redhat.

Linux Kernel Synchronization Primitives on Large NUMA Systems

Waiman Long
Principal Software Engineer, Red Hat Inc
Feb 20, 2019

Linux Kernel Synchronization Primitives

- Atomic operations
- RCU (Read-Copy-Update)
- Spinning Locks (can be used in non-process context)
  - spinlocks
  - rwlocks
  - seqlocks (spinlock + sequence number)
- Sleeping Locks (process context only)
  - mutexes (exclusive locks)
  - RW semaphores
  - Others (rt-mutexes, percpu-rwsems, semaphores)

Spinning Locks

- Two main issues with spinning lock performance:
  - Unfairness (unfair locks can cause lock starvation)
  - Cacheline bouncing (due to lock cacheline spinning by multiple CPUs)
- Spinlocks: TAS locks ==> Ticket locks (fair) ==> Queued locks (fair & local cacheline spinning)
- Queued spinlocks combine a TAS lock with an MCS lock into a combination lock.
- Rwlocks: Unfair rwlocks ==> Queued rwlocks (fair & local cacheline spinning with qspinlock)

Sleeping Locks

- The main performance issue with sleeping locks is lock waiter wakeup latency.
- The solution to this performance problem is to make a sleeping lock behaves somewhat like a spinning lock – optimistic spinning (if lock owner is running) and lock stealing. This was first introduced in mutexes.
- This introduced the same spinning lock performance problems – unfairness and cacheline bouncing.
- The cacheline bouncing problem was eliminated with the introduction of MCS lock for queuing. This was later enhanced to an OSQ (optimistic spinning queue) lock to lock backing out from the queue.
- The unfairness problem was eliminated by introducing a lock handoff mechanism to force transfer of lock ownership to the first waiter in the wait queue.
- Optimistic spinning was then added to writers waiting for rw semaphores to eliminate performance degradation when converting mutexes to rw semaphores.

In Development

- NUMA-aware queued spinlocks
- RW semaphore lock handoffs and reader optimistic spinning.


Q & A
redhat.