Achieving Balanced Lock Usage Fairness and Lock Utilization using Wuji-Locks

Leping Li*, Xueheng Wang*, Yuvraj Patel

1 Abstract

As processor clock speeds plateau, software developers increasingly turn to concurrency and multicore machines to boost performance. Locks are crucial for synchronizing concurrent events and ensuring mutual exclusion and are commonly used in building concurrent software like operating systems, servers, and key-value stores. Lock designers focus on specific properties that locks should exhibit while designing locks, thereby representing different worlds. For example, spinlocks are simple locks where the cache coherence governs who will acquire the lock and is designed with a focus on high lock utilization and performance. However, they cannot ensure lock usage fairness and may starve one or more threads. On the other hand, Scheduler–Cooperative Locks (SCLs) are complex locks that track the lock usage of the competing entities and dedicate a window of opportunity where only a single entity can acquire the lock multiple times while all other entities wait for their opportunities. SCLs guarantee lock usage fairness by penalizing dominant lock entities to give more opportunities to other entities. Thus, SCLs are non-work conservative and may compromise lock utilization to guarantee lock usage fairness. We observe none of the existing locks can effectively ensure high lock utilization and lock usage fairness. In this work, we ask the question – is it possible to design a lock that can guarantee both high lock usage fairness and high lock utilization.

To address the problem, we understand the desired behavior by identifying under what circumstances one can achieve high lock usage fairness and high utilization and make specific observations. Using these observations, we build Wuji Locks (W-Locks) – a new family of locking primitives that can guarantee both high lock usage fairness and high lock utilization. Like SCLs, W-Locks provide a window of opportunity to entities. However, other compatible entities also share the same window whose critical and non-critical sections align well, thereby increasing the lock utilization. W-Locks prioritize non-dominant entities while forming groups that can share the window of opportunity and penalize dominant entities. Our design avoids scalability collapse, ensures lock acquisition fairness, and incurs minimal overhead.

W-Locks implementation comprises two components – mechanism and policy. The mechanism handles the lock acquisition and release procedures. The policy represents the strategy determining the lock behavior exhibiting desired lock properties. Our design provides both robustness and flexibility, enabling W-Locks to adapt to diverse goals and environments with minimal effort. We implement three locks comprising a userspace W-Lock, a NUMA-aware W-Lock, and a reader-writer lock.

Using microbenchmarks and real-world applications, we show W-Locks can achieve high lock usage fairness and lock utilization in varied and extreme scenarios compared to existing locks. Additionally, we perform lock over-

*Both authors contributed equally to this work.

head and latency sensitivity study to show W-Locks can scale well up to 40 CPUs and deliver low latency to latency-sensitive applications. To show real-world applicability, we port W-Locks to UpScaleDB, KyotoCabinet, and Memcached to compare the performance of W-Locks against other state-of-the-art locks. Our experiments show that W-Locks can dramatically enhance the performance of real-world applications.