1. **Everything you always wanted to know about synchronization but were afraid to ask, SOSP 2013**

This paper presents the most exhaustive study of synchronization to date. Ultimately, it derives a series of observations that suggest that the scalability of synchronization is primarily a attribute of hardware.

This paper has an advantage in that it has been presents that cross-platform synchronization suite, SSYNC, can be used to evaluate the potential for scaling synchronization on different platforms and to develop concurrent applications and systems.

The weakness is that there are no consequences for the relationship of power consumption to various synchronization scheme in the multicore.

It would have been nice to show those results and the same experiment in different architectures.

2. **Read-Log-Update: A Lightweight Synchronization Mechanism for Concurrent Programming, SOSP 2015**

This paper introduces read-log-update (RLU), a novel extension of the popular read-copy-update (RCU) synchronization mechanism that supports scalability of concurrent code by allowing unsynchronized sequences of reads to execute concurrently with updates.

RCU programming is notoriously difficult because all the changes need to properly use single pointer update. To mitigate this issue, RLU extends the RCU framework to ease concurrent programming. RLU allows read-only traversals, while supporting multiple updates by internally maintaining copies of multiple objects in a per-thread log, which is similar to the reader-writer programming model.

It has strengths because RLU improves programmability in two ways. Firstly, it allows atomic multi-pointer updates which simplifies the design of many concurrent data structures like doubly linked list and tree. Secondly, because every reader gets a consistent snapshot of the data-structure protected by RLU, it removes the need of data structure specific pre-commit validation step required by most lock-free data structures.

But it has weakness that trees are also unbalanced due to the difficulty of synchronizing updaters. And also has weakness that how many object versions RLU maintains so that when more than create maximum number, it may occur the synchronously waiting.

So it is better if there is a mention of these weakness.

3. An Analysis of Linux Scalability to Many Cores, OSDI 2010

This paper analyzes the scalability of seven system applications (Exim, memcached, Apache, PostgreSQL, gmake, Psearchy, and MapReduce) running on Linux on a 48-core computer.

But also kind of a big idea in this paper, I think it is the strength of this paper. People (including the authors) have been proposing that traditional OS designs would not scale to many-core machines. But in turns out rather than considering a complete rewrite of entire legacy operating system, all we have to do is fix into the scalability bottlenecks and this paper show how to do it via new technique, sloppy counters that allow us to replace just shared counters few uses without modifying the many other uses in the kernel.

The weakness is that it was not known what is the roots of the problem and if they had same results for different platforms

It is better if there is a mention of these weakness.

4. The Scalable Commutativity Rule: Designing Scalable Software for Multicore Processors, SOSP 2013

In this paper, they introduce the idea of using commutativity as an interface design requirement to improve scalability. They demonstrate opportunities out there for more scalable interfaces and presents definitions and proofs of the scalable commutativity rule. They also describe a tool (COMMUTER) for automatically developing test cases and testing if a particular implementation commutes.

They present formalization of the rule and proof of its correctness in paper. They present a systematic, test-driven approach to applying commutativity rule to real implementations embodied in a tool named COMMUTER. So these are the strength of this paper.

But, it is wonder that this rule and tool helps in distributed environment? I was wondering if the tool components can be used elsewhere independently. The weakness is that there is no mention of differences from distributed systems, not just multi-core systems.

It is better if there is a mention of this weakness. And It would have been better had there been a detailed explanation of the non-monotonic aspect of SI-commutativity.