Fine-Grained Multiprocessor Real-Time Locking with Improved Blocking

Bryan C. Ward
James H. Anderson
Dept. of Computer Science
UNC-Chapel Hill
Motivation

• Locks can be used to control access to:
  • Shared memory objects
  • Shared hardware resources (e.g., buses, GPUs, shared caches, etc.)
• However, locks cause blocking.
• Our goal: improve worst-case blocking so as to improve real-time schedulability.
Multi-Resource Locking

• Sometimes jobs need to access multiple shared resources **concurrently**.

• Two general approaches:
  • **Coarse-grained** locking.
  • **Fine-grained** locking.

• The **Real-Time Nested Locking Protocol (RNLP)** was the first **fine-grained** locking protocol for multicore real-time systems.
Our Contributions

• We extend the RNLP in three ways:
  1. Reduce system-call overhead.
  2. Improve worst-case blocking when critical sections have different lengths.
  3. Support replicated resources.

• Techniques can be combined, but presented separately (unless noted).
RNLP Background

Token Lock

Request Satisfaction Mechanism

T_4

T_3

A

T_2 T_1

B


C
Different token locks can be used to achieve better blocking bounds under different schedulers and analysis assumptions.
Request satisfaction mechanism (RSM) orders the satisfaction of requests. We focus exclusively on it for the rest of the talk.
Fine-Grained Locking Challenge

A

B

C
Fine-Grained Locking Challenge

A

B

T_1

C
Fine-Grained Locking Challenge

A

T_2

B

T_1

C
Fine-Grained Locking Challenge

A

B

C

\( T_2 \)

\( T_1 \)

\( T_2 \)
Fine-Grained Locking Challenge
Fine-Grained Locking Challenge
Fine-Grained Locking Challenge

A

B

C

T_3

T_2

T_1

T_4
Fine-Grained Locking Challenge

A

B

C

T_2

T_1

T_4

T_3
Fine-Grained Locking Challenge

**Observation:** A later-issued request ($T_4$) blocks earlier-issued requests ($T_1$-$T_3$).
RNLP Solution

A

B

C
RNLP Solution

A

B

C

Placeholder
RNLP Solution

A

B

C

T_2

T_1

Placeholder
RNLP Solution

A

B

C

Placeholder
RNLP Solution

\[ \text{A} \quad T_3 \quad T_2 \quad \text{B} \quad T_2 \quad T_1 \quad \text{C} \quad T_1 \quad \text{Placeholder} \]
RNLP Solution

A

B

C

Placeholder
Invariant: A later-issued request never blocks an earlier-issued requests.
Dynamic Group Locks (DGLs)

- Goal: reduce number of lock/unlock calls.
- A job requests all potentially needed resources at once.
  - Request a subset of all resources.
  - Job may have to request resources that it doesn’t use.
- Scheduled when all resources available.
DGL Example

A

B

C
DGL Example

A

B

C

$T_1$

$T_1$
DGL Example

A

B

C

T₂

T₁

T₁
DGL Example

A

B

C
DGL Example

A

B

C

T_1

T_2

T_3

T_2

T_4
**Invariant:** A later-issued request never blocks an earlier-issued requests.
DGLs vs. RNLP

- Same worst-case blocking behavior.
- DGLs require **only one** lock and unlock call per outermost request instead of **one per resource**.
- RNLP may provide better **observed** blocking behavior.
Short vs. Long

- Critical sections may take longer for some resources than others. For example:
  - GPUs = long (milliseconds).
  - Shared memory objects = short (microseconds).
Short vs. Long Problem

A

T3

T2

Long

B

T2

T1

Long

C

T1

Short

Placeholder
Short vs. Long Problem

A
Long

B
Long

C
Short
Problem: A short request \((T_4)\) is blocked by the placeholder of a long request \((T_1)\)
Eliminating Short-on-Long Blocking

• Key ideas:
  • Long requests do not insert placeholders for short resources.
  • Long requests enqueue in short resource queues based on outermost lock request time.
  • Details in the paper show how different waiting mechanisms (spin vs. suspend) can be used for short and long resources.
  • Asymptotic optimality retained.
Multi-unit Locking

• Resource model:
  • For each resource, there are $k$ replicas.
  • Jobs require access to any of the $k$ replicas.

• Example use:
  • Shared hardware (e.g. GPUs).
k-RNLP

- FIFO-ordered queue per replica.
- At request time, the job is enqueued in the shortest replica queue for each potentially required resource.
- Could insert a placeholder for potentially needed resources.
Queue Structure

A
Replica 1  Replica 2

B
Replica 1  Replica 2
Queue Structure

A

Replica 1  Replica 2

B

Replica 1  Replica 2
Queue Structure

A

Replica 1  Replica 2

T_1  T_2

B

Replica 1  Replica 2

T_1
Queue Structure

A

Replica 1

T₁

T₃

Replica 2

T₂

B

Replica 1

T₁

Replica 2

T₃
Queue Structure

A

Replica 1 | Replica 2
---|---

T₁

T₃

B

Replica 1 | Replica 2
---|---

T₁

T₃
Queue Structure

A

Replica 1

T₁

T₃

Replica 2

T₄

B

Replica 1

T₁

T₃

Replica 2
Queue Structure

**Observation:** Later-issued requests ($T_4$) may be satisfied earlier by acquiring a different replica.
**k-RNLP Blocking Analysis**

- Queue length reduced by factor of $k$.
- **Worst-case** blocking is at most the maximum queue length.
- In the paper, we show that the $k$-RNLP can be configured for $O(m/k)$ or $O(n/k)$ blocking under different analysis assumptions.
- Asymptotically optimal.
Schedulability

\[ k = 2 \]
Schedulability

$k = 2$

System Utilization

HRT Schedulability

Schedulability

[1] RNLP
[2] OMLP
[4] CK-OMLP
[5] NOLOCK
Schedulability

$k = 2$

System Utilization

HRT Schedulability vs. System Utilization

Legend:
- RNLP
- OMLP
- K-RNLP
- CK-OMLP
- NOLOCK
Schedulability

$k = 2$

HRT Schedulability vs System Utilization graph.
Conclusions

• We presented three RNLP modifications:
  • **DGLs** - reduce system-call overhead.
  • Eliminate **short-on-long** blocking.
  • Support for **multi-unit** resources.
  • Modifications improve real-time **schedulability**.
  • Maintain **asymptotic optimality**.
Questions?