Citron: Distributed Range Lock Management with One-sided RDMA

Jian Gao, Youyou Lu, Minhui Xie, Qing Wang, Jiwu Shu

Tsinghua University

FAST’23
Range Locks

Concurrency safety for accessing storage address space
Previous Approaches

- **CPU-based** range lock manager with **two-sided RDMA-based** RPC
- Rely fully on server-side CPUs -> CPU bottleneck
One-sided Approach?

- Range lock = **mutex (already solved!)** * range size
- Excessive network roundtrips -> high latencies

---

One-sided Approach? (Cont.)

- Combine multiple mutexes into a larger one
- **Resource waste for small ranges leads to low throughput**

Our One-sided Solution: 

effi\textit{C}lent dis\textit{T}ributed \textit{R}ange \textit{I}Ock \textit{maN}ager
The Big Picture

**Data Structure**
- Segment Tree
- Ranges to mutexes

**Lock Protocol**
- Extended atomics
- Resolve conflicts

**Citron**
The Segment Tree

A perfect tree whose each node represents a range.

Any range $\leftrightarrow O(\log N)$ tree nodes

Example: $degree = 2$
Map Ranges to Tree Nodes

**Throughput-Optimal**
+ Precise mapping
- **High latencies:** \( O(\log N) \) nodes, tens in the worst case

**Latency-Optimal**
+ Lock only one node
- **Low throughput:** false conflicts

---

Citron’s trade-off

Lock up to \( k \) nodes. \( k = 2 \) by default.
Selecting the Nodes Properly

**Goal**: minimize locked but unneeded range.

- **Method**: Knapsack algorithm
- Should I split here?
  - **Weight**: # children nodes to lock – 1
  - **Value**: preciseness improvement

\[ k = 2 \text{ reduces false conflicts by 96\% than } k = 1 \]
Lock Protocol - Theory

- **Observation**: Node X conflict only with ancestors and descendants.

- **Diagram**:
  - Desired node
  - Ancestors
  - Descendants
  - Notification + Detection

- **Text**: I want to lock this node!
Lock Protocol - Practice, i.e., Tasks

• **Observation:** Node X conflict only with ancestors and descendants
Lock Protocol - Tools

RDMA NICs nowadays support Extended Atomics.

**Ext-CAS**

Manipulate **leaf nodes**

*Set bit = locked, unset bit = unlocked*

**Ext-FAA**

Manipulate **internal nodes**

*Idea from Lamport’s bakery*
Task 1 - Lock the Desired Node

• **Leaf:** Ext-CAS -- lock desired bits

• **Internal:**
  • Enter bakery
    
  ```
  mypos = rear++;
  ```
  • Then wait for my turn
    
  ```
  while (head != mypos);
  ```

Blue fields are members of internal nodes.
Task 2 - Notify Descendants

- A 1-bit flag called **occupied**
  - Blocks subsequent conflicts @ descendants
  - \texttt{occupied = 1;}

ExtFAA
Set the \texttt{occupied} flag
Another client’s Task 2:

- Read the *occupied* flag
- Wait for occupied ancestors
  - *while* (ancestor.*occupied*);
Task 4 - Notify Ancestors

• Another pair of counters
  • \(d_{\text{head}}, d_{\text{rear}}\)

• Enter the “descendant queue” of each ancestor
  • \(\text{ancestor}.d_{\text{rear}}++;\)
Task 5 - Detect Descendants

Another client’s Task 4:

- Wait for my “descendant queue” to get emptied
  - while (d_head != d_rear);

Diagram:
- Counterpart
- ExtFAA
  - Increment d_rear
- Read
  - Poll d_head & d_rear
- Desired node
- Ancestors
- Descendants
Data Structure Summary

Segment Tree

Internal node: $\textit{Fields(bit-widths)}$

- occupied(1)
- head(15)
- rear(15)
- d_head(16)
- d_rear(16)

Leaf node: $\textit{Bitmap}$

```
1 1 0 0 1 1 1 1 ... 1 0 1
```
# Lock Protocol Summary

<table>
<thead>
<tr>
<th>Task</th>
<th>RDMA Verbs</th>
<th>Verbs’ Targets</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Lock the desired node</td>
<td>ExtCAS / ExtFAA + Read</td>
<td>Desired node</td>
</tr>
<tr>
<td>2. Notify descendants</td>
<td>ExtFAA</td>
<td>Desired node</td>
</tr>
<tr>
<td>3. Detect conflicts at ancestors</td>
<td>Read</td>
<td>Ancestors</td>
</tr>
<tr>
<td>4. Notify ancestors</td>
<td>ExtFAA</td>
<td>Ancestors</td>
</tr>
<tr>
<td>5. Detect conflicts at descendants</td>
<td>Read</td>
<td>Desired node</td>
</tr>
</tbody>
</table>
Unlock

Revert modifications done in lock acquisition.

- **ExtFAA**
  - Increment \(d_{\text{head}}\)

- **ExtCAS / ExtFAA**
  - Clear the bitmap / Increment \(\text{head} \& \text{clear occupied}\)

Diagram:
- Ancestors
- Desired node
- Descendants
Supportive Designs

- **Timing-based Sync (Task 4 & 5)** to enforce protocol correctness

- **Strided Notification (Task 4)** to reduce overheads

  - Runtime scaling, parameter tuning, fast path, failure recovery, ...
  - *See our paper for more details!*
Experiment Setup

- 3 clients + 1 server
- **Microbenchmark**: lock acquire & release
- **Application**: Filebench & IO500 & NPB BT-IO

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel Xeon Gold 5220 @ 2.20 GHz</td>
</tr>
<tr>
<td>Memory</td>
<td>256 GB</td>
</tr>
<tr>
<td>NIC</td>
<td>Mellanox ConnectX-6</td>
</tr>
</tbody>
</table>
## Baselines

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two-sided RDMA with eRPC [2]</td>
<td>Maple tree from Oracle Linux UEK</td>
</tr>
<tr>
<td></td>
<td>Interval tree from Lustre</td>
</tr>
<tr>
<td></td>
<td><strong>Linked list</strong> from [1]</td>
</tr>
<tr>
<td>One-sided RDMA</td>
<td><strong>Linked list</strong> from [1] with pure RDMA</td>
</tr>
<tr>
<td></td>
<td><strong>Trivially mutex * range size</strong></td>
</tr>
</tbody>
</table>

Citron vs. Two-sided

<table>
<thead>
<tr>
<th>Size</th>
<th>P99 Lat (us)</th>
<th>Thpt (Mops/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>256</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- P99 Lat (us) for different sizes:
  - Size = 1: 3.9x decrease
  - Size = 16: 9.0x decrease
  - Size = 256: 1.4x increase

- Thpt (Mops/s) for different sizes:
  - Size = 1: 1.7x decrease
  - Size = 16: 2.4x increase
  - Size = 256: 0.97x increase

Citron vs. Two-sided for different number of clients.
Citron vs. Two-sided

Citron has overall higher throughputs and lower latencies than CPU-centric baselines.
Citron vs. Trivial One-sided

- **P99 Lat (us)**
  - Size = 1
  - Size = 16
  - Size = 256
  - Size increases from 1 to 256.

- **Thpt (Mops/s)**
  - Size = 1
  - Size = 16
  - Size = 256
  - Size increases from 1 to 256.

Number of Clients:
- Citron
- One-sided

- Size = 1
  - Citron: 1.9x
  - One-sided: 1.9x

- Size = 16
  - Citron: 5.7x
  - One-sided: 5.7x

- Size = 256
  - Citron: 1.02x
  - One-sided: 1.3x
Citron vs. Trivial One-sided

Citron is suboptimal under mutex-only workloads.

With unaligned ranges Citron performs better.

Note: aligned = mutex-only
Other Evaluation Results

• P50 latencies *similar*
• P99 latencies *orders-of-magnitude lower*

• Limited *false conflict rates* and *abort rates* (1e-5 ~ 1e-2)

• Quickly adapt to storage resource size growths (*sub-millisecond* level)
Conclusion

• We designed Citron, an efficient *distributed range lock manager* using *only one-sided RDMA* to acquire and release locks.

• Citron translates ranges into RDMA-friendly mutexes with a *segment tree* and use *RDMA Extended Atomics* to perform synchronization.

• Citron achieves overall higher performance than both two-sided and trivial one-sided baselines.
Thanks & Q/A

Citron: Distributed Range Lock Management with One-sided RDMA

Jian Gao, Youyou Lu, Minhui Xie, Qing Wang, Jiwu Shu

Contact: gaoj20@mails.tsinghua.edu.cn