Be My Guest – MCS Lock Now Welcomes Guests

Tianzheng Wang, University of Toronto
Milind Chabbi, Hewlett Packard Labs
Hideaki Kimura, Hewlett Packard Labs
Protecting shared data using locks

```c
foo() {
    lock.acquire();
    data = my_value;
    lock.release();
}
```

Centralized spin locks

- Test-and-set, ticket, etc.
- Easy implementation
- Widely adopted
- Waste Interconnect traffic
- Cache ping-ponging

Contention on a centralized location
MCS Locks

Non-standard interface

```java
foo(qnode) {
    lock.acquire(qnode);
    data = my_value;
    lock.release(qnode);
}
```

Queue nodes everywhere

- Local spinning
- FIFO order
“...it was especially complicated when the critical section spans multiple functions. That required having functions also accepting an additional MCS node in its parameter.”

- Jason Low, HPE’s Linux kernel developer

Not easy to adopt MCS lock with non-standard API
“...out of the 300+ places that make use of the dcache lock, 99% of the contention came from only 2 functions. Changing those 2 functions to use the MCS lock was fairly trivial...”

- Jason Low, HPE’s Linux kernel developer

Not all lock users are created equal
– Transaction workers vs. DB snapshot composer
– Worker threads vs. daemon threads
Existing approaches

<table>
<thead>
<tr>
<th></th>
<th>Multi-process applications</th>
<th>Storage requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread-local queue nodes</td>
<td>Works</td>
<td>Bloated memory usage</td>
</tr>
<tr>
<td>K42-MCS</td>
<td>Queue nodes on the stack</td>
<td>Satisfies</td>
</tr>
<tr>
<td>Cohort locks</td>
<td>Works</td>
<td>Extra memory per node</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Possible data layout change</td>
</tr>
</tbody>
</table>
**MCSg:** best(MCS) + best(TAS)

Regular users

```plaintext
foo(qnode) {
    lock.acquire(qnode);
    ...
    lock.release(qnode);
}
```

Keeps all the benefits of MCS

Guests

```plaintext
bar() {
    lock.acquire();
    ...
    lock.release();
}
```

No queue node needed
MCSg: use cases

– Drop-in replacement for MCS to support guests
– Replace a centralized spinlock for performance
  – Start from all guests,
  – Gradually identify regular users and adapt
– As a building block for composite locks
  – Same interface as MCS
  – Same storage requirement
Guests in MCSg

\[ \text{Guests: similar to using a centralized spin lock} \]

\[ \text{lock} \]

\[ \text{CAS(NULL, } \pi \text{)} \]

acquire()

\[ \text{CAS(} \pi \text{, NULL)} \]

Kay until success

\[ \text{release()} \]

\[ \pi: \text{ “guest has the lock”} \]

Standard interface
Regular users – change in `acquire()`

\[ r = \text{SWAP}(N1) \]

\( \pi \)

No guest: same as MCS

waiting | NULL
Regular users – change in `acquire()`

\[ r = \text{SWAP}(N1) \]

`acquire(N1)` waiting | NULL
Regular users – change in \texttt{acquire()}

\[ r = \text{SWAP}(N1) \]

\texttt{acquire(N1)}

\[ t = \text{SWAP}(\pi) \]

\[ r == \pi, \text{ return } \pi \text{ for the guest to release the lock} \]

\[ t == N1/\text{another ptr} \]

\[ r == \text{NULL} \]

\text{Got lock}

\[ +5 \text{ LoC in } \texttt{acquire(...), no change in } \texttt{release(...)} \]
MCSg++ extensions

– Guest starvation
– CAS: no guaranteed success in a bounded # of steps
– Solution: attach the guest after a regular user
– FIFO order violations
– Retrying XCHG might line up after a later regular user
– Solution: retry with ticket
Reducing guest starvation

\[ r = \text{XCHG}(\pi) \]
\[ r.\text{next} = \text{Guest Waiting} \]
spin until \( r.\text{next} == \text{Guest Granted} \)
\[ r.\text{next} = \text{Guest Acquired} \]
Reducing guest starvation

\[ r = \text{XCHG}(\pi) \]
\[ r.\text{next} = \text{Guest Waiting} \]
spin until \( r.\text{next} == \text{Guest Granted} \)
\[ r.\text{next} = \text{Guest Acquired} \]
Reducing guest starvation

\[ r = \text{XCHG}(\pi) \]
\[ r\.next = \text{Guest Waiting} \]
spin until \( r\.next == \text{Guest Granted} \)
\[ r\.next = \text{Guest Acquired} \]
Reducing guest starvation

\[ r = \text{XCHG}(\pi) \]

\[ r\.next = \text{Guest Waiting} \]

spin until \( r\.next == \text{Guest Granted} \)

\[ r\.next = \text{Guest Acquired} \]
Reducing guest starvation

\[ r = \text{XCHG}(\pi) \]
\[ r.\text{next} = \text{Guest Waiting} \]

spin until \( r.\text{next} = \text{Guest Granted} \)

\[ r.\text{next} = \text{Guest Acquired} \]
Evaluation

– HP DragonHawk
  – 15-core Xeon E7-4890 v2 @ 2.80GHz
  – 16 sockets ➔ 240 physical cores
  – L2 256KB/core, L3 38MB/socket, 12TB DRAM

– Microbenchmarks
  – MCSg, MCSg++, CLH, K42-MCS, TATAS
  – Critical section: 2 cache line accesses, high contention

– TPC-C with MCSg in FOEDUS, an OSS database
Maintaining MCS’s scalability

– TPC-C Payment
– 192 workers
– Highly contented – one warehouse

<table>
<thead>
<tr>
<th>Lock</th>
<th>MTPS</th>
<th>STDEV</th>
</tr>
</thead>
<tbody>
<tr>
<td>TATAS</td>
<td>0.33</td>
<td>0.095</td>
</tr>
<tr>
<td>MCS</td>
<td>0.46</td>
<td>0.011</td>
</tr>
<tr>
<td>MCSg</td>
<td>0.45</td>
<td>0.004</td>
</tr>
</tbody>
</table>
One guest + 223 regular users

Throughput (10^6 acquire/s)

TATAS  K42  CLH  MCS  MCSg++  MCSg

224 regular users
One guest + 223 regular users

Starved

Guest average latency (ms)

- TATAS: 4.359
- K42: 0.226
- CLH: 0.259
- MCS: 0.126
- MCSg++: 0.005
- MCSg: 8669.66
Varying number of guests

Total throughput

Throughput (10^6 acquires/s)

Number of guests

TATAS
K42
MCSg
MCSg++

No ticketing
Varying number of guests

Guest throughput

Throughput ($10^6$ acquire/s)

- TATAS
- K42
- MCSg
- MCSg++
- MCSg+

Number of guests

1 2 4 10 15 30 60 120

No ticketing
Conclusions

– Not all lock users are created equal
  – Pervasive guests prevent easy adoption of MCS lock
– MCSg: dual-interface
  – Regular users: \texttt{acquire/release(lock, qnode)}
  – Infrequent guests: \texttt{acquire/release(lock)}
– Easy-to-implement: \texttt{\sim20} additional LoC
– As scalable as MCS (guests being minority at runtime)

\textbf{Find out more in our paper!}