Highlight Paper:

A Persistent Lock-Free Queue for Non-Volatile Memory (PPoPP’18)

Michal Friedman

Maurice Herlihy

Virendra Marathe

Erez Petrank

SYSTOR ‘19
THIS TALK

Concurrent Data Structure ➔ Non-Volatile Byte-Addressable Memory

- Platform & Challenge
- Definitions
- Queue designs
- Evaluation
Upon a crash Cache and Memory content is lost
Upon a crash Cache content is lost
Instead of writing blocks to disk, make our normal data structures persistent!
MAJOR PROBLEM: ORDERING NOT MAINTAINED

- Write \( x = 1 \)
- Write \( y = 1 \)
- Flush & \( x \)
- Flush & \( y \)

Due to implicit eviction:
Upon a crash, memory may contain \( y = 1 \) and \( x = 0 \).

\( O_2 \) can follow up on \( O_1 \), but only \( O_2 \) is reflected in the memory.
EXAMPLE

Suppose everything has been written except for this one pointer.

If a crash occurs, the memory will contain:

Head

Tail

7
CHALLENGE

CPU & Registers

High-Speed Cache

Challenge: make data persistent at minimal cost

Problem: Caches and registers are volatile.

- Usually don’t care what’s in the cache/memory
- Here we care!
- Flush some data to maintain consistency in memory - costly!
THE MODEL

- Main memory is non-volatile
- Caches and registers are volatile
- All threads crash together
  - New threads are created to continue the execution
Definitions

The queue designs

- Surprisingly many details and challenges
LINEARIZABILITY

- [HerlihyWing ‘90]
  - Each method call should appear to take effect instantaneously at some moment between its invocation and response

Thread 1
- \texttt{q.enq(5)}

Thread 2
- \texttt{q.enq(1)}

\texttt{q.deq()=5}

\texttt{q.deq()=1}

\texttt{time}
CORRECTNESS FOR NVM

- **Consistent state**

1. **Buffered Durable Linearizability**
   - [IzraelevitzMendesScott ’16]

2. **Durable Linearizability**
   - [IzraelevitzMendesScott ’16]

3. **Detectable Execution**
   - [FHerlihyMarathePetrank ’18]

Strength
DURABLE LINEARIZABILITY

- [IzraelevitzMendesScott '16]
  - Operations completed before the crash are recoverable (plus some overlapping operations)
  - Prefix of linearization order

Thread 1
- `q.enq(5)`
- `q.deq=(1)`

Thread 2
- `q.enq(1)`
- `q.deq=(5)`

Time
[IzralevitzMendesScott ‘16]

- Some **prefix** of a linearization ordering
- Support: a “sync” persists all previous operations
Even in durable-linearizability - no ability to determine completion.

**Detectable execution** extends definitions:

- Provide a mechanism to check if operation completed
- Implementation example: a persistent log
Three New Queue Designs

- Three lock-free queues for non-volatile memory
  [FHerlihyMarathePetrank ‘18]

  - **Relaxed**
    - A prefix of executed operations is recovered
      *(Buffered)*
  - **Durable**
    - All operations completed before the crash are recovered
      *(Durable)*
  - **Log**
    - Durable + can tell if an operation recovered
      *(Detectable)*

- Based on lock-free queue [MichaelScott ‘96]
MICHAEL AND SCOTT’S QUEUE (BASELINE)

- A Lock-Free queue
- The base algorithm for the queue in java.util.concurrent
- A common simple data structure, but
- Complicated enough to demonstrate the challenges
Enqueue (data):

1. Allocate a node with its values.
   
   1.a. Flush node content to memory. *(Initialization guideline.)*

2. Read *tail* and *tail->next* values.
   

3. Insert node to queue - CAS last pointer *ptr* point to it.
   
   3.a. Flush *ptr* to memory. *(Completion guideline.)*

4. Update *tail*.

---

**DURABLE ENQUEUE**

- [x] volatile
- [ ] non-volatile
Enqueue (data):
1. Allocate a node with its values.
   1.a. Flush node content to memory. (Initialization guideline.)
2. Read \textit{tail} and \textit{tail->next} values.
3. Insert node to queue - CAS last pointer \textit{ptr} point to it.
   3.a. Flush \textit{ptr} to memory. (\text{Completion} guideline.)
4. Update \textit{tail}.

For example, if this CAS fails due to concurrent activity, we need to be careful to maintain durable linearizability...
DURABLE ENQUEUE – MORE COMPLEX

- Enqueue (data):
  1. Allocate a node with its values.
     1.a. Flush node content to memory. (Initialization guideline.)
  2. Read tail and tail->next values.
  3. Insert node to queue - CAS last pointer ptr point to it.
     3.a. Flush ptr to memory. (Completion guideline.)
  4. Update tail.
Enqueue (data):

1. Allocate a node with its values.
   1.a. Flush node content to memory. (Initialization guideline.)
2. Read \textit{tail} and \textit{tail->next} values.
   2.a. Help: Update \textit{tail}.
3. Insert node to queue - CAS last pointer \textit{ptr} point to it.

Complete (and persist) previous operation:

5. Flush \textit{ptr} to memory. (Dependence guideline.)
6. Update \textit{tail}.
RELAXED QUEUE

- Buffered Durable linearizable
- Challenge 1: Obtain snapshot at sync() time
- Challenge 2: Making sync() concurrent

LOG QUEUE

- Durable linearizable
- Detectable execution
- Log operations
- More complicated dependencies and recovery
Compare the three queues: durable, relaxed, log and Michael and Scott’s queue

Platform: 4 AMD Opteron(TM) 6376 2.3GHz processors, 64 cores in total, Ubuntu 14.04.

Workload: threads run enqueue-dequeue pairs concurrently
EVALUATION - THROUGHPUT

Operations/Sec [Millions]

Not persistent

Persistent

Implementation details:
- Frequent sync: every 10 ops/thread
- Infrequent sync: every 1000 ops/thread
- Queue initial size: 1 M

Buffered durability less costly

Durability & detectable costly. Similar overhead

Michael and Scott’s - baseline
Durable (durable linearizable)
Log (detectable)
Relaxed - frequent “sync”
Relaxed - between in/frequent
Relaxed - infrequent “sync”
CONCLUSION

- A new definition: detectable execution
- Three lock-free queues for NVM: Relaxed, Durable, Log
- Guidelines
- Evaluation
  - Durability and detectability - similar overhead
  - Buffered durability is less costly