Monitoring Weak Consistency

Michael Emmi
SRI International

Constantin Enea
Université Paris Diderot

Motivation

Concurrent Objects

High-level abstractions

e.g. numeric & collection ADTs

High performance

e.g. lock-free shared-memory access

Available on modern platforms

e.g. dozens in JDK

```
/**
 * a concurrent collection is thread-
 * safe, but not governed by a single
 * exclusion lock.
 */
package java.util.concurrent;

// we've considered these objects
class ConcurrentHashMap { … }
class ConcurrentSkipListMap { … }
class ConcurrentSkipListSet { … }
class ConcurrentLinkedQueue { … }
class LinkedTransferQueue { … }
class LinkedBlockingQueue { … }
class ConcurrentLinkedDeque { … }

// and there are several more
```

Weak Consistency

Performance optimization

synchronization vs. guarantees

Out in the wild

e.g. collections in JDK

Relaxed *visibility* framework

Burckhardt et al, POPL 2014

```
package java.util.concurrent;
class ConcurrentSkipListSet { … }

/**
 * Iterators and spliterators are
* weakly consistent...
*
* They are guaranteed to traverse
* elements as they existed upon
* construction exactly once and may
* (but are not guaranteed to) reflect
* any modification subsequent to
* construction.
*/
```

E.g. the Size Method

History

partial order among invocations

Linearization

total order of history

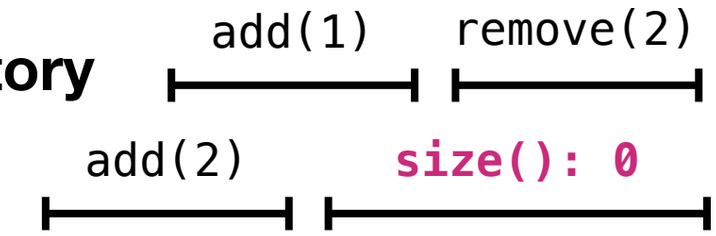
Visibility

subsequence of linearized-before including happens-before

Consistency

visible sequence admitted by ADT

History



Linearization

`add(1); add(2); remove(2); size(): 0`

visible to size			inc HB	ADT size
add(1)	add(2)	rem(2)		
√	√	√	√	1
√	√		√	2
√		√		1
	√	√	√	0
√				1
	√		√	1
		√		0
				0

Weak Consistency Validation

Contributions

Effective algorithm

consider only ***minimal*** visibilities

Theoretical justification

minimals-only is sound

Empirical evaluation

drastic reduction in validation time

Algorithm

Given history

trace of call and return actions

Incremental search

one invocation at a time

Enumerate linearizations

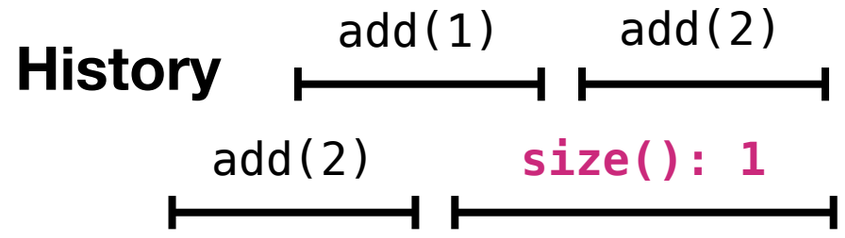
consistent with history order

Enumerate visibilities

consistent with ADT

```
class Validator {
    h: History

    validate(lin, vis): boolean {

        // base case
        if (lin.isComplete(this.h))
            return true;

        // branch enumeration
        for (let l2 of lin.extend(this.h))
            for (let v2 of vis.extend(l2))
                if (this.validate(l2, v2))
                    return true;

        // backtracking
        return false;
    }
}
interface Linearization {
    isComplete(h: History): boolean;
    extend(h: History): this[];
}
interface Visibility {
    extend(lin: Linearization): this[];
}
```

Minimal Visibilities



Consistent

visible sequence admitted by ADT including happens-before

Linearization

`add(1); add(2); add(2); size(): 1`

Minimal

no consistent subsequence

Theorem

sound to enumerate only minimal consistent visibilities

visible to size			inc HB	ADT size	min
add(1)	add(2)	add(2)			
√	√	√	√	2	
√	√		√	2	
√		√		2	
	√	√	√	1	
√				1	
	√		√	1	√
		√		1	
				0	

Experiments

Setup

Java concurrent objects

reported on ConcurrentSkipListMap

Random test generation

up to 15 invocations across 3 threads

History recording

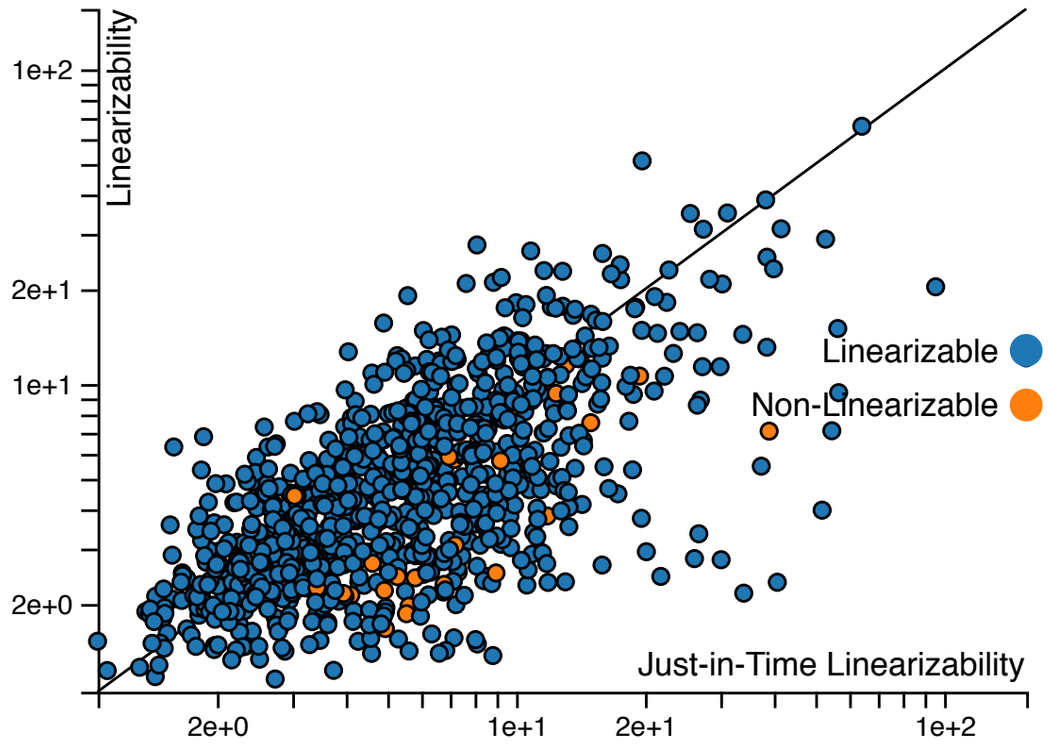
4K sampled during stress testing

Offline, incremental validation

linearize one invocation at a time

Basic consistency

all HB-predecessors are visible



Baseline

Naïve enumeration

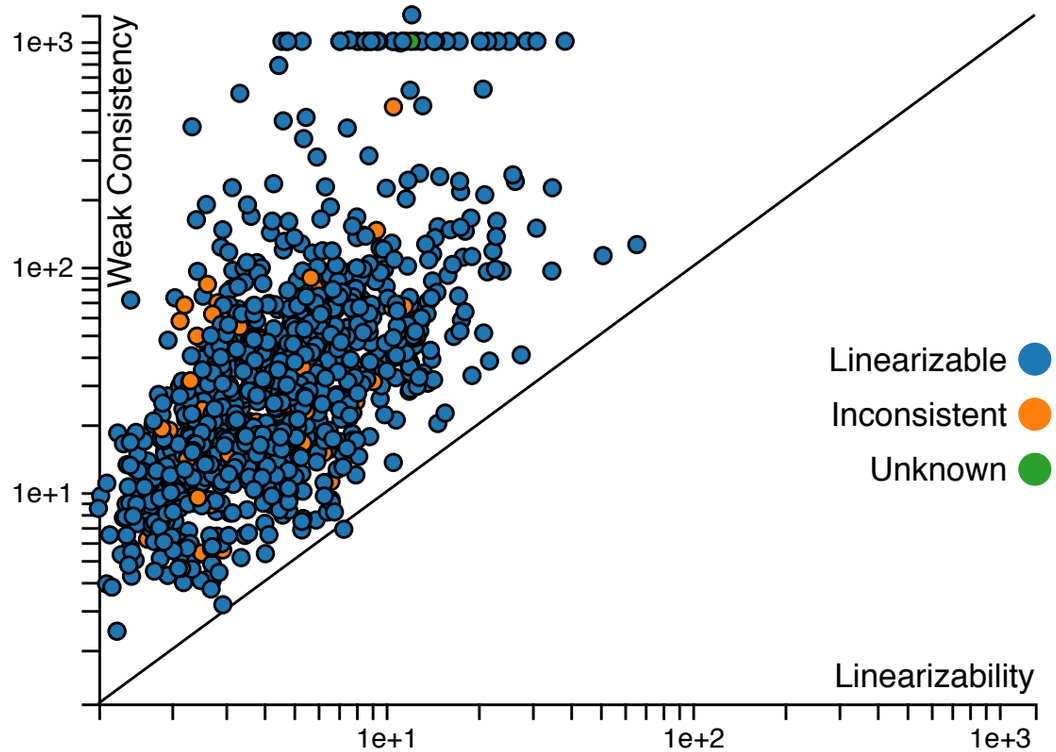
vs. linearizability

10x overhead

due to visibility enumeration

Up to 3 OOM worse

3 ms → 1000 ms (timeout)



Optimization

Minimals-only enumeration

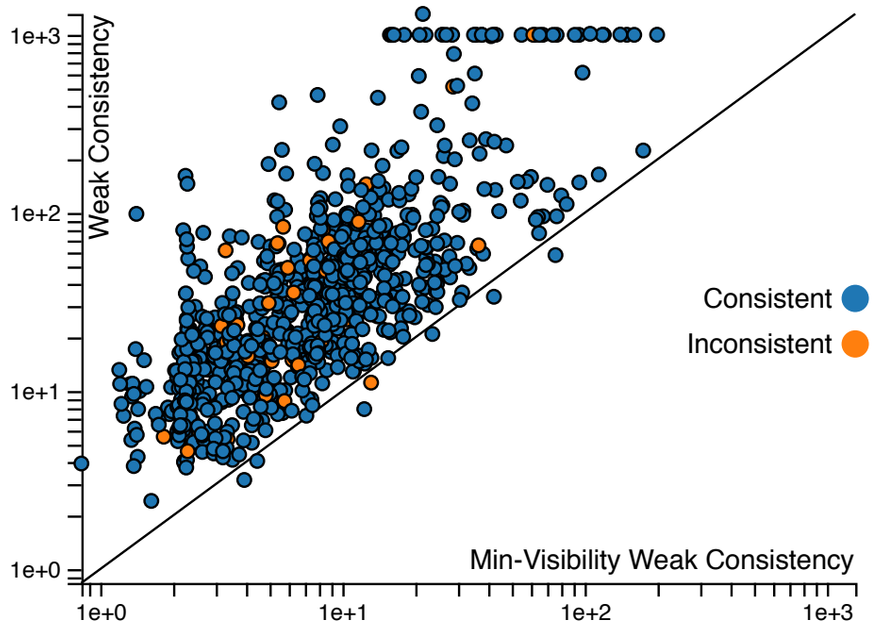
vs. naive enumeration

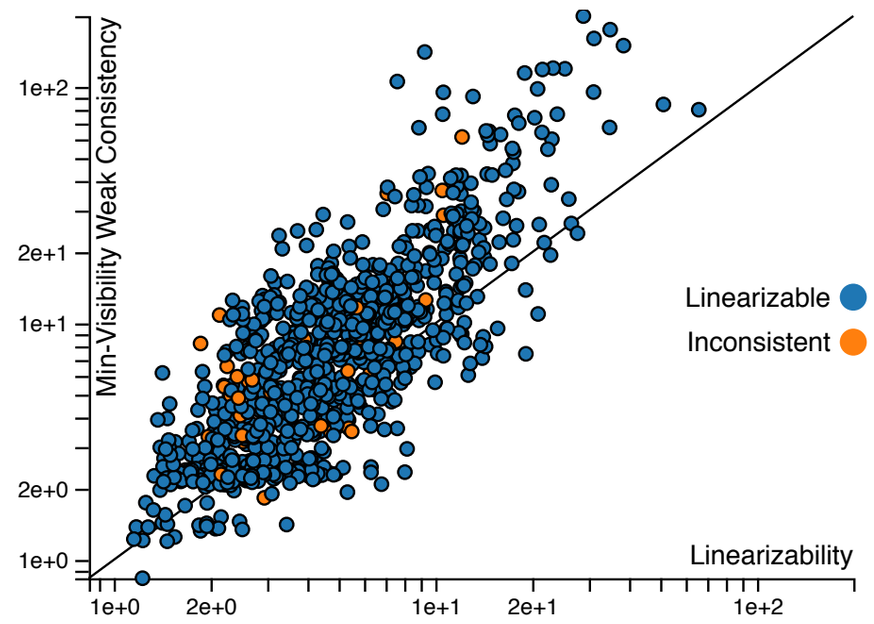
Up to 2 OOM improvement

1000 ms (timeout) → 18 ms

2x overhead

compared with 10x for naïve enumeration





Related Work

Linearization enumeration

Wing & Gong: *Testing and Verifying Concurrent Objects*. JPDC 1993

Without specifications

Burckhardt et al: *Line-Up: a Complete and Automatic Linearizability Checker.* PLDI 2010

With commutativity specifications

Shacham et al: *Testing Atomicity of Composed Concurrent Operations*. OOPSLA 2011

Symbolic enumeration

Emmi, Enea, Hamza: *Monitoring Refinement via Symbolic Reasoning.* PLDI 2015

Just-in-time enumeration

Lowe: *Testing for Linearizability*. C&C 2017