AN RCU WITH LOW SYNC LATENCY

PRCU

张恒
ZHANG HENG
WHAT IS RCU

- Read-Copy Update
- A synchronization primitive which allows readers and writers execute concurrently
- Great for read-mostly data
- Two phase:
  - update on a new copy
  - reclaim the old copy (when it is safe)
LIST WITH RWLOCK
RWLOCK OR RCU
SAME PROBLEM

- Reader-Writer Locks
  - when writers can enter CS
- RCU
  - when updaters can reclaim the resource

ENSURE ALL PREVIOUS READERS HAVE LEFT
RWLOCK OR RCU
DIFFERENT ATTITUDE

- Reader-Writer Locks
  - different algorithms for different workloads, e.g. rwsem, brlock …

- RCU
  - extremely low overhead on read side (fastpath)
  - extremely high overhead on write side
## RWLOCK OR RCU

### TRADEOFF

<table>
<thead>
<tr>
<th>RWSEM</th>
<th>HEAVY CONTENTION</th>
<th>LIGHTWEIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSNZI</td>
<td>LIGHT CONTENTION</td>
<td>MORE ATOMIC OPS</td>
</tr>
<tr>
<td>BRLOCK</td>
<td>NO CONTENTION</td>
<td>MUCH ATOMIC OPS</td>
</tr>
<tr>
<td>PRWLOCK</td>
<td>NO FENCE</td>
<td>IPI</td>
</tr>
<tr>
<td>PERCPU-RWSEM</td>
<td>NO FENCE</td>
<td>IPI</td>
</tr>
<tr>
<td>PRCU</td>
<td>NO BLOCKING</td>
<td>IPI</td>
</tr>
<tr>
<td>RCU</td>
<td>NO BLOCKING</td>
<td>IPI</td>
</tr>
</tbody>
</table>
USING ATOMIC OPS

- Ref Counter - Single
  - R: heavy contention
  - W: check one counter
USING ATOMIC OPS

- Ref Counter - Multiple
  - R: contention reduced
  - W: check more counters
USING ATOMIC OPS

- CSNZI - Hierarchical
  - R: contention reduced, may increase latency
  - W: check one counter
USING ATOMIC OPS

- Ref Counter - 1:1
  - R: No contention
  - W: check all counters
REMOVE FENCE

WHAT IS FENCE

1.WRITE X
2.READ Y

CPU0

CPU1

CPU2

CPU3

1.WRITE X
2.FENCE
3.READ Y

CACHE SYSTEM

CACHE

CACHE
LOCK WITH FENCE

1. \( R = 1 \)
2. FENCE;
3. IF \( W == 0 \)
4. ENTER CS
5. ELSE
6. WAIT

1. \( W = 1 \);
2. FENCE;
3. IF \( R == 0 \)
4. ENTER CS
5. ELSE
6. …

1
2
3
4
5
6

1
2
3
4
5
6

CPU0
CPU1
CPU2
CPU3

CACHE

CACHE SYSTEM
PROBLEM WITHOUT FENCE
INCONSISTENCY

1. R = 1
2. IF (W == 0)
3. ENTER CS
4. ELSE
5. WAIT

1. W = 1;
2. FENCE;
3. IF (R == 0)
4. ENTER CS
5. ELSE
6. …
PROBLEM WITHOUT FENCE
INCONSISTENCY

1. R = 1
2. IF (W == 0)
3. ENTER CS
4. ELSE
5. WAIT

1. W = 1;
2. FENCE;
3. IF (R == 0)
4. ENTER CS
5. ELSE
6. …

WHEN DO WE ENSURE READERS CAN SEE W=1?
THINKING ON HARDWARE I

BOUNDED STALENESS

CPU0 --> CACHE --> CPU1

CPU2 --> CACHE --> CPU3

CACHE SYSTEM

THE WRITER WILL SEE THE READER

W:1  R:1

WATI FOR BS
THINKING ON HARDWARE I

BOUNDED STALENESS

THE READER WILL SEE THE WRITER

CPU0 -> CACHE -> CPU1

CPU2 -> CACHE -> CPU3

CPU0
CPU1
CPU2
CPU3

CACHE SYSTEM

WATI FOR BS

1
W:1

2
R:1
THINKING ON HARDWARE II
BIASED FENCE

• Target: the buffer of issued instructions

• Light fence:
  • Async or Passively report the buffer info

• Heavy fence:
  • Wait for remote cores’ previously issued instructions committing
SOLUTION ON SOFTWARE
MONOTONE VERSION

READER

WRITER

GLOBAL_CNT++

LOCAL_CNT SYNC

CHECK

LOCAL_CNT SYNC

READER
WHAT IF READERS HAVE NO CHANCE TO SYNC
SUPPLEMENT
EVENTS & REDUCE EVENTS

- Event
- IPI
- Reducing Events
- Domain
SYNCHRONIZATION PHASES

INCREASE VERSION

No dependency: $O(1)$ if hardware support

WAIT FOR LEAVING

IPI

CHECK VERSION

P1

P2

P3

P4
SYNCHRONIZATION PHASES

INCREASE VERSION

No fence: $O(1)$ cache miss + $O(N)$ instructions (1 cycle on pipeline)

WAIT FOR LEAVING
CORRECTNESS

- Testing
  - Pass rcutorture (—torture rcu)
- Formal Verification
  - Pass model checking
FORMAL VERIFICATION

- Tool

- Target
  - prcu_read_lock, prcu_read_unlock, synchronized_prcu

- Hardware
  - 16 cores, Intel Xeon CPU@2.4GHz, 16G Memory

- Configuration
  - 2 reader threads + 1 writer thread + 1 main thread (+ 3 interrupt threads)
  - safety + liveness
  - Memory model : SC, TSO, PSO
PERFORMANCE

COMPARE WITH TREE RCU (LINUX 4.0.5)
SUMMARY

- Introduce a problem on reader-writer synchronization
- A solution call PRCU which has low latency on ideal hardwares
- Proof correctness with testing and formal verification
- Code: https://github.com/lihao/linux-prcu
THANKS