SOA paper crunch

Lecture 4

Why the Grass May Not Be Greener On The Other Side: A Comparison of Locking vs. Transactional Memory

Paul E. McKenney, Maged M. Michael, Josh Triplett, Jonathan Walpole

Advanced Operating Systems

October 26, 2011

SOA paper crunch

Outline

Context

- parallelism is norm
- synchronization
- locking – long and viable record of production use
- transactional memory (TM) – promising synchronization mechanism
- constructive critique

Locking

- mutual exclusion
- critical section
- overhead
- contention
- mutex, semaphore, spinlock

Transactional Memory

- "atomic transaction" for group of memory operations
- lock-free operations
- basic operations
  - load transaction (read), store transaction (write)
  - commit (successful if no conflict)
  - abort
  - validate
- extended LL/SC (load-link, store-conditional)
- software/hardware (STM, HTM)

Outline

Strengths

- intuitive and easy to use
  - one CPU may manipulate a given object
- can be used on any hardware
- ubiquitous
  - well defined API
  - large body of software using locking
  - large number of experienced developers
- contention concentrated within locking primitives

Strengths (2)

- minimal performance degradation with locking
- wide range of protected operations (including non-idempotent)
- natural interactions with a large variety of synchronization mechanisms (RCU, atomic operations)
- natural interaction with debuggers and other software tools

Weaknesses

- deadlock
  - more than one lock
  - random ordering
  - non-composable
  - interrupt or signal handlers
- priority inversion
  - low-priority thread holds lock
  - medium-priority thread preempts low-priority thread
  - high-priority thread attempts to acquire lock and blocks
  - medium-priority thread runs instead of high-priority thread

Weaknesses (2)

- data partitioning
  - some data structures may not be partitioned
- expensive cache misses
- blocking synchronization primitive
- locking acquisition is non-deterministic

Improvements

- deadlocks
  - locking hierarchy
  - conditional lock acquisition
  - masking signals or interrupts
- priority inversion
  - priority inheritance
  - raising the lock holder's priority
  - preemption disabling when locks are held
  - RCU

Improvements (2)

- partionable data
  - hash tables
  - radix trees
- contention, overhead
  - specialized designs, patterns
  - RCU
- preemption, blocking, page faulting
  - scheduler-conscious synchronization
- non-deterministic lock acquisition
  - RCU (read-side critical section)
  - FCFS primitives

Remaining Challenges

- software tools to aid in static analysis
- software tools to evaluate lock contention
- better codification of effective design rules
- augmenting synchronization methodologies
- locking algorithms for good scalability and performance for ill-structured update-heavy non-partionable data structures

Outline

Strengths

- atomic execution of operations spanning multiple object
- simplicity
  - load and store sequence of memory
  - atomic, linearizable transactions
- composability
  - may be nested
  - span multiple data structures

Strengths (2)

- scalability
  - small transactions rarely conflict (intersect)
  - fine grained locking without effort and complexity
- non-blocking operations
  - thread failure doesn't affect another thread
- hardware implementation, durable usage in database systems

Weaknesses

- non-idempotent operations
  - may be performed multiple times upon transaction retry
  - client buffers message and commits after server reply
- poor interaction with existing synchronization mechanisms
  - 300% overhead for locks acquired within transactions
- excessive conflicts
  - data structures that impede fine-grained locking
  - software may be designed to avoid this problem

Weaknesses (2)

- rollback overhead
  - in case of conflicts
  - starvation
- sparse hardware support (HTM)
  - not in commodity hardware
  - portability problems
- performance issues
  - atomic operations
  - dynamic allocations
  - memory reclamation
  - data copying
  - bookkeeping
- no standard API
- poor interaction with existing tools

Improvements

- non-idempotent operations
  - include buffering mechanism with the scope of the transaction
  - apply simple locking
- partitioning – sample designs that apply to locking
- rollback overhead
  - contention manager
  - convert read-only transactions to non-transactional form
- focus on STM (performance issue?), debugging HTM?
- apply transactions to heavyweight operations (system calls)

Outline

Where Does TM Fit in?

- update-heavy workloads on large non-partitionable data structures
- complex fine-grained locking designs incurs complexity to avoid deadlock
- single threaded software having an embarrassingly parallel core

Conclusion

- the grass is not necessarily uniformly greener on the other side
- work required to integrate synchronization mechanisms
- combining strengths

Outline

Keywords

- research
- papers
- group
- conference

Resources

- Maurice Herlihy, J. Eliot B. Moss – Transactional Memory: Architectural Support for Lock-Free Data Structures
- Paul. E McKenney, Maged M. Michael, Josh Triplett, Jonathan Walpole – Why the Grass May Not Be Greener On The Other Side: A Comparison of Locking vs. Transactional Memory

Outline

Questions

?