```c
int nvm_lock (nvm_mutex *omutex, int excl, int timeout) {
    nvm_amutex *mutex = (nvm_amutex*)omutex;
    ...
    nvm_lkrec *lk = nvm_add_lock_op(tx,td,mutex,st);
    ...
    lk->state = nvm_lock_acquire_s;
    nvm_persist1(&lk->state);
    mutex->owners--;  
    nvm_persist1(&mutex->owners);
    if (mutex->level > lk->new_level)  
        lk->new_level = mutex->level;
    lk->state = nvm_lock_held_s;
    nvm_persist1(&lk->state);
}
```

**nvm_lock from NVM-Direct**

**Phase 2: Bottom-Up Analysis**

**Data structure graph for nvm_lock**
Adapting Data Structure Analysis to Persistent Objects

```c
1  int nvm_lock (nvm_mutex *omutex, int excl, int timeout) {
2      nvm_amutex *mutex = (nvm_amutex*)omutex;
3      ...
3  nvm_lkrec *lk = nvm_add_lock_op(tx,td,mutex,st);
4      ...
4    lk->state = nvm_lock_acquire_s;
5    nvm_persist1(&lk->state);
6    mutex->owners--;  
7    nvm_persist1(&mutex->owners);
8    if (mutex->level > lk->new_level)  
9        lk->new_level = mutex->level;
10   lk->state = nvm_lock_held_s;
11   nvm_persist1(&lk->state);
12  }
```

**nvm_lock from NVM-Direct**

**Phase 2: Bottom-Up Analysis**

Data structure graph for nvm_lock:

```
int: nvm_lock
  omutex excl timeout

mutex
  excl
  timeout

lk

void: nvm_persist1
  pers_obj
```

Systems Platform Research Group at UIUC
Adapting Data Structure Analysis to Persistent Objects

```c
int nvm_lock (nvm_mutex *omutex, int excl, int timeout) {
    nvm_amutex *mutex = (nvm_amutex*)omutex;
    ...
    nvmlkrec *lk = nvm_add_lock_op(tx,td,mutex,st);
    ...
    lk->state = nvm_lock_acquire_s;
    nvm_persist1(&lk->state);
    mutex->owners--;
    nvm_persist1(&mutex->owners);
    if (mutex->level > lk->new_level)
        lk->new_level = mutex->level;
    lk->state = nvm_lock_held_s;
    nvm_persist1(&lk->state);
}
```

Phase 2: Bottom-Up Analysis

*nvm_lock* from NVM-Direct

Data structure graph for *nvm_lock*
Adapting Data Structure Analysis to Persistent Objects

```c
1  int nvm_lock (nvm_mutex *omutex, int excl, int timeout) {
2     nvm_amutex *mutex = (nvm_amutex*)omutex;
3     ...
4     lk->state = nvm_lock_acquire_s;
5     nvm.persist1(&lk->state);
6     mutex->owners--;  
7     nvm.persist1(&mutex->owners);
8     if (mutex->level > lk->new_level)
9         lk->new_level = mutex->level;
10     lk->state = nvm_lock_held_s;
11     nvm.persist1(&lk->state);
12 }
```

nvm_lock from NVM-Direct

Phase 2: Bottom-Up Analysis
Adapting Data Structure Analysis to Persistent Objects

1 int nvm_lock (nvm_mutex *omutex, int excl, int timeout) {
2     nvm_amutex *mutex = (nvm_amutex*)omutex;
3         ...
4     nvm_lkrec *lk = nvm_add_lock_op(tx,td,mutex,st);
5         ...
6     lk->state = nvm_lock_acquire_s;
7     nvm_persist1(&lk->state);
8     mutex->owners--;    
9     nvm_persist1(&mutex->owners);
10    if (mutex->level > lk->new_level)
11       lk->new_level = mutex->level;
12    lk->state = nvm_lock_held_s;
13    nvm_persist1(&lk->state);
14 }

nvm_lock from NVM-Direct

Data structure graph for nvm_lock

Phase 3: Top-Down Analysis

omutex  excl  timeout
mutext  excl  timeout
lk
Adapting Data Structure Analysis to Persistent Objects

Phase 3: Top-Down Analysis

```c
int nvm_lock (nvm_mutex *omutex, int excl, int timeout) {
    nvm_amutex *mutex = (nvm_amutex*)omutex;
    ...
    nvm_lkrec *lk = nvm_add_lock_op(tx,td.mutex,st);
    ...
    lk->state = nvm_lock_acquire_s;
    nvm_persist1(&lk->state);
    mutex->owners--;  /* lk->state */  
    nvm_persist1(&mutex->owners);
    if (mutex->level > lk->new_level)
      lk->new_level = mutex->level;
    lk->state = nvm_lock_held_s;
    nvm_persist1(&lk->state);
}
```

Data structure graph for nvm_lock

nvm_lock from NVM-Direct
Detecting Memory Persistency Bugs with DeepMC
Detecting Memory Persistency Bugs with DeepMC

Combine with checking rules for static checking
Applying the Data Structure Graph to NVM Programs

Traverse control flow graph in depth-first order

Local Trace in function A
... write a
call function B()
... persist barrier

Local Trace in function B
write b
... persist barrier
Applying the Data Structure Graph to NVM Programs

Traverse control flow graph in depth-first order

Local Trace in function A
... write a
call function B()
... persist barrier

Local Trace in function B
write b
... persist barrier

Merge Point
Applying the Data Structure Graph to NVM Programs

Merge function calls into their call sites

Local Trace in function A
... write a
    call function B()
... persist barrier

Local Trace in function B
write b
... persist barrier

Merged Traces in function A
... write a
write b
... persist barrier
... persist barrier

Merge Point
Applying the Data Structure Graph to NVM Programs

Local Trace in function A

... write a
call function B()...
persist barrier

Merge Point

Local Trace in function B

write b...
persist barrier

Split Point

Merge

Merged Traces in function A

... write a
... write b
... persist barrier
... persist barrier

Merge function calls into their call sites
Applying the Data Structure Graph to NVM Programs

Local Trace in function A

... write a
call function B()
... persist barrier

Merge Point

Local Trace in function B

write b
... persist barrier

Split Point

Merged Traces in function A

... write a
write b
... persist barrier
... persist barrier

Split Point

Traces in function A

... write a
write b
... persist barrier
... persist barrier

Split into smaller traces at persistent barriers
Applying our Checking Rules to Static Analysis

```
1    int nvm_lock (nvm_mutex *omutex, int excl, int timeout) {
2        nvm_amutex *mutex = (nvm_amutex*)omutex;
3            ...
4        nvm_lkrec *lk = nvm_add_lock_op(tx,td,mutex,st);
5            ...
6        lk->state = nvm_lock_acquire_s;
7        nvm_persist1(&lk->state);
8        mutex->owners--;  
9        nvm_persist1(&mutex->owners);
10       if (mutex->level > lk->new_level)
11          lk->new_level = mutex->level;
12       lk->state = nvm_lock_held_s;
13       nvm_persist1(&lk->state);
14    }
```

nvm_lock from NVM-Direct

Trace
Applying our Checking Rules to Static Analysis

```c
int nvm_lock (nvm_mutex *omutex, int excl, int timeout) {
    nvm_amutex *mutex = (nvm_amutex*)omutex;
    ...
    nvm_lkrec *lk = nvm_add_lock_op(tx,td,mutex,st);
    ...
    lk->state = nvm_lock_acquire_s;
    nvm_persist1(&lk->state);
    mutex->owners--;  
    nvm_persist1(&mutex->owners);
    if (mutex->level > lk->new_level)
        lk->new_level = mutex->level;
    lk->state = nvm_lock_held_s;
    nvm_persist1(&lk->state);
}
```

**nvm_lock from NVM-Direct**
Applying our Checking Rules to Static Analysis

```c
1 int nvm_lock (nvm_mutex *omutex, int excl, int timeout) {
2    nvm_amutex *mutex = (nvm_amutex*)omutex;
3    ...
4    lk->state = nvm_lock_acquire_s;
5    nvm_persist1(&lk->state);
6    mutex->owners--;
7    nvm_persist1(&mutex->owners);
8    if (mutex->level > lk->new_level)
9       lk->new_level = mutex->level;
10   lk->state = nvm_lock_held_s;
11   nvm_persist1(&lk->state);
12 }
```

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>4</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>5</td>
<td>state</td>
</tr>
</tbody>
</table>

nvm_lock from NVM-Direct

Trace
int nvm_lock (nvm_mutex *omutex, int excl, int timeout) {
    nvm_amutex *mutex = (nvm_amutex*)omutex;
    ...
    nvm_lkrec *lk = nvm_add_lock_op(tx,td,mutex,st);
    ...
    lk->state = nvm_lock_acquire_s;
    nvm_persist1(&lk->state);
    mutex->owners--;
    nvm_persist1(&mutex->owners);
    if (mutex->level > lk->new_level)
        lk->new_level = mutex->level;
    lk->state = nvm_lock_held_s;
    nvm_persist1(&lk->state);
}
int nvm_lock (nvm_mutex *omutex, int excl, int timeout) {
     nvm_amutex *mutex = (nvm_amutex*)omutex;

     ...

     nvm_lkrec *lk = nvm_add_lock_op(tx,td,mutex,st);
     ...

     lk->state = nvm_lock_acquire_s;

     nvmPersist1(&lk->state);

     mutex->owners--;

     nvmPersist1(&mutex->owners);

    if (mutex->level > lk->new_level)
       lk->new_level = mutex->level;

    lk->state = nvm_lock_held_s;

    nvmPersist1(&lk->state);

} nvm_lock from NVM-Direct
Applying our Checking Rules to Static Analysis

```
int nvm_lock (nvm_mutex *omutex, int excl, int timeout) {
    nvm_amutex *mutex = (nvm_amutex*)omutex;
    ...
    nvm_lkrec *lk = nvm_add_lock_op(tx,td,mutex,st);
    ...
    lk->state = nvm_lock_acquire_s;
    nvm_persist1(&lk->state);
    mutex->owners--;
    nvm_persist1(&mutex->owners);
    if (mutex->level > lk->new_level)
        lk->new_level = mutex->level;
    lk->state = nvm_lock_held_s;
    nvm_persist1(&lk->state);
}
```

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>4</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>Write</td>
<td>6</td>
<td>owners</td>
</tr>
<tr>
<td>Flush</td>
<td>7</td>
<td>owners</td>
</tr>
<tr>
<td>Fence</td>
<td>7</td>
<td>owners</td>
</tr>
<tr>
<td>Write</td>
<td>9</td>
<td>new_level</td>
</tr>
</tbody>
</table>

nvm_lock from NVM-Direct
Applying our Checking Rules to Static Analysis

```c
int nvm_lock (nvm_mutex *omutex, int excl, int timeout) {
    nvm_amutex *mutex = (nvm_amutex*)omutex;
    ...
    nvm_lkrec *lk = nvm_add_lock_op(tx,td,mutex,st);
    ...
    lk->state = nvm_lock_acquire_s;
    nvm_persist1(&lk->state);
    mutex->owners--;
    nvm_persist1(&mutex->owners);
    if (mutex->level > lk->new_level)
        lk->new_level = mutex->level;
    lk->state = nvm_lock_held_s;
    nvm_persist1(&lk->state);
}
```

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>4</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>Write</td>
<td>6</td>
<td>owners</td>
</tr>
<tr>
<td>Flush</td>
<td>7</td>
<td>owners</td>
</tr>
<tr>
<td>Fence</td>
<td>7</td>
<td>owners</td>
</tr>
<tr>
<td>Write</td>
<td>9</td>
<td>new_level</td>
</tr>
<tr>
<td>Write</td>
<td>10</td>
<td>state</td>
</tr>
</tbody>
</table>

**nvm_lock from NVM-Direct**
Applying our Checking Rules to Static Analysis

```c
int nvm_lock (nvm_mutex *omutex, int excl, int timeout) {
    nvm_amutex *mutex = (nvm_amutex*)omutex;
    ...
    nvm_lkrec *lk = nvm_add_lock_op(tx,td,mutex,st);
    ...
    lk->state = nvm_lock_acquire_s;
    nvm_persist1(&lk->state);
    mutex->owners--;
    nvm_persist1(&mutex->owners);
    if (mutex->level > lk->new_level)
        lk->new_level = mutex->level;
    lk->state = nvm_lock_held_s;
    nvm_persist1(&lk->state);
}
```

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>4</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>Write</td>
<td>6</td>
<td>owners</td>
</tr>
<tr>
<td>Flush</td>
<td>7</td>
<td>owners</td>
</tr>
<tr>
<td>Fence</td>
<td>7</td>
<td>owners</td>
</tr>
<tr>
<td>Write</td>
<td>9</td>
<td>new_level</td>
</tr>
<tr>
<td>Write</td>
<td>10</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>11</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>11</td>
<td>state</td>
</tr>
</tbody>
</table>

**nvm_lock from NVM-Direct**
Applying our Checking Rules to Static Analysis

```
1 int nvm_lock (nvm_mutex *omutex, int excl, int timeout) {
2    nvm_amutex *mutex = (nvm_amutex*)omutex;
3          ...
4    nvm_lkrec *lk = nvm_add_lock_op(tx,td,mutex,st);
5          ...
6    lk->state = nvm_lock_acquire_s;
7    nvm_persist1(&lk->state);
8    mutex->owners--;  
9    nvm_persist1(&mutex->owners);
10   if (mutex->level > lk->new_level)  
11      lk->new_level = mutex->level;  
12      lk->state = nvm_lock_held_s;
13     nvm_persist1(&lk->state);
14 }
```

**nvm_lock from NVM-Direct**

---

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>4</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>Write</td>
<td>6</td>
<td>owners</td>
</tr>
<tr>
<td>Flush</td>
<td>7</td>
<td>owners</td>
</tr>
<tr>
<td>Fence</td>
<td>7</td>
<td>owners</td>
</tr>
<tr>
<td>Write</td>
<td>9</td>
<td>new_level</td>
</tr>
<tr>
<td>Write</td>
<td>10</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>11</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>11</td>
<td>state</td>
</tr>
</tbody>
</table>

**Trace**
Applying our Checking Rules to Static Analysis

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>4</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>Write</td>
<td>6</td>
<td>owners</td>
</tr>
<tr>
<td>Flush</td>
<td>7</td>
<td>owners</td>
</tr>
<tr>
<td>Fence</td>
<td>7</td>
<td>owners</td>
</tr>
<tr>
<td>Write</td>
<td>9</td>
<td>new_level</td>
</tr>
<tr>
<td>Write</td>
<td>10</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>11</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>11</td>
<td>state</td>
</tr>
</tbody>
</table>
### Applying our Checking Rules to Static Analysis

#### Model Violations

- Every write is followed by a flush
- Every flush is preceded by a single write

#### Strict Persistency Checking Rules

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>4</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>Write</td>
<td>6</td>
<td>owners</td>
</tr>
<tr>
<td>Flush</td>
<td>7</td>
<td>owners</td>
</tr>
<tr>
<td>Fence</td>
<td>7</td>
<td>owners</td>
</tr>
<tr>
<td>Write</td>
<td>9</td>
<td>new_level</td>
</tr>
<tr>
<td>Write</td>
<td>10</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>11</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>11</td>
<td>state</td>
</tr>
</tbody>
</table>

#### Traces

- Every write is followed by a flush
- Every flush is preceded by a single write
Applying our Checking Rules to Static Analysis

Performance Bugs Model Violations

- Every write is followed by a flush
- Every flush is preceded by a single write
- Every flush should have a preceding write
- Flushes should flush different addresses
- Transactions must have at least one write

Strict Persistency Checking Rules

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>4</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>5</td>
<td>state</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>6</td>
<td>owners</td>
</tr>
<tr>
<td>Flush</td>
<td>7</td>
<td>owners</td>
</tr>
<tr>
<td>Fence</td>
<td>7</td>
<td>owners</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>9</td>
<td>new_level</td>
</tr>
<tr>
<td>Write</td>
<td>10</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>11</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>11</td>
<td>state</td>
</tr>
</tbody>
</table>

Traces
Applying our Checking Rules to Static Analysis

**Strict Persistency Checking Rules**

- Every write is followed by a flush
- Every flush is preceded by a single write
- Every flush should have a preceding write
- Flushes should flush different addresses
- Transactions must have at least one write

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>4</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write</td>
<td>6</td>
<td>owners</td>
</tr>
<tr>
<td>Flush</td>
<td>7</td>
<td>owners</td>
</tr>
<tr>
<td>Fence</td>
<td>7</td>
<td>owners</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write</td>
<td>9</td>
<td>new_level</td>
</tr>
<tr>
<td>Write</td>
<td>10</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>11</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>11</td>
<td>state</td>
</tr>
</tbody>
</table>
Applying our Checking Rules to Static Analysis

**Strict Persistency Checking Rules**

- Every write is followed by a flush
- Every flush is preceded by a single write
- Every flush should have a preceding write
- Flushes should flush different addresses
- Transactions must have at least one write

**Traces**

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>4</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>5</td>
<td>state</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>6</td>
<td>owners</td>
</tr>
<tr>
<td>Flush</td>
<td>7</td>
<td>owners</td>
</tr>
<tr>
<td>Fence</td>
<td>7</td>
<td>owners</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>9</td>
<td>new_level</td>
</tr>
<tr>
<td>Write</td>
<td>10</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>11</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>11</td>
<td>state</td>
</tr>
</tbody>
</table>
Applying our Checking Rules to Static Analysis

### Performance Bugs
- Every write is followed by a flush
- Every flush is preceded by a single write
- Every flush should have a preceding write
- Flushes should flush different addresses
- Transactions must have at least one write

### Model Violations

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>4</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>5</td>
<td>state</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>6</td>
<td>owners</td>
</tr>
<tr>
<td>Flush</td>
<td>7</td>
<td>owners</td>
</tr>
<tr>
<td>Fence</td>
<td>7</td>
<td>owners</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>9</td>
<td>new_level</td>
</tr>
<tr>
<td>Write</td>
<td>10</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>11</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>11</td>
<td>state</td>
</tr>
</tbody>
</table>

### Traces
"Applying our Checking Rules to Static Analysis"

### Strict Persistency Checking Rules

<table>
<thead>
<tr>
<th>Performance Bugs</th>
<th>Model Violations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Every write is followed by a flush</td>
<td>✓</td>
</tr>
<tr>
<td>Every flush is preceded by a single write</td>
<td>✓</td>
</tr>
<tr>
<td>Every flush should have a preceding write</td>
<td>✓</td>
</tr>
<tr>
<td>Flushes should flush different addresses</td>
<td></td>
</tr>
<tr>
<td>Transactions must have at least one write</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>4</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>5</td>
<td>state</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>6</td>
<td>owners</td>
</tr>
<tr>
<td>Flush</td>
<td>7</td>
<td>owners</td>
</tr>
<tr>
<td>Fence</td>
<td>7</td>
<td>owners</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>9</td>
<td>new_level</td>
</tr>
<tr>
<td>Write</td>
<td>10</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>11</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>11</td>
<td>state</td>
</tr>
</tbody>
</table>

### Traces
Applying our Checking Rules to Static Analysis

**Performance Bugs Model Violations**

- Every write is followed by a flush
  - Op: Write, Line: 4, Obj: state
  - Op: Flush, Line: 5, Obj: state
  - Op: Fence, Line: 5, Obj: state

- Every flush is preceded by a single write
  - Op: Fence, Line: 7, Obj: owners

- Every flush should have a preceding write
  - Op: Fence, Line: 7, Obj: owners

- Flushes should flush different addresses
  - Op: Write, Line: 6, Obj: owners

- Transactions must have at least one write
  - Op: Write, Line: 9, Obj: new_level
  - Op: Write, Line: 10, Obj: state
  - Op: Flush, Line: 11, Obj: state
  - Op: Fence, Line: 11, Obj: state

---

**Strict Persistency Checking Rules**
Applying our Checking Rules to Static Analysis

**Strict Persistency Checking Rules**

- Every write is followed by a flush
- Every flush is preceded by a single write
- Every flush should have a preceding write
- Flushes should flush different addresses
- Transactions must have at least one write

**Traces**

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>4</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>Write</td>
<td>6</td>
<td>owners</td>
</tr>
<tr>
<td>Flush</td>
<td>7</td>
<td>owners</td>
</tr>
<tr>
<td>Fence</td>
<td>7</td>
<td>owners</td>
</tr>
<tr>
<td>Write</td>
<td>9</td>
<td>new_level</td>
</tr>
<tr>
<td>Write</td>
<td>10</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>11</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>11</td>
<td>state</td>
</tr>
</tbody>
</table>
Applying our Checking Rules to Static Analysis

**Strict Persistency Checking Rules**

**Model Violations**

- Every write is followed by a flush
- Every flush is preceded by a single write
- Every flush should have a preceding write
- Flushes should flush different addresses
- Transactions must have at least one write

---

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>4</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>Write</td>
<td>6</td>
<td>owners</td>
</tr>
<tr>
<td>Flush</td>
<td>7</td>
<td>owners</td>
</tr>
<tr>
<td>Fence</td>
<td>7</td>
<td>owners</td>
</tr>
<tr>
<td>Write</td>
<td>9</td>
<td>new_level</td>
</tr>
<tr>
<td>Write</td>
<td>10</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>11</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>11</td>
<td>state</td>
</tr>
</tbody>
</table>

**Traces**
Applying our Checking Rules to Static Analysis

**Performance Bugs Model Violations**

- Every write is followed by a flush
- Every flush is preceded by a single write
- Every flush should have a preceding write
- Flushes should flush different addresses
- Transactions must have at least one write

**Strict Persistency Checking Rules**

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>4</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>Write</td>
<td>6</td>
<td>owners</td>
</tr>
<tr>
<td>Flush</td>
<td>7</td>
<td>owners</td>
</tr>
<tr>
<td>Fence</td>
<td>7</td>
<td>owners</td>
</tr>
<tr>
<td>Write</td>
<td>9</td>
<td>new_level</td>
</tr>
<tr>
<td>Write</td>
<td>10</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>11</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>11</td>
<td>state</td>
</tr>
</tbody>
</table>

**Traces**

Systems Platform Research Group at UIUC
Applying our Checking Rules to Static Analysis

**Strict Persistency Checking Rules**

- **Model Violations**
  - Every write is followed by a flush
  - Every flush is preceded by a single write
  - Every flush should have a preceding write
  - Flushes should flush different addresses
  - Transactions must have at least one write

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>4</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>5</td>
<td>state</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>6</td>
<td>owners</td>
</tr>
<tr>
<td>Flush</td>
<td>7</td>
<td>owners</td>
</tr>
<tr>
<td>Fence</td>
<td>7</td>
<td>owners</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>9</td>
<td>new_level</td>
</tr>
<tr>
<td>Write</td>
<td>10</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>11</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>11</td>
<td>state</td>
</tr>
</tbody>
</table>

**Traces**

Systems Platform Research Group at UIUC
Applying our Checking Rules to Static Analysis

Strict Persistency Checking Rules

Performance Bugs Model Violations

Every write is followed by a flush
Every flush is preceded by a single write
Every flush should have a preceding write
Flushes should flush different addresses
Transactions must have at least one write

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>4</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>Write</td>
<td>6</td>
<td>owners</td>
</tr>
<tr>
<td>Flush</td>
<td>7</td>
<td>owners</td>
</tr>
<tr>
<td>Fence</td>
<td>7</td>
<td>owners</td>
</tr>
<tr>
<td>Write</td>
<td>9</td>
<td>new_level</td>
</tr>
<tr>
<td>Write</td>
<td>10</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>11</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>11</td>
<td>state</td>
</tr>
</tbody>
</table>
## Applying our Checking Rules to Static Analysis

### Performance Bugs Model Violations

- Every write is followed by a flush
- Every flush is preceded by a single write
- Every flush should have a preceding write
- Flushes should flush different addresses
- Transactions must have at least one write

### Traces

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>4</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>5</td>
<td>state</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>6</td>
<td>owners</td>
</tr>
<tr>
<td>Flush</td>
<td>7</td>
<td>owners</td>
</tr>
<tr>
<td>Fence</td>
<td>7</td>
<td>owners</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>9</td>
<td>new_level</td>
</tr>
<tr>
<td>Write</td>
<td>10</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>11</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>11</td>
<td>state</td>
</tr>
</tbody>
</table>

---

**Strict Persistency Checking Rules**

Systems Platform Research Group at UIUC
Applying our Checking Rules to Static Analysis

**Strict Persistency Checking Rules**

- Every write is followed by a flush
- Every flush is preceded by a single write
- Every flush should have a preceding write
- Flushes should flush different addresses
- Transactions must have at least one write

**Performance Bugs Model Violations**

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>4</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>5</td>
<td>state</td>
</tr>
</tbody>
</table>

| Write | 6    | owners |
| Flush | 7    | owners |
| Fence | 7    | owners |

| Write | 9    | new_level |
| Write | 10   | state     |
| Flush | 11   | state     |
| Fence | 11   | state     |

Traces
Applying our Checking Rules to Static Analysis

**Performance Bugs Model Violations**

- Every write is followed by a flush: **X**
- Every flush is preceded by a single write: **X**
- Every flush should have a preceding write: **✓**
- Flushes should flush different addresses: **✓**
- Transactions must have at least one write: **✓**

**Strict Persistency Checking Rules**

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>4</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>5</td>
<td>state</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>6</td>
<td>owners</td>
</tr>
<tr>
<td>Flush</td>
<td>7</td>
<td>owners</td>
</tr>
<tr>
<td>Fence</td>
<td>7</td>
<td>owners</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>9</td>
<td>new_level</td>
</tr>
<tr>
<td>Write</td>
<td>10</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>11</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>11</td>
<td>state</td>
</tr>
</tbody>
</table>

Traces
Applying our Checking Rules to Static Analysis

**Strict Persistency Checking Rules**

<table>
<thead>
<tr>
<th>Op</th>
<th>Line</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>4</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>Write</td>
<td>6</td>
<td>owners</td>
</tr>
<tr>
<td>Flush</td>
<td>7</td>
<td>owners</td>
</tr>
<tr>
<td>Fence</td>
<td>7</td>
<td>owners</td>
</tr>
<tr>
<td>Write</td>
<td>9</td>
<td>new_level</td>
</tr>
<tr>
<td>Write</td>
<td>10</td>
<td>state</td>
</tr>
<tr>
<td>Flush</td>
<td>11</td>
<td>state</td>
</tr>
<tr>
<td>Fence</td>
<td>11</td>
<td>state</td>
</tr>
</tbody>
</table>

**Traces**

- Every flush should have a preceding write
- Flashes should flush different addresses
- Transactions must have at least one write

**Model Violations**

- Every write is followed by a flush
- Every flush is preceded by a single write

**Performance Bugs Model Violations**

- Every flush should have a preceding write
- Flashes should flush different addresses
- Transactions must have at least one write
Dynamic Analysis for Epoch and Strand Persistency

Higher Possible Performance
Dynamic Analysis for Epoch and Strand Persistency

Higher Possible Performance

Read-after-Write Dependencies
Dynamic Analysis for Epoch and Strand Persistency

- Higher Possible Performance
- Read-after-Write Dependencies
- Write-after-Write Dependencies
Dynamic Analysis for Epoch and Strand Persistency

Dynamic Analysis
Dynamic Analysis for Epoch and Strand Persistency

Dynamic Analysis

High Overhead
Dynamic Analysis for Epoch and Strand Persistency

Dynamic Analysis

High Overhead
Dynamic Analysis for Epoch and Strand Persistency

Dynamic Analysis

High Overhead

Use DSG to track only persistent objects!
Dynamic Analysis for Epoch and Strand Persistency

Dynamic Analysis

Use DSG to track only persistent objects!

Reuse existing library annotations!

High Overhead
Checking for Epoch and Strand Violations

Epoch 1
begin_epoch;
x = a;
y = b;
barrier;
z = x + y;
end_epoch;

Epoch 2
begin_epoch;
w = c;
v = d;
barrier;
u = x + v*w;
end_epoch;
Checking for Epoch and Strand Violations

Epoch 1

```
begin_epoch;
x = a;
y = b;
barrier;
z = x + y;
end_epoch;
```

Epoch 2

```
begin_epoch;
w = c;
v = d;
barrier;
u = x + v*w;
end_epoch;
```

Start tracking upon epoch annotations.
Checking for Epoch and Strand Violations

Epoch 1

begin_epoch;
x = a;
y = b;
barrier;
z = x + y;
end_epoch;

Epoch 2

begin_epoch;
w = c;
v = d;
barrier;
u = x + v*w;
end_epoch;
Checking for Epoch and Strand Violations

Epoch 1
begin_epoch;
 x = a;
 y = b;
 barrier;
 z = x + y;
end_epoch;

Epoch 2
begin_epoch;
 w = c;
 v = d;
 barrier;
 u = x + v*w;
end_epoch;

Shadow Segment

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>v</td>
<td>w</td>
</tr>
<tr>
<td>x</td>
<td>y</td>
<td>z</td>
</tr>
</tbody>
</table>

Only include persistent object in the shadow segment
Checking for Epoch and Strand Violations

Epoch 1

```plaintext
begin_epoch;
x = a;
y = b;
barrier;
z = x + y;
end_epoch;
```

Epoch 2

```plaintext
begin_epoch;
w = c;
v = d;
barrier;
u = x + v*w;
end_epoch;
```

Shadow Segment

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
</tr>
<tr>
<td>v</td>
</tr>
<tr>
<td>w</td>
</tr>
<tr>
<td>x:1</td>
</tr>
<tr>
<td>y</td>
</tr>
<tr>
<td>z</td>
</tr>
</tbody>
</table>
Checking for Epoch and Strand Violations

Epoch 1
begin_epoch;
  x = a;
  y = b;
  barrier;
  z = x + y;
end_epoch;

Epoch 2
begin_epoch;
  w = c;
  v = d;
  barrier;
  u = x + v*w;
end_epoch;

Shadow Segment

+---+---+---+---+
<table>
<thead>
<tr>
<th>u</th>
<th>v</th>
<th>w</th>
<th>x:1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>y:1</td>
</tr>
</tbody>
</table>
+---+---+---+---+

z
Checking for Epoch and Strand Violations

Epoch 1
begin_epoch;
x = a;
y = b;
barrier;
z = x + y;
end_epoch;

Epoch 2
begin_epoch;
w = c;
v = d;
barrier;
u = x + v*w;
end_epoch;

Shadow Segment

<table>
<thead>
<tr>
<th>Shadow Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
</tr>
<tr>
<td>v</td>
</tr>
<tr>
<td>w</td>
</tr>
<tr>
<td>x:1</td>
</tr>
<tr>
<td>y:1</td>
</tr>
<tr>
<td>z:1</td>
</tr>
</tbody>
</table>
Checking for Epoch and Strand Violations

Epoch 1

```
begin_epoch;
x = a;
y = b;
barrier;
z = x + y;
end_epoch;
```

Epoch 2

```
begin_epoch;
w = c;
v = d;
barrier;
u = x + v*w;
end_epoch;
```

Shadow Segment

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>v</td>
<td>w:2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x:1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>y:1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>z:1</td>
</tr>
</tbody>
</table>
Checking for Epoch and Strand Violations

Epoch 1
begin_epoch;
  x = a;
  y = b;
  barrier;
  z = x + y;
end_epoch;

Epoch 2
begin_epoch;
  w = c;
  v = d;
  barrier;
  u = x + v* w;
end_epoch;

Shadow Segment

<table>
<thead>
<tr>
<th></th>
<th>u</th>
<th>v:2</th>
<th>w:2</th>
<th>x:1</th>
<th>y:1</th>
<th>z:1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Checking for Epoch and Strand Violations

Epoch 1

\begin{align*}
\text{begin\_epoch;} \\
x &= a; \\
y &= b; \\
\text{barrier;} \\
z &= x + y; \\
\text{end\_epoch;} 
\end{align*}

Epoch 2

\begin{align*}
\text{begin\_epoch;} \\
w &= c; \\
v &= d; \\
\text{barrier;} \\
u &= x + v \times w; \\
\text{end\_epoch;} 
\end{align*}

Shadow Segment

\begin{array}{c}
\text{u:2} \\
\text{v:2} \\
\text{w:2} \\
\text{x:1,2} \\
\text{y:1} \\
\text{z:1} 
\end{array}
Checking for Epoch and Strand Violations

Epoch 1

\begin{align*}
\text{begin\_epoch;} \\
x &= a; \\
y &= b; \\
\text{barrier;} \\
z &= x + y; \\
\text{end\_epoch;}
\end{align*}

Epoch 2

\begin{align*}
\text{begin\_epoch;} \\
w &= c; \\
v &= d; \\
\text{barrier;} \\
u &= x + v*w; \\
\text{end\_epoch;}
\end{align*}

Shadow Segment

\begin{tabular}{|c|}
\hline
u:2 \\
v:2 \\
w:2 \\
x:1,2 \\
y:1 \\
z:1 \\
\hline
\end{tabular}
Checking for Epoch and Strand Violations

Epoch 1
begin_epoch;
x = a;
y = b;
barrier;
z = x + y;
end_epoch;

Epoch 2
begin_epoch;
w = c;
v = d;
barrier;
u = x + v*w;
end_epoch;

Shadow Segment

<table>
<thead>
<tr>
<th>Variable</th>
<th>Shadow Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>2</td>
</tr>
<tr>
<td>v</td>
<td>2</td>
</tr>
<tr>
<td>w</td>
<td>2</td>
</tr>
<tr>
<td>x</td>
<td>1,2</td>
</tr>
<tr>
<td>y</td>
<td>1</td>
</tr>
<tr>
<td>z</td>
<td>1</td>
</tr>
</tbody>
</table>

End tracking with end of epochs
Checking for Epoch and Strand Violations

Epoch 1

\[
\text{begin\_epoch;} \\
x = a; \\
y = b; \\
\text{barrier;} \\
z = x + y; \\
\text{end\_epoch;}
\]

Epoch 2

\[
\text{begin\_epoch;} \\
w = c; \\
v = d; \\
\text{barrier;} \\
u = x + v \times w; \\
\text{end\_epoch;}
\]

Shadow Segment

\[
\begin{array}{c}
u:2 \\
v:2 \\
w:2 \\
x:1,2 \\
y:1 \\
z:1 \\
\end{array}
\]
Checking for Epoch and Strand Violations

Epoch 1

```plaintext
begin_epoch;
x = a;
y = b;
barrier;
z = x + y;
end_epoch;
```

Epoch 2

```plaintext
begin_epoch;
w = c;
v = d;
barrier;
u = x + v*w;
end_epoch;
```

Shadow Segment

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>v</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>w</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>1,2</td>
<td></td>
</tr>
</tbody>
</table>

Accesses to x race and should be ordered!
Detecting Memory Persistency Bugs with DeepMC

NVM Program \(\rightarrow\) LLVM \(\rightarrow\) IR

\(\rightarrow\) Data Structure Graph
\(\rightarrow\) Control Flow Graph
\(\rightarrow\) Call Graph

\[+\]

Static Checking Rules
Static Checker

\[\rightarrow\]

Error Warnings

Systems Platform Research Group at UIUC
Detecting Memory Persistency Bugs with DeepMC

Dynamic component to catch strand persistency violations
Detecting Memory Persistency Bugs with DeepMC

- NVM Program
- LLVM
- IR
  - Data Structure Graph
  - Control Flow Graph
  - Call Graph
  - Automated Code Annotation
  - Static Checking Rules
  - Static Checker
  - Dynamic Analysis
  - Instrumented Program
  - Error Warnings
  - Strand Persistency Warnings
DeepMC
Implementation
DeepMC Implementation

Static Analysis
13k LoC on top of LLVM/Clang

Dynamic Analysis
450 LoC on top of ThreadSanitizer
DeepMC Implementation

Experimental Setup

Static Analysis
13k LoC on top of LLVM/Clang

Dynamic Analysis
450 LoC on top of ThreadSanitizer

Server
8 Intel Xeon(R), 3.3 GHz
16GB Main Memory
Ubuntu 18.04, Linux kernel 5.0
Clang/Clang++ 7.0.0, O3 optimization

Workloads
Memcached, Redis, Nstore
PMDK, PMFS, NVM-Direct, Mnemosyne
# New Persistency Bugs

<table>
<thead>
<tr>
<th>Library</th>
<th>File</th>
<th>Line</th>
<th>Bug Description</th>
<th>Location</th>
<th>Consequences</th>
<th>Years</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMDK v1.2</td>
<td>btree_map.c</td>
<td>365, 465</td>
<td>Flushing unmodified fields of tree node</td>
<td>EP</td>
<td>Perf. Overhead</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td>rbtree_map.c</td>
<td>259</td>
<td>Flushing unmodified fields of tree node</td>
<td>EP</td>
<td>Perf. Overhead</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td>pminvaders.c</td>
<td>249, 266, 351</td>
<td>Durable transaction without persistent writes</td>
<td>EP</td>
<td>Perf. Overhead</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td>hashmap_atomic.c</td>
<td>120, 264, 285, 496</td>
<td>Multiple epochs write to different fields of an object</td>
<td>EP</td>
<td>Model Violation</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td>obj_pmemlog_simple.c</td>
<td>207, 252</td>
<td>Multiple epochs write to different fields of an object</td>
<td>LIB</td>
<td>Model Violation</td>
<td>4.4</td>
</tr>
<tr>
<td>PMFS</td>
<td>super.c</td>
<td>542, 543, 584</td>
<td>Flushing unmodified fields of an object</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>3.2</td>
</tr>
<tr>
<td>NVM-Direct</td>
<td>nvm_locks.c</td>
<td>905</td>
<td>Durable transaction without persistent writes</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>5.3</td>
</tr>
<tr>
<td>v0.3</td>
<td>nvm_locks.c</td>
<td>1411</td>
<td>Flushing unmodified fields of an object</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>5.3</td>
</tr>
<tr>
<td></td>
<td>nvm_locks.c</td>
<td>932</td>
<td>Missing flush</td>
<td>LIB</td>
<td>Model Violation</td>
<td>5.3</td>
</tr>
<tr>
<td></td>
<td>nvm_heap.c</td>
<td>1675</td>
<td>Flushing unmodified fields of an object</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>5.3</td>
</tr>
<tr>
<td>Mnemosyne</td>
<td>phlog_base.c</td>
<td>132</td>
<td>Unflushed write</td>
<td>LIB</td>
<td>Model Violation</td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td>chhash.c</td>
<td>185, 270</td>
<td>Multiple writes to the same object in a transaction</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td>CHash.c</td>
<td>150</td>
<td>Multiple flushes to a persistent object</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>10.0</td>
</tr>
</tbody>
</table>
## New Persistency Bugs

<table>
<thead>
<tr>
<th>Library</th>
<th>File</th>
<th>Line</th>
<th>Bug Description</th>
<th>Location</th>
<th>Consequences</th>
<th>Years</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMDK v1.2</td>
<td>btree_map.c</td>
<td>365, 465</td>
<td>Flushing unmodified fields of tree node</td>
<td>EP</td>
<td>Perf. Overhead</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td>rbtree_map.c</td>
<td>259</td>
<td>Flushing unmodified fields of tree node</td>
<td>EP</td>
<td>Perf. Overhead</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td>pminvaders.c</td>
<td>249, 266, 351</td>
<td>Durable transaction without persistent writes</td>
<td>EP</td>
<td>Perf. Overhead</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td>hashmap_atomic.c</td>
<td>120, 264, 285, 496</td>
<td>Multiple epochs write to different fields of an object</td>
<td>EP</td>
<td>Model Violation</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td>obj_pmemlog_simple.c</td>
<td>207, 252</td>
<td>Multiple epochs write to different fields of an object</td>
<td>LIB</td>
<td>Model Violation</td>
<td>4.4</td>
</tr>
<tr>
<td>PMFS</td>
<td>super.c</td>
<td>542, 543, 584</td>
<td>Flushing unmodified fields of an object</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>3.2</td>
</tr>
<tr>
<td>NVM-Direct v0.3</td>
<td>nvm_locks.c</td>
<td>905</td>
<td>Durable transaction without persistent writes</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>5.3</td>
</tr>
<tr>
<td></td>
<td>nvm_locks.c</td>
<td>1411</td>
<td>Flushing unmodified fields of an object</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>5.3</td>
</tr>
<tr>
<td></td>
<td>nvm_locks.c</td>
<td>932</td>
<td>Missing flush</td>
<td>LIB</td>
<td>Model Violation</td>
<td>5.3</td>
</tr>
<tr>
<td></td>
<td>nvm_heap.c</td>
<td>1675</td>
<td>Flushing unmodified fields of an object</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>5.3</td>
</tr>
<tr>
<td>Mnomosyne</td>
<td>phlog_base.c</td>
<td>132</td>
<td>Unflushed write</td>
<td>LIB</td>
<td>Model Violation</td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td>chhash.c</td>
<td>185, 270</td>
<td>Multiple writes to the same object in a transaction</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td>CHash.c</td>
<td>150</td>
<td>Multiple flushes to a persistent object</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>10.0</td>
</tr>
</tbody>
</table>

1. **24 new bugs, 18 confirmed**
## New Persistency Bugs

<table>
<thead>
<tr>
<th>Library</th>
<th>File</th>
<th>Line</th>
<th>Bug Description</th>
<th>Location</th>
<th>Consequences</th>
<th>Years</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMDK v1.2</td>
<td>btree_map.c</td>
<td>365, 465</td>
<td>Flushing unmodified fields of tree node</td>
<td>EP</td>
<td>Perf. Overhead</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td>rbtree_map.c</td>
<td>259</td>
<td>Flushing unmodified fields of tree node</td>
<td>EP</td>
<td>Perf. Overhead</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td>pminvaders.c</td>
<td>249, 266, 351</td>
<td>Durable transaction without persistent writes</td>
<td>EP</td>
<td>Perf. Overhead</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td>hashmap_atomic.c</td>
<td>120, 264, 285, 496</td>
<td>Multiple epochs write to different fields of an object</td>
<td>EP</td>
<td>Model Violation</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td>obj_pmemlog_simple.c</td>
<td>207, 252</td>
<td>Multiple epochs write to different fields of an object</td>
<td>LIB</td>
<td>Model Violation</td>
<td>4.4</td>
</tr>
<tr>
<td>PMFS</td>
<td>super.c</td>
<td>542, 543, 584</td>
<td>Flushing unmodified fields of an object</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>3.2</td>
</tr>
<tr>
<td>NVM-Direct</td>
<td>nvm_locks.c</td>
<td>905</td>
<td>Durable transaction without persistent writes</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>5.3</td>
</tr>
<tr>
<td></td>
<td>nvm_locks.c</td>
<td>1411</td>
<td>Flushing unmodified fields of an object</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>5.3</td>
</tr>
<tr>
<td></td>
<td>nvm_locks.c</td>
<td>932</td>
<td>Missing flush</td>
<td>LIB</td>
<td>Model Violation</td>
<td>5.3</td>
</tr>
<tr>
<td></td>
<td>nvm_heap.c</td>
<td>1675</td>
<td>Flushing unmodified fields of an object</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>5.3</td>
</tr>
<tr>
<td>Mnemosyne</td>
<td>phlog_base.c</td>
<td>132</td>
<td>Unflushed write</td>
<td>LIB</td>
<td>Model Violation</td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td>chhash.c</td>
<td>185, 270</td>
<td>Multiple writes to the same object in a transaction</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td>CHash.c</td>
<td>150</td>
<td>Multiple flushes to a persistent object</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>10.0</td>
</tr>
</tbody>
</table>

### 1. 24 new bugs, 18 confirmed

### 2. 8 model violations, 16 performance bugs
## New Persistency Bugs

<table>
<thead>
<tr>
<th>Library</th>
<th>File</th>
<th>Line</th>
<th>Bug Description</th>
<th>Location</th>
<th>Consequences</th>
<th>Years</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMDK v1.2</td>
<td>btree_map.c</td>
<td>365, 465</td>
<td>Flushing unmodified fields of tree node</td>
<td>EP</td>
<td>Perf. Overhead</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td>rbtree_map.c</td>
<td>259</td>
<td>Flushing unmodified fields of tree node</td>
<td>EP</td>
<td>Perf. Overhead</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td>pminvaders.c</td>
<td>249, 266, 351</td>
<td>Durable transaction without persistent writes</td>
<td>EP</td>
<td>Perf. Overhead</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td>hashmap_atomic.c</td>
<td>120, 264, 285, 496</td>
<td>Multiple epochs write to different fields of an object</td>
<td>EP</td>
<td>Model Violation</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td>obj_pmemlog_simple.c</td>
<td>207, 252</td>
<td>Multiple epochs write to different fields of an object</td>
<td>LIB</td>
<td>Model Violation</td>
<td>4.4</td>
</tr>
<tr>
<td>PMFS</td>
<td>super.c</td>
<td>542, 543, 584</td>
<td>Flushing unmodified fields of an object</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>3.2</td>
</tr>
<tr>
<td>NVM-Direct</td>
<td>nvm_locks.c</td>
<td>905</td>
<td>Durable transaction without persistent writes</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>5.3</td>
</tr>
<tr>
<td>v0.3</td>
<td>nvm_locks.c</td>
<td>1411</td>
<td>Flushing unmodified fields of an object</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>5.3</td>
</tr>
<tr>
<td></td>
<td>nvm_locks.c</td>
<td>932</td>
<td>Missing flush</td>
<td>LIB</td>
<td>Model Violation</td>
<td>5.3</td>
</tr>
<tr>
<td></td>
<td>nvm_heap.c</td>
<td>1675</td>
<td>Flushing unmodified fields of an object</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>5.3</td>
</tr>
<tr>
<td>Mnemosyne</td>
<td>phlog_base.c</td>
<td>132</td>
<td>Unflushed write</td>
<td>LIB</td>
<td>Model Violation</td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td>chhash.c</td>
<td>185, 270</td>
<td>Multiple writes to the same object in a transaction</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td>CHash.c</td>
<td>150</td>
<td>Multiple flushes to a persistent object</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>10.0</td>
</tr>
</tbody>
</table>

1. **24 new bugs, 18 confirmed**
2. **8 model violations, 16 performance bugs**
# New Persistency Bugs

<table>
<thead>
<tr>
<th>Library</th>
<th>File</th>
<th>Line</th>
<th>Bug Description</th>
<th>Location</th>
<th>Consequences</th>
<th>Years</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMDK v1.2</td>
<td>btree_map.c</td>
<td>365, 465</td>
<td>Flushing unmodified fields of tree node</td>
<td>EP</td>
<td>Perf. Overhead</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td>rbtree_map.c</td>
<td>259</td>
<td>Flushing unmodified fields of tree node</td>
<td>EP</td>
<td>Perf. Overhead</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td>pminvaders.c</td>
<td>249, 266, 351</td>
<td>Durable transaction without persistent writes</td>
<td>EP</td>
<td>Perf. Overhead</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td>hashmap_atomic.c</td>
<td>120, 264, 285, 496</td>
<td>Multiple epochs write to different fields of an object</td>
<td>EP</td>
<td>Model Violation</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td>obj_pmemlog_simple.c</td>
<td>207, 252</td>
<td>Multiple epochs write to different fields of an object</td>
<td>LIB</td>
<td>Model Violation</td>
<td>4.4</td>
</tr>
<tr>
<td>PMFS</td>
<td>super.c</td>
<td>542, 543, 584</td>
<td>Flushing unmodified fields of an object</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>3.2</td>
</tr>
<tr>
<td>NVM-Direct v0.3</td>
<td>nvm_locks.c</td>
<td>905</td>
<td>Durable transaction without persistent writes</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>5.3</td>
</tr>
<tr>
<td></td>
<td>nvm_locks.c</td>
<td>1411</td>
<td>Flushing unmodified fields of an object</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>5.3</td>
</tr>
<tr>
<td></td>
<td>nvm_locks.c</td>
<td>932</td>
<td>Missing flush</td>
<td>LIB</td>
<td>Model Violation</td>
<td>5.3</td>
</tr>
<tr>
<td></td>
<td>nvm_heap.c</td>
<td>1675</td>
<td>Flushing unmodified fields of an object</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>5.3</td>
</tr>
<tr>
<td>Mnemosyne</td>
<td>phlog_base.c</td>
<td>132</td>
<td>Unflushed write</td>
<td>LIB</td>
<td>Model Violation</td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td>chhash.c</td>
<td>185, 270</td>
<td>Multiple writes to the same object in a transaction</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td>CHash.c</td>
<td>150</td>
<td>Multiple flushes to a persistent object</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>10.0</td>
</tr>
</tbody>
</table>

1. **24 new bugs, 18 confirmed**
2. **8 model violations, 16 performance bugs**
3. **18 statically detected, 6 dynamically detected**
# New Persistency Bugs

<table>
<thead>
<tr>
<th>Library</th>
<th>File</th>
<th>Line</th>
<th>Bug Description</th>
<th>Location</th>
<th>Consequences</th>
<th>Years</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMDK v1.2</td>
<td>btree_map.c</td>
<td>365, 465</td>
<td>Flushing unmodified fields of tree node</td>
<td>EP</td>
<td>Perf. Overhead</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td>rbtree_map.c</td>
<td>259</td>
<td>Flushing unmodified fields of tree node</td>
<td>EP</td>
<td>Perf. Overhead</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td>pminvaders.c</td>
<td>249, 266, 351</td>
<td>Durable transaction without persistent writes</td>
<td>EP</td>
<td>Perf. Overhead</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td>hashmap_atomic.c</td>
<td>120, 264, 285, 496</td>
<td>Multiple epochs write to different fields of an object</td>
<td>EP</td>
<td>Model Violation</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td>obj_pmemlog_simple.c</td>
<td>207, 252</td>
<td>Multiple epochs write to different fields of an object</td>
<td>LIB</td>
<td>Model Violation</td>
<td>4.4</td>
</tr>
<tr>
<td>PMFS</td>
<td>super.c</td>
<td>542, 543, 584</td>
<td>Flushing unmodified fields of an object</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>3.2</td>
</tr>
<tr>
<td>NVM-Direct v0.3</td>
<td>nvm_locks.c</td>
<td>905</td>
<td>Durable transaction without persistent writes</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>5.3</td>
</tr>
<tr>
<td></td>
<td>nvm_locks.c</td>
<td>1411</td>
<td>Flushing unmodified fields of an object</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>5.3</td>
</tr>
<tr>
<td></td>
<td>nvm_locks.c</td>
<td>932</td>
<td>Missing flush</td>
<td>LIB</td>
<td>Model Violation</td>
<td>5.3</td>
</tr>
<tr>
<td></td>
<td>nvm_heap.c</td>
<td>1675</td>
<td>Flushing unmodified fields of an object</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>5.3</td>
</tr>
<tr>
<td>Mnemosyne</td>
<td>phlog_base.c</td>
<td>132</td>
<td>Unflushed write</td>
<td>LIB</td>
<td>Model Violation</td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td>chhash.c</td>
<td>185, 270</td>
<td>Multiple writes to the same object in a transaction</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td>CHash.c</td>
<td>150</td>
<td>Multiple flushes to a persistent object</td>
<td>LIB</td>
<td>Perf. Overhead</td>
<td>10.0</td>
</tr>
</tbody>
</table>

1. 24 new bugs, 18 confirmed
2. 8 model violations, 16 performance bugs
3. 18 statically detected, 6 dynamically detected
4. Common performance bug was flushing unmodified data!
## Impact of DeepMC on Performance

Static analysis introduces minimal compilation overhead

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Baseline (secs)</th>
<th>Compilation with DeepMC (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memcached</td>
<td>8.5</td>
<td>11.9</td>
</tr>
<tr>
<td>Redis</td>
<td>54.9</td>
<td>62.4</td>
</tr>
<tr>
<td>NStore</td>
<td>31.9</td>
<td>35.6</td>
</tr>
</tbody>
</table>
Impact of DeepMC on Performance

Dynamic analysis adds minimal performance overhead!
Limitations of DeepMC

Lack of dynamic context for DSA

Certain memory references cannot be resolved statically!
Limitations of DeepMC

Lack of dynamic context for DSA

Approved violations of the model

Programmers may violate the model intentionally for performance
Limitations of DeepMC

- Lack of dynamic context for DSA
- Approved violations of the model
- Checking rules can be further enriched

Checking rules can be enriched as models are added and refined
DeepMC
Summary
DeepMC
Summary

Study Bugs in NVM Programs
DeepMC Summary

- Study Bugs in NVM Programs
- Develop Static and Dynamic Detection Tools
DeepMC Summary

Study Bugs in NVM Programs

Develop Static and Dynamic Detection Tools

Discover 24 new persistency bugs in NVM Programs
Thank You!

Benjamin Reidys, Jian Huang
breidys2@illinois.edu

Systems Platform Research Group