Durinn: Adversarial Memory and Thread Interleaving for Detecting Durable Linearizability Bugs

Xinwei Fu*, Dongyoon Lee+, Changwoo Min*

*VT VIRGINIA TECH. + Stony Brook University
Summary

• Crash-consistent software without paying storage overhead
• Writing crash-consistent programs is error-prone

• NVM Correctness Condition: Durable Linearizability

Durinn
• The first Durable Linearizability checker
• Three Durable Linearizability bug patterns
• Adversarial crash state and thread interleaving construction
• Likely-Linearization Point inference
• Detected 27 (15 new) bugs
Outline

• Introduction

• Durinn

• Evaluation

• Conclusion
**NVM Correctness Condition: Durable Linearizability**

**Durable Linearizability** requires:
- (C1) without a crash, all operations are *Linearizable*

**Linearizability** requires that all operations:
- take effect instantaneously at a *program point* (*Linearization Point*)
- and that point is between the operation begin and end

```
insert (k, v1)
T1: get (k)
T2: return v1

insert (k, v2)
get (k)
return v1 or v2
```
NVM Correctness Condition: Durable Linearizability

Durable Linearizability requires:
• (C1) without a crash, all operations are linearizable
• (C2) completed operations before a crash → All semantic
• (C3) incomplete operations upon a crash → All or nothing semantic

```
T1: insert (k, v1)  insert (k, v2)  crash
T2: get (k)  get (k)  recovery
    return v1  return v1 or v2  (All or nothing semantic)
    return v2  return v2  (All semantic)
```

```
NVM Correctness Condition: Durable Linearizability

Durable Linearizability describes correct operation behaviors:

- Crash state
- Thread interleaving

Any incorrect operation behavior leads to a Durable Linearizability bug.

Diagram:

- T1: insert (k, v1) → insert (k, v2) → crash → get (k) → return v2
- T2: get (k) → return v2

(All semantic)
Our Contributions

Existing Solutions
• Linearizability testing tools
• NVM-specific crash-consistency bug detectors

Durinn
• Three Durable Linearizability bug patterns
• Adversarial NVM State and Thread Interleaving Construction
• Likely-Linearization Point Inference
Outline

- Introduction
- Durinn
- Evaluation
- Conclusion
Durinn Overview

- **Linearization Point** → understand operation behaviors

- **Key idea 1:** three durable linearizability bug patterns
- **Key idea 2:** adversarial test for both crash state and thread interleaving

---

**Diagram:**

- **Program Test Case** → Tracing Memory Access
- **Trace** → Likely-LP Inference → Likely LPs
- **Adversarial Test:** NVM States Thread Interleave
- **NVM States** → Thread Interleave → DL Validation → DL Bugs
Outline

• Introduction

• Durinn

  • *Durable Linearizability Bugs and Adversarial Testing*

  • Likely-Linearization Point Inference

• Evaluation

• Conclusion
The gap between LP and DP

Linearization Point ●

- a program point where an operation takes effect and its effects become visible

Durability Point ●

- a program point where the effect becomes persisted

---

**Diagram:**

- **Operation** timeline
  - Linearization Point (LP)
  - Durability Point (DP)
  - CAS(T)
  - Persist(T)
- **Time**
  - R1: not visible
  - R2: visible but not durable
  - R3: both visible and durable

---
Correctness condition:
A crash between LP and DP, if the effect has been observed before crash, the operation should preserve All Semantic.

Some or all of the writes are not persisted

Incorrectly returns NULL
returns V (All Semantic)
Adversarial test for DL3 (Visible-But-Not-Durable) Bug

Correctness condition:
A crash between LP and DP, if the effect has been observed before crash, the operation should preserve **All Semantic.**

Correctness condition:

- **Linearization Point**
- **Durability Point**

T1: insert (K, V)

W(K) \[\rightarrow\] W(V)

T2: get (K)

W(T)

if (R(T)) \{ R(K) R(V) \} returns V

All legal crash states

<table>
<thead>
<tr>
<th>K</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>Persisted</td>
<td>Persisted</td>
</tr>
<tr>
<td>Persisted</td>
<td>Unpersisted</td>
</tr>
<tr>
<td>Unpersisted</td>
<td>Persisted</td>
</tr>
<tr>
<td>Unpersisted</td>
<td>Unpersisted</td>
</tr>
</tbody>
</table>

Worst case

- A pair of racy operations
- A specific thread interleaving
Adversarial test for DL3 Bug

**Single-threaded trace**

```
insert (...)  
insert (...)  
delete (...)  
insert. (...)  

insert (K, V):  
W (K)  
W (V)  
W (T)  

get (K):  
if ( R (T) )  
R (K)  
R (V)  
```

**Race**

**Main Thread**

```
insert (...)  
insert (...)  
delete (...)  
insert. (...)  

```

**Thread 1**

```
insert (K, V):  
W (K)  
W (V)  
W (T)  

```

**Thread 2**

```
get (K):  
if ( R (T) )  
R (K)  
R (V)  
```

**crash**
Outline

• Introduction

• Durinn
  • Durable Linearizability Bugs and Adversarial Testing
    • Likely-Linearization Point Inference

• Evaluation

• Conclusion
Likely-Linearization Point Inference

(1) Atomic Instruction

Lock-free Insert

......

Atomic Inst

- CAS (T)

......

Linearization Point

(2) Guarded-Protection Pattern

Insert (K, V)

- W(K)
- W(V)
- Persist(KV)

Set guardian

- W(T)
- Persist(T)

Get (K)

if ( R(T) ) {
- R(K)
- R(V)
}

(3) Publish-after-Initialization

Insert

- node = malloc()

Initialization

- node.key = k
- node.val = v

......

Publish

- prev.next = node

No LPs.
Evaluation

Tested Applications:
• 13 concurrent NVM data structures
  • Array, queue, linked list, skip list, hashtable, radix tree, B+tree and trie
• Low-level persistence primitives and high-level persistence transactions
• Lock-based and lock-free
• 1000 operations generated by AFL++ fuzzer

Evaluation Questions:
• Can Durinn detect new bugs?
• How effective and sound is Durinn’s likely-LP technique?
• Does Durinn outperform the state-of-the-art?
Detected DL bugs

## Detected 10 DL1 bugs, 7 DL2 bugs, and 10 DL3 bugs.

<table>
<thead>
<tr>
<th>Name (Total #Bugs)</th>
<th>Bug ID</th>
<th>New</th>
<th>Confirm</th>
<th>Code</th>
<th>Type</th>
<th>Description</th>
<th>Impact</th>
<th>Fix strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>P-LF-BST (1)</td>
<td>1</td>
<td>✓</td>
<td>✓</td>
<td>BSTAravindTraverse.h:331</td>
<td>DL1</td>
<td>Missing persistence primitives</td>
<td>Points to garbage</td>
<td>add persistence primitives</td>
</tr>
<tr>
<td>P-LF-Hash (1)</td>
<td>2</td>
<td>✓</td>
<td>✓</td>
<td>ListTraverse.h:212</td>
<td>DL1</td>
<td>Missing persistence primitives</td>
<td>Points to garbage</td>
<td>add persistence primitives</td>
</tr>
<tr>
<td>P-LF-List (1)</td>
<td>3</td>
<td>✓</td>
<td>✓</td>
<td>ListTraverse.h:212</td>
<td>DL1</td>
<td>Missing persistence primitives</td>
<td>Points to garbage</td>
<td>add persistence primitives</td>
</tr>
<tr>
<td>P-LF-Skiplist(1)</td>
<td>4</td>
<td>✓</td>
<td>✓</td>
<td>SkiplistTraverse.h:218</td>
<td>DL1</td>
<td>Missing persistence primitives</td>
<td>Points to garbage</td>
<td>add persistence primitives</td>
</tr>
<tr>
<td>P-LF-Queue(1)</td>
<td>5</td>
<td>✓</td>
<td>✓</td>
<td>DurableQueue.h:174</td>
<td>DL1</td>
<td>Missing persistence primitives</td>
<td>Points to garbage</td>
<td>add persistence primitives</td>
</tr>
<tr>
<td>CCEH (2)</td>
<td>6</td>
<td>✓</td>
<td>✓</td>
<td>CCEH_MSB.cpp:280</td>
<td>DL2</td>
<td>Atomicity in rehashing</td>
<td>Lost key-value</td>
<td>fix concurrency control/help persist consistency-recoverable design</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>✓</td>
<td>✓</td>
<td>CCEH_MSB.cpp:103</td>
<td>DL3</td>
<td>Incorrect concurrency control</td>
<td>Lost key-value</td>
<td>fix concurrency control/help persist consistency-recoverable design</td>
</tr>
<tr>
<td>FAST-FAIR (5)</td>
<td>8</td>
<td>✓</td>
<td>✓</td>
<td>btrees.h:955,979</td>
<td>DL3</td>
<td>Incorrect concurrency control</td>
<td>Lost key-value</td>
<td>fix concurrency control/help persist consistency-recoverable design</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>✓</td>
<td>✓</td>
<td>btrees.h:955,1007</td>
<td>DL3</td>
<td>Incorrect concurrency control</td>
<td>Lost key-value</td>
<td>fix concurrency control/help persist consistency-recoverable design</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>✓</td>
<td>✓</td>
<td>btrees.h:224</td>
<td>DL2</td>
<td>Partial inconsistency is never recovered</td>
<td>Unable to recover</td>
<td>add persistence primitives</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>✓</td>
<td>✓</td>
<td>btrees.h:213</td>
<td>DL2</td>
<td>Atomicity in node splitting</td>
<td>Unable to recover</td>
<td>add persistence primitives</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>✓</td>
<td>✓</td>
<td>btrees.h:576</td>
<td>DL3</td>
<td>Incorrect concurrency control</td>
<td>Lost key-value</td>
<td>fix concurrency control/help persist consistency-recoverable design</td>
</tr>
<tr>
<td>P-ART (4)</td>
<td>13</td>
<td>✓</td>
<td>✓</td>
<td>Tree.cpp:35,258</td>
<td>DL3</td>
<td>Incorrect concurrency control</td>
<td>Lost key-value</td>
<td>fix concurrency control/help persist consistency-recoverable design</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>✓</td>
<td>✓</td>
<td>Tree.cpp:35,384</td>
<td>DL3</td>
<td>Incorrect concurrency control</td>
<td>Lost key-value</td>
<td>fix concurrency control/help persist consistency-recoverable design</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>✓</td>
<td>✓</td>
<td>N16.cpp:15</td>
<td>DL2</td>
<td>Atomicity between metadata and key-value</td>
<td>Unable to recover</td>
<td>add persistence primitives</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>✓</td>
<td>✓</td>
<td>N4.cpp:17</td>
<td>DL2</td>
<td>Atomicity between metadata and key-value</td>
<td>Unable to recover</td>
<td>add persistence primitives</td>
</tr>
<tr>
<td>P-CLHT (3)</td>
<td>17</td>
<td>✓</td>
<td>✓</td>
<td>clht_lb_res.c:315,370</td>
<td>DL3</td>
<td>Incorrect concurrency control</td>
<td>Lost key-value</td>
<td>fix concurrency control/help persist consistency-recoverable design</td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>✓</td>
<td>✓</td>
<td>clht_lb_res.c:315,468</td>
<td>DL3</td>
<td>Incorrect concurrency control</td>
<td>Lost key-value</td>
<td>fix concurrency control/help persist consistency-recoverable design</td>
</tr>
<tr>
<td></td>
<td>19</td>
<td>✓</td>
<td>✓</td>
<td>clht_lb_res.c:166</td>
<td>DL1</td>
<td>Missing persistence primitives</td>
<td>Lost key-value</td>
<td>add persistence primitives</td>
</tr>
<tr>
<td>P-HOT (4)</td>
<td>20</td>
<td>✓</td>
<td>✓</td>
<td>HOTRowex.h:61,84</td>
<td>DL3</td>
<td>Incorrect concurrency control</td>
<td>Lost key-value</td>
<td>fix concurrency control/help persist consistency-recoverable design</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td>✓</td>
<td>✓</td>
<td>TwoEntriesNode.h:30</td>
<td>DL1</td>
<td>Missing persistence primitives</td>
<td>Points to garbage</td>
<td>add persistence primitives</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>✓</td>
<td>✓</td>
<td>HOTRowexNode.h:315</td>
<td>DL1</td>
<td>Missing persistence primitives</td>
<td>Points to garbage</td>
<td>add persistence primitives</td>
</tr>
<tr>
<td></td>
<td>23</td>
<td>✓</td>
<td>✓</td>
<td>HOTRowex.h:270</td>
<td>DL1</td>
<td>Missing persistence primitives</td>
<td>Points to garbage</td>
<td>add persistence primitives</td>
</tr>
<tr>
<td>P-Masstree (3)</td>
<td>24</td>
<td>✓</td>
<td>✓</td>
<td>masstree.h:1837,744</td>
<td>DL3</td>
<td>Incorrect concurrency control</td>
<td>Lost key-value</td>
<td>fix concurrency control/help persist consistency-recoverable design</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>✓</td>
<td>✓</td>
<td>masstree.h:1837,941</td>
<td>DL3</td>
<td>Incorrect concurrency control</td>
<td>Lost key-value</td>
<td>fix concurrency control/help persist consistency-recoverable design</td>
</tr>
<tr>
<td></td>
<td>26</td>
<td>✓</td>
<td>✓</td>
<td>masstree.h:1378</td>
<td>DL2</td>
<td>Atomicity in node splitting</td>
<td>Unable to recover</td>
<td>logging/transaction</td>
</tr>
<tr>
<td>pmdk-array (1)</td>
<td>27</td>
<td>✓</td>
<td>✓</td>
<td>array.c:486</td>
<td>DL2</td>
<td>Atomicity between metadata and data</td>
<td>Unable to recover</td>
<td>logging/transaction</td>
</tr>
</tbody>
</table>
Effectiveness and soundness of Likely-Linearization Point Inference

- Durinn only tests 35% and 82% of Total Stores
- Durinn did not miss true Linearization points
Comparison against Witcher

**Bug Detection:**
- Durinn reports 10 DL3 bugs that Witcher missed
- Durinn reduces the test space of thread interleaving

**Test Space Reduction:**
- Witcher performs several times more tests than Durinn
- Durinn only adversarially tests worst-case scenarios
Outline

• Introduction

• Durinn

• Evaluation

• Conclusion
Conclusion

Durinn
• The first Durable Linearizability checker
• Three Durable Linearizability bug patterns
• Adversarial Crash State and Thread Interleaving Construction
• Likely-Linearization Point inference
• Detected 27 (15 new) bugs
Durinn: Adversarial Memory and Thread Interleaving for Detecting Durable Linearizability Bugs

Xinwei Fu*, Dongyoon Lee+, Changwoo Min*

*Virginia Tech. +Stony Brook University