Mindicator
A Nonblocking Multiset Optimized for Querying the Minimum Value

Yujie Liu and Michael Spear
Presenter: Yujie Liu
Problem: How old is the youngest?

Room

<table>
<thead>
<tr>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>18</td>
</tr>
</tbody>
</table>

Age of Youngest: 4
Problem: How old is the youngest?

Room

Arrive()

Depart()

Query()

Age of Youngest: 4

32 15 11
9 4 7
20 5 18

Youngest: 4
Problem: How old is the youngest?
Why are we interested in knowing the minimum..
Usage

• Garbage Collection
  – Epoch-based memory reclamation

• Software Transactional Memory
  – Privatization
  – Progress Guarantee

• Operating System Kernels
  – RCU synchronization
Use Case: Memory Reclamation

Epoch: a monotonically increasing timestamp
Use Case: Memory Reclamation

- Epoch-based memory reclamation
  - Before operation $T$ reclaims memory location $X$, $T$ must ensure no other operation currently holds reference to $X$.
  - **INV**: Only operations older than $T$ can possibly hold reference to the location $X$.
  - **INV**: Operations newer than $T$ cannot hold reference to the location $X$. 
Use Case: Memory Reclamation

Need to ensure no one else is reading the location I’m going to reclaim.

Arrive() when operation begins

Depart() when operation ends

Share Data Structure

Epoch: a monotonically increasing timestamp
Use Case: Memory Reclamation

**Epoch**: a monotonically increasing timestamp

```
1 3 7
9 12 1
20 25 28
```

**Shared Data Structure**

**Arrive()**
when operation begins

**Query()**
loops until (minimum >= 12)

**Depart()**
when operation ends

Need to wait until I am the minimum...
Use Case: Memory Reclamation

Epoch: a monotonically increasing timestamp
Simple solutions do exist..
Simple Solution 1

- Ordered Linked List
  - Query is reading the list head, $O(1)$
  - Arrive: insert my value, $O(n)$
  - Depart: remove my value, $O(n)$, $O(1)$ for doubly linked list
Simple Solution 2

- Array of Per-thread Slots
  - Query is at least $O(n)$
  - Arrive and Depart: $O(1)$

```plaintext
Query()
scan array
```

<table>
<thead>
<tr>
<th>2</th>
<th>3</th>
<th>6</th>
<th>8</th>
<th>9</th>
<th>15</th>
<th>22</th>
<th>37</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Penguins" /></td>
<td><img src="image" alt="Penguins" /></td>
<td><img src="image" alt="Penguins" /></td>
<td><img src="image" alt="Penguins" /></td>
<td><img src="image" alt="Penguins" /></td>
<td><img src="image" alt="Penguins" /></td>
<td><img src="image" alt="Penguins" /></td>
<td><img src="image" alt="Penguins" /></td>
</tr>
</tbody>
</table>
Desired Properties of Mindicator

- **Linearizable**
  - Operations happen *instantly* at some time point between invocation and response.

- **Nonblocking**
  - Arrive/Depart does not block each other at *any* step on-the-fly.
  - Lock-free: *someone* makes progress in finite number of steps.

- **Fast & Scalable** Query
  - O(1) time complexity
  - Cache friendly
Simple Solutions (cons)

• Linked List
  – Insert & Remove is linear to #thread
  – Need expensive memory alloc/free, or GC

• Doubly linked List
  – Insert is linear to #thread
  – Needs Locking

• Slot Array
  – Query is linear to #threads
  – Cache unfriendly
  – Non-general if no locking in Query
Other Data Structures

• Lock-free Skip List
  – Support log(n) insert/remove
  – Need garbage collection
    • How can we use it in garbage collectors?
  – Expensive node alloc/free
Literature we found..
Previous Work: **f-Array** by Jayanti [PODC’02]

- General solution for computing function $f$
  - Tree-based structure
  - Wait-free
  - But not scalable
    - Query reads the root
    - Every Arrive/Depart writes the root
Previous Work: **SNZI** by Ellen et al [PODC’07]

- Solve a weaker problem
  - Query answers whether the room is empty or not
  - Tree-based structure
    - Parent is 1 iff. *some* child is 1
- Scalable
  - Query reads the root
  - Arrive & Depart affect small fraction of the tree, not including the root in the common cases
A simple Mindicator implementation inspired by SNZI + f-Array
Simple Mindicator

Tree-based Structure
- Each thread assigned with a dedicated leaf

Initially
Every node has value \( T \) (maximum possible value)
Simple Mindicator

Tree-based Structure
- Each thread assigned with a dedicated leaf
- Threads propagate changes from leaf to root
- Parent stores the minimum of children
- Query reads the root

Initially
Every node has value $T$ (maximum possible value)
Building Blocks

• Load-linked/Store-conditional (**LL/SC**)
  – Existing hardware support
  – Or simply implemented via CAS
Building Blocks

- Load-linked/Store-conditional (LL/SC)
  - Existing hardware support
  - Or simply implemented via CAS
Simple Mindicator: Arrive()

Arrive of p:
1.  \( P = T, P \leftarrow 2 \)
Simple Mindicator: Arrive()

Arrive of p:
1. $P = T, P \leftarrow 2$
2. $X = T, X \leftarrow 2$
Simple Mindicator: Arrive()

Arrive of p:
1. $P = T$, $P \leftarrow 2$
2. $X = T$, $X \leftarrow 2$
3. $R = T$, $R \leftarrow 2$
* Finish *
Simple Mindicator: Arrive()

Arrive of q:
1. $Q = T, Q \leftarrow 7$

![Diagram of a simple mindicator with a tree structure and nodes labeled with numbers and symbols.]
Simple Mindicator: Arrive()

Arrive of q:
1. $Q = T, Q \leftarrow 7$
2. $X = 2 < 7$
* Finish *
Simple Mindicator: Depart()
Simple Mindicator: Depart()

Depart of q:
1. $Q = 7$, $Q \leftarrow T$
Simple Mindicator: Depart()

Depart of q:
1. \( Q = 7, Q \leftarrow T \)
2. \( X = 2 < 7 \)
* Finish *
How things can go wrong..
A Problematic Interleaving

Arrive of p: 

Arrive of q: 

Diagram:

- T_R
- T_X
- T_P
- T_Q
A Problematic Interleaving

Arrive of p:
P = T, P ← 2

Arrive of q:

2
P
T
X
R

2
P
T
Q
A Problematic Interleaving

Arrive of p:
\[ P = T, P \leftarrow 2 \]

Arrive of q:
\[ Q = T, Q \leftarrow 7 \]
A Problematic Interleaving

Arrive of $p$:
$P = T, P \leftarrow 2$
$X = T, P \leftarrow 2$

Arrive of $q$:
$Q = T, Q \leftarrow 7$
A Problematic Interleaving

Arrive of $p$: $P = T, P \leftarrow 2$

Arrive of $q$: $Q = T, Q \leftarrow 7$

$X = 2 < 7$

* Finish *
A Problematic Interleaving

Arrive of p:
P = T, P ← 2
X = T, P ← 2

Arrive of q:
Q = T, Q ← 7
X = 2 < 7
* Finish *
...

Query of q:
returns R = T
A Problematic Interleaving

Arrive of p: 
P = T, P ← 2
X = T, P ← 2

Arrive of q: 
Q = T, Q ← 7
X = 2 < 7
* Finish *
...

Query of q: 
returns R = T
* Oops! *

I just Arrived..
Where is my value???
Well let’s go one step back..
A Problematic Interleaving

Arrive of $p$: $P = T, P \leftarrow 2$
$X = T, P \leftarrow 2$

Arrive of $q$: $Q = T, Q \leftarrow 7$
$X = 2 < 7$

* ??? Finish ??? *

Can I trust the value 2 at $X$?
More information is needed to distinguish \textit{dependable} values.
A Problematic Interleaving

Arrive of p:
\[ P = T, P \leftarrow 2 \]
\[ X = T, P \leftarrow 2 \]

Arrive of q:
\[ Q = T, Q \leftarrow 7 \]
\[ X = 2 < 7 \]

* ??? Finish ??? *

Can I trust the value 2 at X?
More information is needed to distinguish dependable values.

- One extra bit (per node) suffices.
Adding the **clean bit**

- Values are initially clean
- Values marked as unclean if being propagated up
- Invariants
  - If X is clean, for every ancestor A of X, A.val <= X.val
  - If X is clean, for every ancestor A of X, if A is clean, then A.val < X.val
2-Staged Arrive

• Propagate Stage
  – Start from my leaf propagating my value
  – Value written is marked as “unclean” until operation reaches a “turning point”

• Clean Stage
  – Back from the turning point to my leaf
  – Reset the visited nodes to clean
A Problematic Interleaving (Fixed)

Arrive of p: 

Arrive of q:
A Problematic Interleaving (Fixed)

Arrive of \( p \):
\[
P = T, P \leftarrow 2'
\]

Arrive of \( q \):
\[
50
\]
A Problematic Interleaving (Fixed)

Arrive of p:
\[ P = T, \ P \leftarrow 2' \]

Arrive of q:
\[ Q = T, \ Q \leftarrow 7' \]
A Problematic Interleaving (Fixed)

Arrive of p: 
\[ P = T, P \leftarrow 2' \]
\[ X = T, P \leftarrow 2' \]

Arrive of q: 
\[ Q = T, Q \leftarrow 7' \]
A Problematic Interleaving (Fixed)

Arrive of p:
- $P = T, P \leftarrow 2'$
- $X = T, P \leftarrow 2'$

Arrive of q:
- $Q = T, Q \leftarrow 7'$
- $X = 2' < 7$
  - * $X$ is not clean, cannot finish *
A Problematic Interleaving (Fixed)

Arrive of $p$: $P = T, P \leftarrow 2'$
$X = T, P \leftarrow 2'$

Arrive of $q$: $Q = T, Q \leftarrow 7'$
$X = 2' < 7$
* $X$ is not clean, cannot finish *
$R = T, R \leftarrow 7'$
A Problematic Interleaving (Fixed)

Arrive of p:
P = T, P ← 2'
X = T, P ← 2'

Arrive of q:
Q = T, Q ← 7'
X = 2' < 7
* X is not clean, cannot finish *
R = T, R ← 7'
R = 7', R ← 7
A Problematic Interleaving (Fixed)

Arrive of p:
P = T, P ← 2’
X = T, P ← 2’

Arrive of q:
Q = T, Q ← 7’
X = 2’ < 7
* X is not clean, cannot finish *
R = T, R ← 7’
R = 7’, R ← 7
X = 2’, Skip
A Problematic Interleaving (Fixed)

Arrive of p:
P = T, P ← 2'
X = T, P ← 2'

Arrive of q:
Q = T, Q ← 7'
X = 2' < 7
* X is not clean, cannot finish *
R = T, R ← 7'
R = 7', R ← 7
X = 2', Skip
Q = 7', Q ← 7
Another possible interleaving..
Another Possible Interleaving

Arrive of p: T

Arrive of q: T

2

7
Another Possible Interleaving

Arrive of $p$: $P = T, P \leftarrow 2'$

Arrive of $q$: $\text{ARRIVE OF Q}$
Another Possible Interleaving

Arrive of p: $P = T, P \leftarrow 2'$

Arrive of q: $Q = T, Q \leftarrow 7'$
Another Possible Interleaving

Arrive of p:
P = T, P ← 2
X = T, P ← 2'

Arrive of q:
Q = T, Q ← 7'
Another Possible Interleaving

Arrive of \( p \):
\[
P = T, \ P \leftarrow 2'
\]
\[
X = T, \ P \leftarrow 2'
\]
\[
R = T, \ R \leftarrow 2'
\]

Arrive of \( q \):
\[
Q = T, \ Q \leftarrow 7'
\]
Another Possible Interleaving

Arrive of $p$:
$P = T$, $P \leftarrow 2'$
$X = T$, $P \leftarrow 2'$
$R = T$, $R \leftarrow 2'$
$R = 2'$, $R \leftarrow 2$

Arrive of $q$:
$Q = T$, $Q \leftarrow 7'$
Another Possible Interleaving

Arrive of $p$:
- $P = T$, $P \leftarrow 2'$
- $X = T$, $P \leftarrow 2'$
- $R = T$, $R \leftarrow 2'$
- $R = 2'$, $R \leftarrow 2$
- $X = 2'$, $X \leftarrow 2$

Arrive of $q$:
- $Q = T$, $Q \leftarrow 7'$
Another Possible Interleaving

Arrive of $p$: $P = T, P \leftarrow 2, X = T, P \leftarrow 2, R = T, R \leftarrow 2, R = 2, R \leftarrow 2, X \leftarrow 2$

$x$ is my turning point!

Arrive of $q$: $Q = T, Q \leftarrow 7$

$R = 2, R \leftarrow 2$

$X = 2, X \leftarrow 2$

$X = 2 < 7, X = 2$

* Turns *
Another Possible Interleaving

Arrive of p:
- \( P = T, P \leftarrow 2' \)
- \( X = T, P \leftarrow 2' \)
- \( R = T, R \leftarrow 2' \)
- \( R = 2', R \leftarrow 2 \)
- \( X = 2', X \leftarrow 2 \)

Arrive of q:
- \( Q = T, Q \leftarrow 7' \)
- \( X = 2 < 7, X = 2 \)
  * Turns *
- \( Q = 7', Q \leftarrow 7' \)
  * Finish *
Arrive Operation: Details
Arrive Case 1

Case 1: My value is smaller than the node’s value.
Case 1: My value is smaller than the node’s value.  
-- Decrease it to my value and goes to parent.
Case 2: My value $\geq$ node’s value and the node is not clean.
Arrive Case 2

Case 2: My value >= node’s value and the node is **not** clean.
-- Simply go to the parent.
Case 3: My value >= node’s value and the node is clean.
Case 3: My value $\geq$ node’s value and the node is clean.

Meet the turning point!

Every ancestors’ value $\leq 2$
(Recall the invariant)
Case 3: My value >= node’s value and the node is clean.
-- We can use the clean-node-invariant.
Depart Sketch

• At node X:
  1. \( x \leftarrow \text{LL}(X) \)
  2. If \( x \) is not clean, we are done at \( X \).
     • Go to parent
  3. Otherwise
     • READ each child of \( X \) and compute the minimum
     • \( \text{SC}(X, < \text{min}, \text{(min}=x.\text{val})>) \)
     • If it succeeds, we are done at \( X \)
       – Decide whether we need to traverse to parent, or return early
Depart Sketch

• At node X:
  1. $x \leftarrow \text{LL}(X)$
  2. If $x$ is not clean, we are done at X.
     • Go to parent
  3. Otherwise
     • READ each child of X and compute the minimum
     • $\text{SC}(X, \langle \text{min}, (\text{min}\geq x.\text{val}) \rangle)$
     • If it succeeds, we are done at X
       – Decide whether we need to traverse to parent, or return early

Subtlety 1:
What if $x.\text{value} = \text{my value}$? Why isn’t it my value?
Depart Sketch

• At node X:
  1.  \( x \leftarrow \text{LL}(X) \)
  2.  If \( x \) is not clean, we are done at \( X \)
      • Go to parent
  3.  Otherwise
      • READ each child of \( X \) and compute the minimum
      • \( \text{SC}(X, < \text{min}, (\text{min}>=\text{x.val})>) \)
      • If it succeeds, we are done at \( X \)
          – Decide whether we need to traverse to parent, or return early

Subtlety 2:
Why not simply copy the clean bit of the minimum child to \( X \)?
To answer these subtleties..
The shortest, complete answer we have:

- A 15 pages, rigorous but informal proof
- Using Gries-Owicki invariance style
- Proving Linearizability
  - Define LP points
  - Implementation refines the abstract specification
Evaluation

• List-based implementation
  – Default used in prior work
• Lock-free implementation
• Quiescently consistent implementation
  – Relaxes semantics of Depart
• Stress-tested on microbenchmark
  – All threads repeatedly Arrive and Depart
• A privatization-safe STM using Mindicator
  – RBTree benchmark
Performance (Niagara2 Microbenchmark)
Performance (x86 Microbenchmark)

The graph shows the throughput (1000 Visits/sec) for different thread counts for linearizable, quiescent, and linked list operations.

- Linearizable operation has a sharp drop at 3 threads and stabilizes after that.
- Quiescent operation shows a steady increase.
- LinkedList operation also shows an increase but at a lower rate.

The x-axis represents the number of threads, and the y-axis represents throughput.
Performance (Niagara2 STM)
Conclusion & Future Work

• We are working with a group from IBM to see if this can be integrated into Linux/RCU

• We are still finding new uses for it in STM and elsewhere