Using Lock Servers to Scale Real-Time Locking Protocols: Chasing Ever-Increasing Core Counts

Catherine E. Nemitz, Tanya Amert, James H. Anderson
Contestion-Sensitive Access

\[ C_6 = 5 \]

\[ C_8 = 1 \]

\[ C_8 = 2 \]

blocking

time
Using Lock Servers to Scale Real-Time Locking Protocols: Chasing Ever-Increasing Core Counts
Challenge: Blocking Chains

# of processors: 
m=4

Processing capacity lost: 
75%

A B C D E
Challenge: Blocking Chains

# of processors: m=4

Processing capacity lost: 75%
Challenge: Blocking Chains

# of processors: m=32
Processing capacity lost: 97%

A B C D ...
Challenge: Blocking Chains

# of processors: m=32
Processing capacity lost: 97%

A  B  C  D  ...
Cost of low blocking: high overhead


Proposed Solution: Lock Servers
Four lock server paradigms

- Implementation
- Evaluation

Lock server coordination protocol
Platform Description

dual-socket, 18-cores-per-socket
Intel Xeon E5-2699

Socket 1
Cores 0-17
L1 Data
L1 Instr.
L2 (shared between two cores)
L3 (shared on socket)

Socket 2
Cores 18-36
L1 Data
L1 Instr.
L2 (shared between two cores)
L3 (shared on socket)
Platform Description

Socket 1
- Cores 0-17
- L1 Data
- L1 Instruction
- L2 (shared between two cores)
- L3 (shared on socket)

Socket 2
- Cores 18-36
- L1 Data
- L1 Instruction
- L2 (shared between two cores)
- L3 (shared on socket)

Graph:
- Overhead (microseconds) vs. Number of Tasks
- Lines for C-RNLP and MCS

Legend:
- L1 Data
- L1 Instruction
- L2 (shared between two cores)
- L3 (shared on socket)
Standard C-RNLP

- Core 8
  - request
  - enqueue
  - critical section
  - dequeue

- Lock state

- Socket 1

- Core 35
  - request
  - enqueue
  - critical section
  - dequeue

- Socket 2

- L3

- L1
Standard C-RNLP
The Idea

Remote Core Locking
  ◦ used to reduce critical-section lengths [1]

We developed: lock servers
  ◦ used to reduce overhead

Lock server: a process dedicated to performing lock and unlock functions

### Lock Server Paradigms

#### Mobility

<table>
<thead>
<tr>
<th>Mobility</th>
<th>Locality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>Global</td>
</tr>
<tr>
<td>Floating</td>
<td>Local</td>
</tr>
</tbody>
</table>

- **Static**
  - Global
  - Local

- **Floating**
  - #1
Paradigm #1: Static Global

Core 0

Lock Server

enqueue

dequeue

Core 8

request

enqueue

critical section

dequeue

Core 35

request

enqueue

critical section

dequeue

L3

L1

L3

L1

L1

L1
Paradigm #1: Static Global

Core 0
Lock Server
enqueue
dequeue

Core 8
request
enqueue
critical section
dequeue

Core 35
request
enqueue
critical section
dequeue
Paradigm #1: Static Global

Core 0
Lock Server
  - enqueue
  - dequeue

Core 8
  - request
  - submit
  - critical section
  - dequeue

Core 35
  - request
  - enqueue
  - critical section
  - dequeue

Unlock Server...
Paradigm #1: Static Global

Core 0
Lock Server
enqueue
dequeue

Core 8
request
submit
critical section
dequeue

Core 35
request
enqueue
critical section
dequeue

L1...L1
L3
L1...L1
L3
L1...L1
Paradigm #1: Static Global
Paradigm #1: Static Global
Paradigm #1: Static Global

Core 0
- Lock Server
  - enqueue
  - dequeue

Core 35
- request
  - enqueue
  - critical section
  - dequeue

Core 8

L3
L1
...
Paradigm #1: Static Global

Core 0

Lock Server

enqueue

dequeue

Core 8

Core 35

request

submit

critical section

dequeue

L1

L3

L1

...
Paradigm #1: Static Global

Disadvantage: lose a core
# Lock Server Paradigms

<table>
<thead>
<tr>
<th>mobility</th>
<th>Global</th>
<th>Local</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>#1</td>
<td></td>
</tr>
<tr>
<td>Floating</td>
<td>#2</td>
<td></td>
</tr>
</tbody>
</table>

**locality**
### Lock Server Paradigms

<table>
<thead>
<tr>
<th>mobility</th>
<th>locality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>Local</td>
</tr>
<tr>
<td>Floating</td>
<td>#2</td>
</tr>
</tbody>
</table>

- Lose 1 core
- + L1 cache affinity

- Global
- Local
Lock Server Paradigms

Key insight: blocked requests are busy-waiting, using CPU

- **Floating**
- **#2**

**Locality**
- **Global**
- **Local**

**Mobility**

Paradigm #2: Floating Global
Paradigm #2: Floating Global

Core 0

Core 8
  request
  submit
  critical section
  submit

Core 18

Core 35
  request
  submit
  critical section
  submit

L1 ... L1
L3
L1 ... L1
L1 ... L1
L1 ... L1
L3
L1 ... L1
Paradigm #2: Floating Global
Paradigm #2: Floating Global
Paradigm #2: Floating Global

Core 0
  request
  submit
  critical section
  submit

Core 8
  request
  submit
  critical section
  submit

Core 18
  request
  submit
  critical section
  submit

Core 35
  request
  submit
  critical section
  submit
Paradigm #2: Floating Global
Lock Server Paradigms

- **locality**
  - Global
  - Local
    - Lose 1 core
    + L1 cache affinity

- **mobility**
  - Static
  - Floating
    #2
### Lock Server Paradigms

#### Mobility
- **Static**
  - Lose 1 core
  - + L1 cache affinity
- **Floating**
  - No guaranteed cache affinity

#### Locality
- **Global**
- **Local**

*#3*
Lock Server Paradigms

Floating
- No guaranteed cache affinity

Global

Local

#3

Key insight: use multiple lock servers

locality

mobility
Paradigm #3: Floating Local
Paradigm #3: Floating Local
Paradigm #3: Floating Local

New challenge: coordination between lock servers
Lock Server Coordination

Server 1

1
2
3

A
B

Server 2

5
6
7

A
B

Global Execution Pattern

Reader/reader locking protocol (R^2LP)
Lock Server Coordination

Reader/reader locking protocol (R²LP)

Global Execution Pattern
Lock Server Coordination

Server 1
1
2
3

Server 2
1
2
3
4
5
6

Global Execution Pattern

Reader/reader locking protocol (R^2LP)
Lock Server Coordination

Original C-RNLP bound:

\[(c_i + 1)L_{\text{MAX}}\]

Reader/reader locking protocol (R^2LP)
Lock Server Coordination

Original C-RNLP bound:

\[(c_i + 1)L_{\text{MAX}}\]

Two-server C-RNLP bound with \(R^2\text{LP} \):

\[(c_{i,s} + 1)(L_{\text{MAX},1} + L_{\text{MAX},2})\]

Reader/reader locking protocol \((R^2\text{LP})\)

Global Execution Pattern

\[L_{\text{MAX},1}\]
\[L_{\text{MAX},2}\]
\[L_{\text{MAX},1}\]
\[L_{\text{MAX},2}\]
\[L_{\text{MAX},1}\]
\[L_{\text{MAX},2}\]
## Lock Server Paradigms

<table>
<thead>
<tr>
<th>Mobility</th>
<th>Locality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>- Lose 1 core</td>
</tr>
<tr>
<td></td>
<td>+ L1 cache affinity</td>
</tr>
<tr>
<td>Floating</td>
<td>- No guaranteed cache affinity</td>
</tr>
<tr>
<td></td>
<td>+ L3 cache affinity</td>
</tr>
</tbody>
</table>

**Global**

**Local**
Lock Server Paradigms

<table>
<thead>
<tr>
<th>mobility</th>
<th>Static</th>
<th>Floating</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Global</td>
<td>Local</td>
</tr>
<tr>
<td></td>
<td>- Lose 1 core + L1 cache affinity</td>
<td>- Lose multiple cores + L1 cache affinity</td>
</tr>
<tr>
<td></td>
<td>- No guaranteed cache affinity</td>
<td>+ L3 cache affinity</td>
</tr>
</tbody>
</table>

coordination with R^2LP, additional blocking considerations
Experimental Evaluation

Measured *overhead* and *blocking*

One task per core issuing 10,000 random requests

64 resources, each task requests 4 of these, critical-section lengths = 40µs
Experimental Evaluation

By how much can a lock server paradigm reduce worst-case overhead?

<table>
<thead>
<tr>
<th></th>
<th>Global</th>
<th>Local</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>- Lose 1 core</td>
<td>- Lose multiple cores</td>
</tr>
<tr>
<td></td>
<td>+ L1 cache affinity</td>
<td>+ L1 cache affinity</td>
</tr>
<tr>
<td>Floating</td>
<td>- No guaranteed cache affinity</td>
<td>+ L3 cache affinity</td>
</tr>
</tbody>
</table>

How do **overhead** and **blocking** differ between static and floating lock servers?
Experimental Results

Overhead (microseconds)

Number of tasks
Experimental Results

Overhead (microseconds)

<table>
<thead>
<tr>
<th>Number of tasks</th>
<th>No lock server</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>22</td>
</tr>
</tbody>
</table>

![Graph showing overhead vs. number of tasks for different configurations.]

- C-RNLP
- C-RNLP + SG
- C-RNLP + FG
- MCS

Overhead (microseconds)

Number of tasks
Experimental Results

Overhead (microseconds) vs. Number of tasks

- C-RNLP
- C-RNLP + SG
- C-RNLP + FG
- MCS

Static Global
Experimental Results

| Number of tasks | Overhead (microseconds) |
|----------------|
| Floating Global |

- C-RNLP
- C-RNLP + SG
- C-RNLP + FG
- MCS
Experimental Results

Static lock servers tend to reduce overhead more than floating lock servers.
Experimental Results

Overhead (microseconds)

Number of tasks
Experimental Results

<table>
<thead>
<tr>
<th>Overhead (microseconds)</th>
<th>Number of tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>22</td>
</tr>
</tbody>
</table>

![Graph showing overhead vs. number of tasks]

Overhead (microseconds) vs. Number of tasks graph with different line styles and colors for various conditions.
Experimental Results

<table>
<thead>
<tr>
<th>Number of tasks</th>
<th>Blocking (microseconds)</th>
<th>Overhead (microseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C-RNLP</td>
<td>C-RNLP + SG</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>200</td>
</tr>
<tr>
<td>10</td>
<td>200</td>
<td>400</td>
</tr>
<tr>
<td>15</td>
<td>300</td>
<td>600</td>
</tr>
<tr>
<td>20</td>
<td>400</td>
<td>800</td>
</tr>
</tbody>
</table>

Graph showing the increase in blocking and overhead with the number of tasks.
Experimental Results

With the use of a global lock server, blocking is nearly identical to that without using a lock server.
# Experimental Evaluation

How do **overhead** and **blocking** differ between global and local lock servers?

<table>
<thead>
<tr>
<th>Static</th>
<th>Global</th>
<th>Local</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- Lose 1 core</td>
<td>- Lose multiple cores</td>
</tr>
<tr>
<td></td>
<td>+ L1 cache affinity</td>
<td>+ L1 cache affinity</td>
</tr>
<tr>
<td>Floating</td>
<td>- No guaranteed cache affinity</td>
<td>+ L3 cache affinity</td>
</tr>
</tbody>
</table>
Experimental Results

![Graph showing overhead (microseconds) vs. number of tasks for different models: C-RNLP, C-RNLP + FG, C-RNLP + FL, and MCS. The graph demonstrates how overhead increases with the number of tasks.]
Experimental Results

<table>
<thead>
<tr>
<th>Number of tasks</th>
<th>No lock server</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>24</td>
</tr>
</tbody>
</table>

Overhead (microseconds)

- C-RNLP
- C-RNLP + FG
- C-RNLP + FL
- MCS

Number of tasks

No lock server
Experimental Results

<table>
<thead>
<tr>
<th>Overhead (microseconds)</th>
<th>Number of tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floating Global</td>
<td></td>
</tr>
</tbody>
</table>

- C-RNLP
- C-RNLP + FG
- C-RNLP + FL
- MCS
Experimental Results

Overhead (microseconds)

Number of tasks

C-RNLP
C-RNLP + FG
C-RNLP + FL
MCS

Floating Local
Experimental Results

Local lock servers reduce overhead more than global lock servers.
Local lock servers result in increased blocking compared to global lock servers.
Experimental Evaluation

Measured **overhead** and **blocking**

One task per core issuing 10,000 random requests

Parameter sweep:
- Number of tasks: \(\{2, 4, ..., 36\}\)
- Total # of resources: \(\{16, 32, 64\}\)
- # resources per request: \(\{1, 2, 4, 6, 8, 10\}\)
- Critical-section lengths: \(\{1, 20, 40, ..., 100\} \mu s\)
Four lock server paradigms

• Implementation
• Evaluation

Lock server coordination protocol

Contributions
Lock Server Coordination

Original C-RNLP bound:

\[(c_i + 1)L_{\text{MAX}}\]

Two-server C-RNLP bound with R²LP:

\[(c_{i,s} + 1)(L_{\text{MAX},1} + L_{\text{MAX},2})\]
C-RNLP bound

\[(c_i + 1)L_{\text{MAX}} = (7 + 1)(5) = 40 \text{ time units}\]

40 time units
Arbitrary Split

Server 1

Server 2

Blocking

Server 1: \((3 + 1)(5 + 5) = 40\)

Server 2: \((3 + 1)(5 + 5) = 40\)

\((c_{i,s} + 1)(L_{\text{MAX},1} + L_{\text{MAX},2})\)
Even Split by Critical-Section Length

Server 1

Server 2

Blocking

Server 1: \((3 + 1)(3 + 5) = 32\)

Server 2: \((3 + 1)(3 + 5) = 32\)

\((c_{i,s} + 1)(L_{MAX,1} + L_{MAX,2})\)
Uneven Split by Critical-Section Length

Server 1

Server 1: \((1 + 1)(1 + 5) = 12\)

Server 2

Server 2: \((5 + 1)(1 + 5) = 36\)

Blocking

\((c_{i,s} + 1)(L_{\text{MAX,1}} + L_{\text{MAX,2}})\)
Blocking Bounds

Baseline – no lock server

40 time units

Arbitrary split

Server 1: 40 time units
Server 2: 40 time units

Even split by critical-section length

Server 1: 32 time units
Server 2: 32 time units

Uneven split by critical-section length

Server 1: 12 time units
Server 2: 36 time units
Future Work

• Choose how to split tasks (based on requests) when using lock servers

• Explore accounting for static servers
  • Set server at highest priority, ensure lock state in cache
  • Use a dedicated IRQ-handling core
  • Treat as a special kind of interrupt

• Conduct a large-scale overhead-aware schedulability study
Questions?

dx.doi.org/10.4230/DARTS.4.2.2
Case Study: Four Sockets

four-socket, 6-cores-per-socket
Intel Xeon L7455

L1 data: 32KB
L1 instruction: 32KB
L2: 3MB
L3: 12MB
### Case Study: Profile of Requests

#### TABLE IV. Runnable Average Execution Times

<table>
<thead>
<tr>
<th>Period</th>
<th>Average Execution Times in μs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min.</td>
</tr>
<tr>
<td>1 ms</td>
<td>0,34</td>
</tr>
<tr>
<td>2 ms</td>
<td>0,32</td>
</tr>
<tr>
<td>5 ms</td>
<td>0,36</td>
</tr>
<tr>
<td>10 ms</td>
<td>0,21</td>
</tr>
<tr>
<td>20 ms</td>
<td>0,25</td>
</tr>
<tr>
<td>50 ms</td>
<td>0,29</td>
</tr>
<tr>
<td>100 ms</td>
<td>0,21</td>
</tr>
<tr>
<td>200 ms</td>
<td>0,22</td>
</tr>
<tr>
<td>1000 ms</td>
<td>0,37</td>
</tr>
</tbody>
</table>

---

#### TABLE II. Inter-Task Communication

<table>
<thead>
<tr>
<th>Period</th>
<th>1 ms</th>
<th>2 ms</th>
<th>5 ms</th>
<th>10 ms</th>
<th>20 ms</th>
<th>50 ms</th>
<th>100 ms</th>
<th>200 ms</th>
<th>1000 ms</th>
<th>sync</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ms</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 ms</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>I</td>
</tr>
<tr>
<td>5 ms</td>
<td>I</td>
<td>IV</td>
<td>IV</td>
<td>II</td>
<td>II</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 ms</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>VI</td>
<td>IV</td>
<td>II</td>
<td>II</td>
<td>III</td>
<td>IV</td>
<td></td>
</tr>
<tr>
<td>20 ms</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>IV</td>
<td>VI</td>
<td>II</td>
<td>I</td>
<td>II</td>
<td>IV</td>
<td></td>
</tr>
<tr>
<td>50 ms</td>
<td></td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>III</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100 ms</td>
<td>I</td>
<td>I</td>
<td>V</td>
<td>IV</td>
<td>II</td>
<td>VI</td>
<td>II</td>
<td>III</td>
<td>IV</td>
<td></td>
</tr>
<tr>
<td>200 ms</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000 ms</td>
<td>III</td>
<td>II</td>
<td>III</td>
<td>I</td>
<td>IV</td>
<td>I</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Case Study: Nested Requests

Handling Nested Requests

**Non-nested lock request**

```
lock(A)
  //critical section
unlock(A)
```

**Nested lock request**

```
lock(A)
  lock(B)
  //critical section
unlock(B)
unlock(A)
```

**With Dynamic Group Locks (DGLs)**

```
lock(A)
  //critical section
unlock(A)
```

```
lock(A, B)
  //critical section
unlock(A, B)
```
### Additional Experimental Results

<table>
<thead>
<tr>
<th></th>
<th>U-C-RNLP</th>
<th>U-C-RNLP + SGLS</th>
<th>U-C-RNLP + SLLS</th>
<th>U-C-RNLP + FGLS</th>
<th>G-C-RNLP + SGLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Firsts</td>
<td>0</td>
<td><strong>92</strong></td>
<td>0</td>
<td><strong>23</strong></td>
<td><strong>12</strong></td>
</tr>
<tr>
<td>Total Seconds</td>
<td>1</td>
<td>26</td>
<td>18</td>
<td>70</td>
<td>4</td>
</tr>
<tr>
<td>Total Thirds</td>
<td>68</td>
<td>2</td>
<td>17</td>
<td>20</td>
<td>8</td>
</tr>
<tr>
<td>Total</td>
<td>69</td>
<td><strong>120</strong></td>
<td>35</td>
<td><strong>113</strong></td>
<td>24</td>
</tr>
</tbody>
</table>

**Figure 16** Results of total request time comparison.
Challenge: Blocking Chains

Sequential resource acquisition: critical-section length $O(m^D)$


Dynamic group locks: no worst-case critical-section inflation, same asymptotic bounds