CS3210: Scaling Operating Systems: A Case Study

Tim Andersen

Lecture plan

- Ticket Locks
- Performance collapse from non-scalable locks
- Reasons for collapse
- MCS Locks

Lock Contention Performance

- Non-scalable locks like spin locks have poor performance when highly contended
- Many systems, nevertheless, use them, including the Linux kernel
- Performance degradation is not a gradual leveling off (diminishing returns) but a sudden collapse
- A system with 10 cores can performance considerably better than one with 15 or 40

Why it's important

- Non-scalable locks severely degrade performance in common scenarios like file system access, network services, and memory mapping.
- As systems gain more and more cores, contention becomes more common
- Onset of performance collapse can be sudden as cores are added, meaning hardware upgrades without corresponding software upgrades can be catastrophic for system performance
- Critical sections can be very short and still collapse performance

Non-scalable ticket lock default in Linux kernel

```
struct spinlock_t {
    int current_ticket;
    int next_ticket;
}

void spin_lock(spinlock_t *lock)
{
    int t = atomic_fetch_and_inc(&lock->next_ticket);
    while (t != lock->current_ticket)
        ; /* spin */
}

void spin_unlock(spinlock_t *lock)
{
    lock->current_ticket++;
}
```

Q: Why use spinlocks instead of sleep/wakeup?

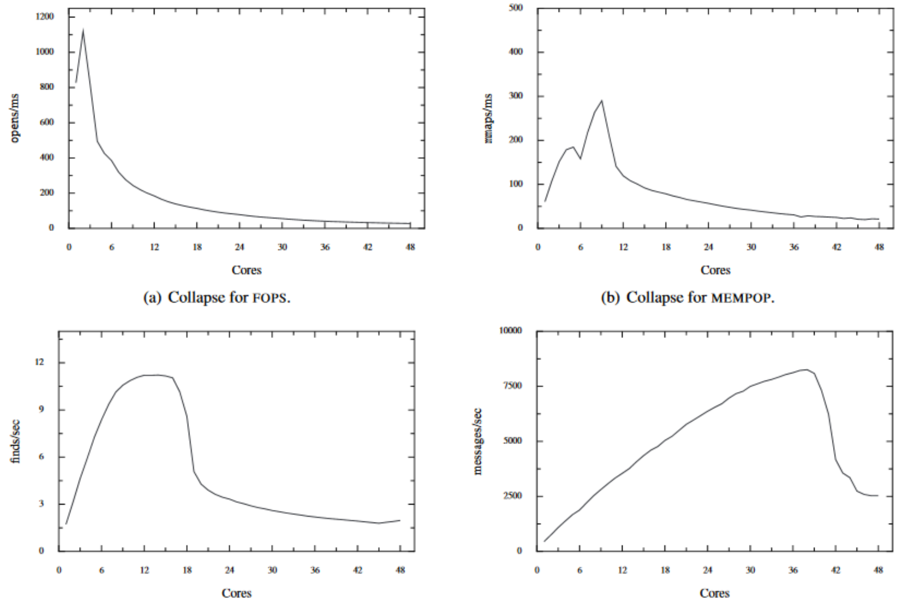
Why lock contention is slow

- If many cores are waiting, lock variables will be cached
- Unlock will invalidate all cached entries
- All cores will read the cache line
- Cache reads are serialized in most architectures
- Next in line core will receive updated cache line on average half-way through complete update
- Lock handoff increases in proportion to number of waiting cores
- Inter-core operations take 100s of cycles, meaning 1000s of cycles are used if dozens of cores are waiting

Benchmarks of Linux kernel locking

- FOPS: creates a file and starts one process on each core, repeatedly opens and closes
- MEMPOP: creates a process on each core, repeatedly maps 64 KB of memory with MAP_POPULATE flag, then munmaps
- PFIND: searches for a file by executing GNU find utility
- EXIM: mail server listens for connections and forks for each new connection.

Results of benchmarks



(a) Collapse for FOPS.

(b) Collapse for MEMPOP.

(c) Collapse for PFIND.

(d) Collapse for EXIM.

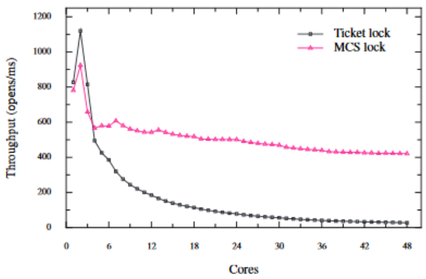
Causes

Benchmark	Operation time (cycles)	Top lock instance name	Acquires per operation	Average critical section time (cycles)	% of operation in critical section
FOPS	503	d_entry	4	92	73%
MEMPOP	6852	anon_vma	4	121	7%
PFIND	2099 M	address_space	70 K	350	7%
EXIM	1156 K	anon_vma	58	165	0.8%

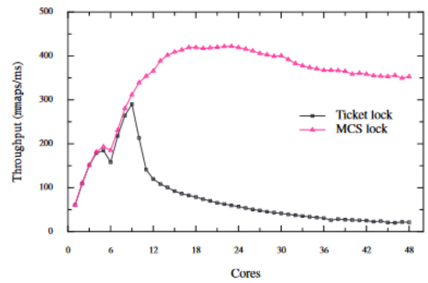
Solution: Scable Locks (MCS)

- Spins on local rather than global variable
- Each core is a node in a queue
- If lock is held, core registers itself by adding node to the queue
- Busy wait on own is_locked field
- When unlocking, set next in line is_locked field to false.
- Lock acquisition is now O(1) in number of cores instead of O(N)

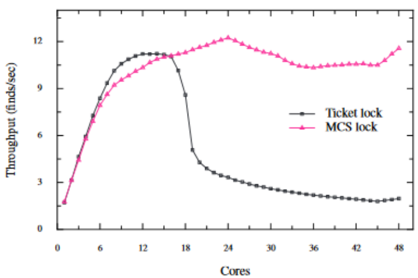
Results comparison



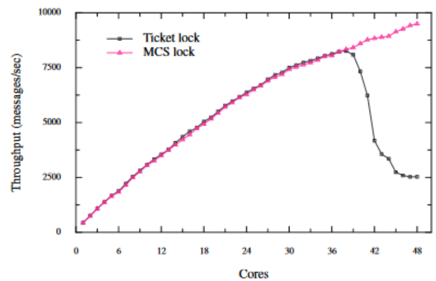
(a) Performance for FOPS.



(b) Performance for MEMPOP.



(c) Performance for PFIND.



(d) Performance for EXIM.