Paper Critique – Week 9

Name: PHAM CONG VINH Student ID: 2019711010

Everything You Always Wanted to Know About Synchronization but Were Afraid to Ask

This paper present a comprehensive study about synchronization on multi-core system in term of CPU architecture (single/multi socket), locking scheme (the type, algorithm). The motivation of this paper is the lack of o results on modern architectures connecting the low-level details of the underlying hardware, e.g., the cache-coherence protocol, with synchronization at the software level.

The authors considered multiple synchronization layers, from basic many-core hardware up to complex concurrent software, which are cache-coherence protocols, the performance of various atomic operations, locking and message passing techniques, concurrent hash table, an in-memory key-value store, and a software transactional memory. We can say that all element or aspect that relate to synchronization was inspected in this paper.

The strength of the paper is that the author have conducted their study with rare and difficult to approach CPU architecture (Niagara and Tilera) over popular CPU architecture (Intel Xeon and AMD Opteron) in order to find out how uniform and non-uniform design affect the synchronization performance. Also, their customly crafted experiment tool - cross-platform synchronization suite is attractive and a big contribution to researchers who want a standard tool that span all aspect of synchronization in order to conduct their experiments.

The limitation of the paper is what the author also stated in their paper. The author haven't consider the role of lock-free and "serializing critical sections over message passing" techniques, how these techniques affect the synchronization.

An Analysis of Linux Scalability to Many Cores

This paper analyzed the scalability of Linux kernel which run on a 48-core CPU system. This study aim to clarify whether the traditional architecture and design of operating system's kernel able to scale up on multi core system the number of core increasing day by day. An important observation this paper showed us is that in reality, parallel tasks usually interact and interaction usually forces serial execution, and this is when scaling bottleneck happened. Scaling bottlenecks limit performance to some maximum, regardless of the number of cores.

Author's methodology are first they crafted a benchmark tool (they named MOSBENCH APPLICATIONS) which stressed many aspect of the Linux kernel (scheduler, the network stack, file name cache, page cache, memory manager, process manager) to find out the scalability bottleneck is whether fault in the Linux kernel or the application itself. Second, the author fix those bottlenecks by modifying the applications or the kernel and finally put altogether into an optimization solution which result in a new patch for Linux kernel (named Patched Kernel, PK); and an conclusion that there is no reason or clear evidence that force us to change the current kernel design in order to gain more scalability.

The strength of the paper is author's methodology. The conducted an exhaustive experiment, benchmark the scalability with a set of diversity application, workload that stressed many aspect of the kernel. The author convinced audience that with right and modest optimization to Linux kernel as well as the application itself, we could gain the higher scalability with modern multi core system.

The limitation of the paper is although, if the kernel is perfect for scaling, the application deployment may induce the bottleneck by I/O. Enhancing the operating system's kernel is not enough for the scalability.

Read-Log-Update A Lightweight Synchronization Mechanism for Concurrent Programming

This paper introduced a novel synchronization mechanism, RLU, which overcome the RCU's limitation in number of concurrent writer. RLU supports concurrency of reads with multiple writers. The author stated this is the first mechanism that support these advantages.

The motivation of this paper is, although RCU has provides many benefits for concurrency read and write operation management, RCU is not a complete solution and remains limitation in performance as well as weakness in its data structures. Moreover, RCU required the programmer efforts to coordinate their code operating. In reverse, RLU overcome those limitation by its unique design which based on clock-based logging mechanism inspired by the ones used in software transactional memory systems.

The strength of the paper is that the author constructed their proposed solution on and over the most famous and popular equivalent mechanism, RCU. This strategy convinced the audience that author's solution is an innovating of the good solution and worth to be adapted, applied. Also, the paper confronted RLU with RCU in term of the reality scenario that exhibit the RCU's weakness, Kyoto Cabinet. This help justify RLU is better than RCU.

The weakness of the paper is its lack of background about concurrency control. This required new audience must refer to others to gain some basic knowledge, terminology to catch the idea of this paper.

The Scalable Commutativity Rule: Designing Scalable Software for Multicore Processors

This paper presented a concept about commutativity rule applying on interface design in order to implement better scalability software. According to this rule, operations commute meaning there's no way to distinguish their execution order using the interface; whenever interface operations commute, they can be implemented in a way that scales.

The author constructed this concept into reasoning theory which help to decide whether a designed interface (API) can be implemented to scalable implementation. Normally, Complex interfaces can make it difficult to spot and reason about all commutative cases. The COMMUTER, which is the automate reasoning tool of the "Commutativity Rule" tool, takes an interface model in the form of a simplified and computes precise conditions under which sets of operations commute, and tests an implementation for conflict-freedom under these conditions.

The strength of this paper is that the author employed their proposed solution itself to guide the implementing of a mini operating system call sv6 and compared this implementation with Linux in term of benchmark the scalability of system calls. This convinced audience how good the concept is by show its application in practice.

It's not really a limitation but the paper is quite complicate to student. It's more reasonable to structure the paper simpler and leave all complexity word to the Appendix.