A Contention-Sensitive Fine-Grained Locking Protocol for Multiprocessor Real-Time Systems

CATHERINE E. JARRETT, BRYAN C. WARD, AND JAMES H. ANDERSON
Outline

Problem
Wait-For Graph Model
C-RNLP
Results
Future Work
Outline

**Problem**

Wait-For Graph Model

C-RNLP

Results

Future Work
Multiprocessor Platform

Considering: nested lock requests

Implementation: *coarse-grained locking* or *fine-grained locking*
Multiprocessor Platform

Considering: nested lock requests

Implementation: *coarse-grained locking* or *fine-grained locking*

```c
lock(a);
...
lock(b);
...
unlock(b);
...
unlock(a);
```
Multiprocessor Platform

Considering: nested lock requests

Implementation: coarse-grained locking or fine-grained locking

lock(a);
...
lock(b);
...
unlock(b);
...
unlock(a);
Multiprocessor Platform

Considering: nested lock requests

Implementation: *coarse-grained locking* or *fine-grained locking*

```c
lock(a);
...
lock(b);
...
unlock(b);
...
unlock(a);
```

*dynamic group locking*
Multiprocessor Platform

Considering: nested lock requests

Implementation: coarse-grained locking or fine-grained locking

```c
lock(a);
...
lock(b);
...
unlock(b);
...
unlock(a);
```

dynamic group locking
Multiprocessor Platform

Considering: nested lock requests

Implementation: coarse-grained locking or fine-grained locking

```
lock(a);
...
lock(b);
...
unlock(b);
...
unlock(a);

lock(c);
...
lock(d);
...
unlock(d);
...
unlock(c);
```

dynamic group locking
Multiprocessor Platform

Considering: nested lock requests

Implementation: coarse-grained locking or fine-grained locking

```c
lock(a);
...
lock(b);
...
unlock(b);
...
unlock(a);
```

```
lock(c);
...
lock(d);
...
unlock(d);
...
unlock(c);
```

**dynamic group locking**
Real-time Nested Locking Protocol (RNLP)

Processor count, \( m \)

Task count, \( n \)
Real-time Nested Locking Protocol (RNLP)

Processor count, $m$

Task count, $n$

Minimize: priority-inversion blocking (pi-blocking)
Real-time Nested Locking Protocol (RNLP)

Processor count, $m$

Task count, $n$

Minimize: priority-inversion blocking (pi-blocking)

Coarse-grained locking: $O(m)$ or $O(n)$ pi-blocking per request
Real-time Nested Locking Protocol (RNLP)

Processor count, $m$

Task count, $n$

Minimize: priority-inversion blocking (pi-blocking)

Coarse-grained locking: $O(m)$ or $O(n)$ pi-blocking per request

RNLP: $O(m)$ or $O(n)$ pi-blocking per request
Real-time Nested Locking Protocol (RNLP)

Contestion: the number of existing requests that access the same resources as the request of interest

Coarse-grained locking: $O(m)$ or $O(n)$ pi-blocking per request

RNLP: $O(m)$ or $O(n)$ pi-blocking per request
Real-time Nested Locking Protocol (RNLP)

**Contention**: the number of existing requests that access the same resources as the request of interest

Coarse-grained locking: $O(m)$ or $O(n)$ pi-blocking per request
RNLP: $O(m)$ or $O(n)$ pi-blocking per request
C-RNLP Goal: contention sensitive pi-blocking
Key Problem: transitive blocking chains
Key Problem: transitive blocking chains
Key Problem: transitive blocking chains

Proposed C-RNLP cutting ahead. Acquisition time for $R_3$ would not be delayed.
Key Problem: transitive blocking chains
Outline

Problem
Wait-For Graph Model
C-RNLP
Results
Future Work
Outline

Problem

Wait-For Graph Model

C-RNLP

Results

Future Work
Wait-for graph model

When is a request satisfied?

\[ R_1 \{d\} \]
Wait-for graph model

When is a request satisfied?
Wait-for graph model

When is a request satisfied?

\[ R_1 \{d\} \]

\[ R_2 \{c, d\} \]
Wait-for graph model

When is a request satisfied?
Wait-for graph model

When is a request satisfied?

- $P_1$
- $R_1 \{d\}$
- $R_2 \{c, d\}$
- $R_3 \{b, c\}$

The diagram shows a series of requests and processes. The requests are $R_1$, $R_2$, and $R_3$, with corresponding sets of resources they require. The processes are $P_1$, which has a connection to $R_1$. The diagram illustrates the dependencies and resource requests among the processes and requests.
Wait-for graph model

When is a request satisfied?
Wait-for graph model

When is a request satisfied?

\[
\begin{align*}
\mathcal{R}_1 & \rightarrow \{a, b\} \\
\mathcal{R}_2 & \rightarrow \{c, d\} \\
\mathcal{R}_3 & \rightarrow \{c, d\} \\
\mathcal{R}_4 & \rightarrow \{a, c\}
\end{align*}
\]
Outline

Problem

**Wait-For Graph Model**

C-RNLP

Results

Future Work
Outline

Problem
Wait-For Graph Model
C-RNLP
Results
Future Work
Outline

Problem
Wait-For Graph Model

C-RNLP
- Safety
- Delay Preservation
- Capacity of a Position

Results

Future Work
Safety

R_1

R_2

\{a\}
Safety

Add: \( R_3 \)

\( R_2 \) \( \rightarrow \) \( R_1 \)
Safety

Add: \( R_3 \)

\( \{a\} \)

\( R_2 \)

\( \{a\} \)

\( R_1 \)

\( \{a\} \)

\( P_1 \)
Safety

Add: $R_3$

Not safe!
Safety

Add: $R_3$

Diagram:

- $R_1$
- $P_2$
- $R_2$

Arrows:
- $R_1$ to $R_2$
- $R_2$ to $P_2$
Safety

Add: \( R_3 \)

Safe!
Safety

Add: \( R_3 \)

\( \{a\} \)
Safety

Add: \( R_3 \)

\( a \)

Deadlock!
Not safe!
Safety

Add: $R_3$

$P_4$ → $R_2$ → $R_1$
Safety

Add: \( R_3 \)  

Safe!
Safety

Add: \( R_3 \)
Safety

Add: $R_3$

Not safe!
Outline

Problem

Wait-For Graph Model

**C-RNLP**
- Safety
- Delay Preservation
- Capacity of a Position

Results

Future Work

**Safety**: There must be a unique ordering per resource and a new request may not cut ahead of a satisfied request.
Outline

Problem
Wait-For Graph Model

C-RNLP
- Safety
- Delay Preservation
- Capacity of a Position

Results

Future Work
Delay preservation
Delay preservation

![Diagram showing network nodes R1, R2, R3 with edge labels 4, 5, and 4+5=9]
Delay preservation
Delay preservation

Delay preserving!
Delay preservation
Delay preservation

Not delay preserving!

$R_2$ has increased worst-case blocking.

$4 + 2 = 6$
Delay preservation
Delay preservation

Delay preserving!
Delay preservation
Delay preservation

Delay preserving?
Worst-case blocking for $R_3$ depends on execution of $R_1$. 
Outline

Problem
Wait-For Graph Model

**C-RNLP**
- Safety
- Delay Preservation
- Capacity of a Position

Results
Future Work

---

**Delay Preservation**: The insertion of a new request must not increase the worst-case blocking of any request.
Outline

Problem

Wait-For Graph Model

**C-RNLP**
- Safety
- Delay Preservation
- Capacity of a Position

Results

Future Work
Capacity of a Position
Outline

Problem
Wait-For Graph Model
C-RNLP
Results
Future Work
Outline

Problem
Wait-For Graph Model
C-RNLP
Results
Future Work
Outline

Problem
Wait-For Graph Model
C-RNLP

Results
  ◦ Theoretical
  ◦ Experimental

Future Work
Theorem

A request $R_i$ is satisfied within $O(\min(m, c_i))$ time units.
Outline

Problem
Wait-For Graph Model
C-RNLP

Results
  ◦ Theoretical
  ◦ Experimental

Future Work
Outline

Problem
Wait-For Graph Model
C-RNLP

**Results**
- Theoretical
- Experimental

Future Work
Implementation of C-RNLP
Implementation of C-RNLP
Implementation of C-RNLP
Implementation of C-RNLP
Implementation of C-RNLP
Implementation of C-RNLP
Experimental Evaluation

Dual-socket, 18-cores-per-socket Intel Xeon E5-2699 platform

Comparison with:

◦ RNLP
◦ Mellor-Crummey and Scott’s queue lock (MCS)
Experimental Evaluation

Tasks \{2, 4, ..., 36\}

Resources \{1, 2, 4, 6, ..., 64\}

Requested resources \{1, 2, 4, 8, 12, ..., 48\}

Generated task sets to request resources as often as possible

Analyzing: overhead blocking and protocol blocking
Experimental Evaluation

overhead blocking

protocol blocking

---

**C-RNLP Lock**

```plaintext
procedure C-RNLP LOCK(requested)
    lock(Sublock)
    if Pending_requests > 0 then
        start ← (Head + 1) mod SIZE
        while (Table[start] & requested) ≠ 0 do
            start ← (start + 1) mod SIZE
        end while
    else
        start ← Head
    end if
    next ← (start + 1) mod SIZE
    Blocked[next] ← Blocked[next] + 1
    Pending_requests ← Pending_requests + 1
    Table[start] ← Table[start] | requested
    unlock(Sublock)
    while Enabled[start] ≠ 1 do
        /* null */
    end while
end procedure
```
Experimental Evaluation

overhead blocking

protocol blocking

---

C-RNLP Lock

procedure C-RNLP LOCK(requested)
lock(Sublock)
if Pending_requests > 0 then
    start ← (Head + 1) mod SIZE
    while (Table[start] & requested) ≠ 0 do
        start ← (start + 1) mod SIZE
    end while
else
    start ← Head
end if
next ← (start + 1) mod SIZE
Blocked[next] ← Blocked[next] + 1
Pending_requests ← Pending_requests + 1
Table[start] ← Table[start] | requested
unlock(Sublock)
while Enabled[start] ≠ 1 do
    /* null */
end while
end procedure
Experimental Evaluation

overhead blocking

protocol blocking

C-RNLP Lock

```
procedure C-RNLP Lock(requested)
  lock(Sublock)
  if Pending_requests > 0 then
    start ← (Head + 1) mod SIZE
    while (Table[start] & requested) ≠ 0 do
      start ← (start + 1) mod SIZE
    end while
  else
    start ← Head
  end if
  next ← (start + 1) mod SIZE
  Blocked[next] ← Blocked[next] + 1
  Pending_requests ← Pending_requests + 1
  Table[start] ← Table[start] | requested
  unlock(Sublock)
  while Enabled[start] ≠ 1 do
    /* null */
  end while
end procedure
```
Experimental Evaluation

overhead blocking

protocol blocking

C-RNLP Lock

```plaintext
procedure C-RNLP LOCK(requested)
    lock(Sublock)
    if Pending_requests > 0 then
        start ← (Head + 1) mod SIZE
        while (Table[start] & requested) ≠ 0 do
            start ← (start + 1) mod SIZE
        end while
    else
        start ← Head
    end if
    next ← (start + 1) mod SIZE
    Blocked[next] ← Blocked[next] + 1
    Pending_requests ← Pending_requests + 1
    Table[start] ← Table[start] | requested
    unlock(Sublock)
    while Enabled[start] ≠ 1 do
        /* null */
    end while
end procedure
```
Experimental Evaluation

overhead blocking = lock overhead

protocol blocking

---

C-RNLP Lock

```
procedure C-RNLP LOCK(requested)

lock(Sublock)
if Pending_requests > 0 then
    start ← (Head + 1) mod SIZE
    while (Table[start] & requested) ≠ 0 do
        start ← (start + 1) mod SIZE
    end while
else
    start ← Head
end if
next ← (start + 1) mod SIZE
Blocked[next] ← Blocked[next] + 1
Pending_requests ← Pending_requests + 1
Table[start] ← Table[start] | requested
unlock(Sublock)
while Enabled[start] ≠ 1 do
    /* null */
end while
```

end procedure
Tasks: 36
Available resources: 64
Tasks: 36
Available resources: 64

Resources Requested
Resources Requested

Tasks: 36
Available resources: 64
Tasks: 36
Available resources: 64
Tasks: 36
Available resources: 64

Resources Requested

Lock Overhead (microseconds)
Resources Requested

Lock Overhead

Tasks: 36
Available resources: 64
Observation 1. Runtime parallelism can increase worst-case lock and unlock overheads for the C-RNLP.

Tasks: 36
Available resources: 64
Available resources: 64
Requested resources: 4
Available resources: 64
Requested resources: 4
Available resources: 64
Requested resources: 4
Available resources: 64
Requested resources: 4
Available resources: 64
Requested resources: 4
Available resources: 64
Requested resources: 4
Observation 2. For the C-RNLP, observed overheads increase dramatically when resources are shared across sockets.
Observation 2. For the C-RNLP, observed overheads increase dramatically when resources are shared across sockets.
Observation 3. Fine-grained locking protocols have higher overhead than coarse-grained ones.
Tasks: 36
Available resources: 64
Requested resources: 2
Critical-section Length

Tasks: 36
Available resources: 64
Requested resources: 2
Tasks: 36
Available resources: 64
Requested resources: 2

Critical-section Length

Total Blocking Time (microsecond)

Critical-section Length (microseconds)
Observation 4. When critical-section lengths are greater than several microseconds and some parallelism is possible, the C-RNLP has less total blocking than previous protocols.
Observation 4. When critical-section lengths are greater than several microseconds and some parallelism is possible, the C-RNLP has less total blocking than previous protocols.
Observation 4. When critical-section lengths are greater than several microseconds and some parallelism is possible, the C-RNLP has less total blocking than previous protocols.
Contributions

**C-RNLP**, a contention-sensitive nested locking protocol
- Under certain conditions far out-performs existing protocols
- Otherwise does no worse than existing protocols
Future Work

Resource renaming

Lock server

C-RNLP variants corresponding to all RNLP variants
  ◦ short vs. long critical sections
  ◦ reader/writer
Questions?