Fine-Grained Checkpointing with In-Cache-Line Logging

Nachshon Cohen
Hillel Avni

David T. Aksun
James R. Larus
Background: Non-Volatile Memory

- DRAM-like performance, disk-like durability
  - Data is *retained* after shutting down the machine
    - Planned or *unexpected*
Challenge: Cache Reorder Writes

Application

Cache

Data structures in NVM

Graph:
- Node 7
- Node 2
- Node 5
- Node 9
- Node 6
- Node 5
- Node 11
- Node 4
Durable Data Structures

- **Challenge**: design a *durable* data structure for NVM
- **Subject to**: cache can *reorder* writes
- **And**: without *reducing performance* a lot
Existing Approaches

- Log modifications (undo log: old value, redo log: new value)
- Explicitly force a write back (flush) modified cache lines
  - Both log and data

Access to memory is expensive

Can we do better?
Our Approach

- Algorithm

  - Periodic persistency

  - In Cache Line Log (InCLL): our novel contribution

- Zero explicit writes back on the fast path of the data structure
Periodic Persistency

- Flush entire cache infrequently (e.g., every 64ms)
  - E.g., x86’s `wbinvd` instruction
Periodic Persistency

- Flush entire cache infrequently (e.g., every 64ms)
  - E.g., x86’s \textit{wbinvd} instruction
- Return to a consistent state at the end of an epoch
  - Using \textit{undo log}
Ensuring Consistent State: B+ Tree

- put(key: 10, value: 12)
Ensuring Consistent State: B+ Tree

- put(key: 10, value: 12)
- node.value[0] = 12

Can we avoid write backs?

ModifyDataStructureNode
1. Log <- oldValue
2. WriteBack(Log)
3. Node <- newValue
Concurrency

- Modify multiple variables is hard
  - Requires a lock or TM
- Modify a single variable is easy
  - Fetch and Add
  - Compare and Swap
In Cache Line Log

- A cache line is evicted to memory atomically.
- Embed the log inside the same cache line as modified node.
- No explicit write-back.
In Cache Line Log

- A cache line is evicted to memory atomically
- Embed the log inside **the same cache line** as modified node
- No explicit write-back
How In Cache Line Log Works

Cache

No write back

Implicit write back
In Cache Line Log
enables recovery
without explicit write backs
In Cache Line Log: Drawback

- Capacity is very limited
External Undo Log at Node Granularity

- Node is modified two times
  - Probably it will be modified again during the epoch
- Log entire node, explicit write back
  - Subsequent modifications (during same epoch) do not require logging
External Undo Log + In Cache Line Log

- First modification: use InCLL

<table>
<thead>
<tr>
<th>Node</th>
<th>12</th>
<th>1</th>
<th>40</th>
<th>70</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node1</td>
<td>12</td>
<td>1</td>
<td>40</td>
<td>70</td>
<td>11</td>
</tr>
<tr>
<td>Node2</td>
<td>12</td>
<td>1</td>
<td>40</td>
<td>70</td>
<td>11</td>
</tr>
<tr>
<td>Node3</td>
<td>12</td>
<td>1</td>
<td>40</td>
<td>70</td>
<td>11</td>
</tr>
<tr>
<td>Node5</td>
<td>12</td>
<td>1</td>
<td>40</td>
<td>70</td>
<td>11</td>
</tr>
</tbody>
</table>
External Undo Log + In Cache Line Log

- First modification: use InCLL
- 2+ modifications: use external log

On average, $1/#\text{modifications}$ explicit write backs
External Undo Log + In Cache Line Log

- First modification: use InCLL
  - Effective when modifications are sparse
  - Data structure is large
  - Key distribution is uniform

- 2+ modifications: use external log
  - Effective when modifications are dense
  - Splitting a B+ node
  - Modify a range of values
Many additional details, see paper
Implementation and Evaluation

- Incorporated into MassTree [Mao, Kohler, Morris, EuroSys’12]
  - B+ Tree/Trie with excellent performance
- Also made MassTree’s allocator durable with InCLL
  - Avoid dangling pointers and durable memory leaks
- Workloads
  - Ycsb A (50% writes), B (5% writes), C (0% writes), E (scans)
  - Key distribution: Uniform and Zipfian
Performance vs. Workload

![Performance vs. Workload Bar Chart](image-url)
Performance vs. NVM Latency

![Graph showing Performance vs. NVM Latency]

- **YCSB_A LOGGING Uniform**
- **YCSB_A INCLL Uniform**

**Throughput (Millions ops/sec)**

**Added Delay (ns)**
Conclusion

- **Explicit write backs (cache line flushes)** are expensive

- Use **In Cache Line Log**
  - Place log inside cache line and avoid explicit write back

- Plus: Periodic persistence, External log for second modification

- **Durability** with small overhead
Conclusion

- **Explicit write backs** are expensive
- Use **In Cache Line Log**
  - Place log inside cache line and avoid explicit write backs
- Plus: Periodic persistence, External log for modification
- **Durability** with **small overhead**

Questions?