Contextual Concurrency Control

Sujin Park    Irina Calciu    Taesoo Kim    Sanidhya Kashyap
Locks in everywhere!

Operating systems  Cloud services  Data processing systems  Databases

Synchronization mechanisms
Basic building block for designing applications
Locks are critical for application performance

Typical application performance on a multicore machine
Locks are critical for application scalability

Typical application performance on a manycore machine

Operations / second

# threads

Linear scalability

Far from ideal
One lock cannot rule all of them!

Evolving hardware

Read-intensive workload

Various applications & requirements
Specialization bridges the semantic gap
Specialization bridges the semantic gap
Can we tune lock policy on the fly?

Contextual Concurrency Control

New paradigm to tune synchronization mechanism from user space
Need for user-defined locks on the fly

- Lock implementations are application agnostic
- Only few locks contend for a given application
- May need a variant of a lock based on the workload
**Concord Framework**

- **Lock implementations are application agnostic**
  - Let application developers tune locks in the kernel on the fly

- **Only few locks contend for a given application**
  - Modify set of locks at various granularities

- **May need a variant of a lock based on the workload**
  - Exposes set of APIs to modify lock algorithms
CONCORD Overview

Example:

```c
bool cmp_node(node* pre, node* cur){
    return (pre->sock == cur->sock);
}
```

Grouping node from same socket
**Concord Overview**

1. User create **lock policy**
2. Load the policy
3. Verify given lock policy

- Memory access
  - Lock shouldn't be changed arbitrary
- Helper functions
  - Only whitelisted functions can be called
- Code termination
  - No hanging policy
**Concord Overview**

1. **User**
   - User creates lock policy
   - Memory access
   - Helper functions
   - Code termination
   - Boolean function `cmp_node(node* pre, node* cur){ return (pre->sock == cur->sock); }

2. **Kernel**
   - Load the policy
   - Validate given lock policy
   - Lock shouldn't be changed arbitrary
   - Only whitelisted functions can be called
   - No hanging policy

3. **Verifier**
CONCORD Overview

1. User create lock policy
2. Load the policy
3. Verify given lock policy
4. Create patch to specify target point
5. Patch locking function to run with given policy

- All spinlocks in the kernel
- Spinlocks used in filesystem
- A spinlock used in an inode
Safety and APIs

Reordering waiters

• bool cmp_node(lock, node, node){}
• bool skip_shuffle(lock, node){}

Flexibility to change lock on the fly

Fairness

Profiling

• void lock_acquire(lock){}
• void lock_contended(lock){}
• void lock_acquired(lock){}
• void lock_release(lock){}

Fine-grained lock profiling

Increase critical section

Ensure mutual exclusion & safe from crashing
Usecase

1. Avoiding Scheduler Subversion using Scheduler–Cooperative Locks. Eurosys’20

- t1 and t2 competing for the same lock
- t1 holds the lock 3 times longer
- t1 will receive 3 times much CPU time
- t2 will receive fair CPU time competing for the same lock
- t2 is penalized: have less opportunity to grab a lock

Scheduler-Cooperative Locks

1. Avoiding Scheduler Subversion using Scheduler–Cooperative Locks. Eurosys’20
**Usecase**

Let application developers enforce this fairness only when needed.

---

1. Avoiding Scheduler Subversion using Scheduler–Cooperative Locks. Eurosys’20
Overhead of CONCORD

- Overhead of **CONCORD-lock** compared to **pre-compiled lock**

- Almost negligible overhead (And now *we can change lock on the fly!*)

![Graph showing BRAVO lock performance vs number of threads]
Conclusions

• Kernel locks are critical for application performance and scalability
  • Out of the reach of application developers

• C3 : Contextual Concurrency Control
  • Let userspace application to fine tune concurrency control

• CONCORD Framework
  • Exposes a set of APIs
  • Apply to specific target locks (instead of all locks in the kernel)
  • Change locks on the fly with minimal overhead