LINEARIZABILITY TESTING OF MULTI-WORD SYNCHRONIZATION PRIMITIVES

Presenter: Sakib Chowdhury
Joint work with: Diego Cepeda, Wojciech Golab, Nan Li, Raphael Lopez, Jeff Wang
August 2, 2019
OUTLINE

• Motivation
• Background
• The Algorithm
• Demonstration
• Conclusion
MOTIVATION

• Persistent memory will improve performance and reliability of data access
• We want to be sure that our behaviour is correct
MOTIVATION: CODE COMPLEXITY

- CASN
  - Harris, 2002
  - ~40 lines of code

- PMwCAS
  - Larson, Levandoski, Wang, 2017
  - ~80 lines of code

"Mo’ [memory]… mo’ problems.”
— Notorious B.I.G., 1997
MOTIVATION: MWCAS AND PERSISTENT MEMORY

- simultaneous updates required
- new node insertion
MOTIVATION: VERIFYING CORRECTNESS

• Model checking: exhaustive search and test of all possible states of the system
  » Intractable due to state-space explosion as number of processes increases
MOTIVATION: VERIFYING CORRECTNESS

• Blackbox testing: because we use software for what it does
  » Can identify mistakes in behaviour of a system
  » Cannot prove that all behaviours are correct
  » Absence of evidence vs. evidence of absence
BACKGROUND: LINEARIZABILITY

thread 1

W(1)  W(2)  R(2)

thread 2

R(2)

linearizable

thread 1

W(1)  W(2)  R(1)

thread 2

R(2)

not linearizable
**BACKGROUND: MULTI-WORD OPERATIONS**

```c
mwread(addresses) {
    values[n]
    atomic {
        for i in 1...n {
            values[i] = read(addresses[i])
        }
    }
    return values
}

mwcas(addresses, expected-values, desired-values) {
    atomic {
        for i in 1...n {
            if read(addresses[i]) != expected-values[i] {
                return false
            }
        }
        for i in 1...n {
            write(addresses[i], desired-values[i])
        }
    }
    return true
}
```
• Crash failure: when operations fail to return a response due to failure of their threads
BACKGROUND: RECOVERY

thread 1

W(1)  W(2)

thread 2

R(1)  R(2)

operation took effect

thread 1

W(1)  W(2)

thread 2

R(1)  R(1)

operation did not take effect (before read)
• Behaviour of a system is recorded as an execution history
BACKGROUND: EXAMPLE MULTI-WORD HISTORY

1 I t1 A B 1 2
2 I t2 C D 3 4
3 I t3 C D 5 6
4 R t3 C D 0 0 0
5 R t2 C D 5 6 0
6 R t1 A B 0 0 0

• 0 is success, 1 ≤ i ≤ n is failure due to comparison i
• R is for reads
BACKGROUND: LINEARIZING AFTER FAILURE

• No response? No problem!
• Let’s add one if it looks like the operation took effect
• But where?
• Recoverable linearizability
  » The operation must have occurred before the next operation on the same object by the same thread
BACKGROUND: ANALYSIS

• Linearizability testing of arbitrary histories involving reads, writes is \textbf{NP-complete}.

• How can we do blackbox testing in polynomial time?
BACKGROUND: ANALYSIS

• **Solution:** prove P=NP
• **Check:** is it easier with swaps?
  » given the previous value in the register during a write (i.e. a swap), the problem is still NP-complete
  » *Gibbons, Korach. 1997.*
• **Solution:** Given the read-mapping, the problem is $O(n \log n)$ for a history of $n$ operations
  » *Gibbons, Korach. 1997.*
  » We can infer the read-mapping from the log of successful swaps if all values are unique
BACKGROUND: ANALYSIS TECHNIQUES

• Zone-based testing
  » Proposed by Gibbons, Korach
  » Defines interval of time, or “zone”, over which a value is the latest value of an object

• Graph-based testing
  » Build precedence graph of operations and check for cycles and consistency
BACKGROUND: ANALYSIS TECHNIQUES

• Hitting families
  » Order small groups of operations linearizably first
    • Ozkan, Majumdar, Niksic. 2019.

• Data-structure-specific methods
  » P-time algorithms exist for abstract collections
    • Emmi, Enea. 2018.
  » Can reduce some models to simpler models
    • Bouajjani et al. 2015.
  » Local view arguments to easily linearize search operations
    • Feldman et al. 2018.
THE ALGORITHM: THE GRAPH

- Vertices represent each operation
• Directed edges establish order in which they must be linearized
THE ALGORITHM: EDGE BUILDING

• Reads-from edge: read of a value must occur after it is written

(read: 0, write: 5) → (read: 5, write: 6) → (read: 6, write: 7)
THE ALGORITHM: EDGE BUILDING

• Time-precedence edge: non-concurrent operations are totally ordered
THE ALGORITHM: EDGE BUILDING

- Greatly reduce the number of time edges by using an algorithm to select necessary edges
THE ALGORITHM: UNCONDITIONAL MULTI-WORD SWAPS
THE ALGORITHM: MULTI-WORD READS
THE ALGORITHM: UNSUCCESSFUL CAS OPERATIONS
THE ALGORITHM: TYPES OF LINEARIZABILITY ERRORS

- Fork
THE ALGORITHM: TYPES OF LINEARIZABILITY ERRORS

- Time travel
THE ALGORITHM: TYPES OF LINEARIZABILITY ERRORS

- Disconnect
THE ALGORITHM: PERFORMANCE

![Graph showing runtime vs. log size (MB)]
CONCLUSION

• Blackbox testing is a valuable tool when model checking becomes increasingly difficult

• Graph-based methods allow intuitive extension of linearizability testing to multiple words and new operations

• Testing recoverability of multi-word primitives is important because of their usefulness in building linked data structures

Questions?