Non-Blocking Timeout in Scalable Queue-Based Spin Locks

Michael L. Scott
University of Rochester
PODC 2002

- Motivation: context and problem
- Solution(s) and performance
- Impossibility conjecture

→ www.cs.rochester.edu/~scott/synchronization
Busy-wait mutual exclusion

- Preferable to rescheduling when
  » expected wait time is small
  » nothing else needs the processor
- Widely used in multiprocessor OSes; also some user-level programs
- **Scalability** the traditional problem for OSes and large scientific programs; solved with queue-based locks
- Preemption and deadlock the traditional problems in OLTP and related apps; solved with *timeout*
The CLH queue-based lock

- process 1 enters queue and runs; lock available
- process 1 leaves queue; 2 runs
- process 2 enters queue and waits
- process 2 leaves queue; lock free

Cf TAS, MCS
- speed, space, coherence
Spin locks with timeout (try-locks)

- Real time: signal error or pursue alternative code path
- OLTP
  » assume transaction deadlock or preemption of lock holder
  » abort current transaction
  » yield processor
- Easy in test-and-set lock, but not in queue-based lock!
  » must introduce neighbors to each other before leaving

→ Can we have our cake and eat it too?
Queue-based try-locks

- **PPoPP 2001** [Scott & Scherer]
  - CLH and MCS try-locks
    - $O(L+T)$ space
    - Fine for real-time or for deadlock recovery, but not for preemption recovery: timeout is not non-blocking

- **PODC 2002** (current contribution)
  - CLH-NB and MCS-NB try-locks
    - Preemption-safe: non-blocking timeout
    - Unbounded space worst case; can be modified to be $O(LxT)$; $O(L+T)$ expected
How to deal with preemption

- Need to be able to leave the queue without waiting for anybody else
- Craig [1993] proposed abandoning queue node, to be reclaimed at head of queue
  - Unbounded space if threads can re-request same lock; $O(L \times T)$ otherwise
- CLH-NB and MCS-NB-try locks
  - Immediate reclamation of abandoned nodes (by successor, not by departing thread, except at tail of list)
  - Worst-case space same as Craig, but $\sim (L+T)$ in practice
CLH-NB-try lock

- Reclaim dummy node in similar way in uncontested lock
- Complication: space management
  - lose “my” node when timing out
  - need a (lock-free) pool of nodes
Pathological case

- Three threads: A holds lock; initially B and C are waiting
  - B and C decide to leave; stop spinning on A and B’s qnodes
  - B marks own node; CAS fails; leaves queue; C is preempted
  - B requests lock again; gets in line with new node; times out; decides to leave; stops spinning on C’s node
  - C wakes up; marks own node; CAS fails; leaves queue; B is preempted

- Unbounded space worst case; O(TxL) if threads never re-try same lock, or re-use node if they do

Thanks to Victor Luchangco of Sun Labs
Multiprocessor lock-passing time (466MHz E10K)

- 225µs patience; 229ns critical, 440ns non-critical work

Graph showing lock-passing times for different workloads and lock schemes.
Modeling time-out time

\[ T_s = \frac{t_i}{m-1} \left[ T_l + (1-s)(T_p + T_h) + sT_w \right] \]

\[ \leq \frac{t_i}{m-1} \left[ T_l + T_p + (1-s)T_h \right] \]

\[ T_h \geq \frac{(m-1)T_s}{(1-s)t_i} - \frac{T_l + T_p}{1-s} \]

- \( t \) threads
- \( i \) # loop iterations
- \( m \) processors
- \( s \) acquire success rate
- \( T_s \) wall clock time
- \( T_p \) patience
- \( T_l \) loop overhead
- \( T_w \) average lock wait time
- \( T_h \) timeout/handshake time
Time-out time from model

8-processor Enterprise 4500 (336MHz)

- CLH-try
- MCS-try
- MCS-NB-try
- CLH-NB-try
Impossibility conjecture

- Cannot guarantee $O(L+T)$ space with queue and non-blocking timeout:
  - Imagine $N$ threads waiting in line
  - Middle $N-2$ decide (simultaneously) to leave
  - Need to link the edges together in order to reclaim space
  - Cannot do it in constant time!
Conclusions

- Contention matters, more now than ever.
- Scalability and timeout are compatible.
  - Potentially significant benefit for important commercial applications
- Synchronization is still an open problem :-)

- www.cs.rochester.edu/~scott/synchronization