A Brief Review of “Linearizability: A Correctness Condition for Concurrent Objects”

Yi Luo

1 Summary

This paper starts from some straightforward examples to present the intuitions of linearizability, then it gives formal definitions of linearizability along with an informative comparison with the sequential consistency. Linearizability is defined as a correctness condition for concurrent objects. Compared to the sequential consistency we discussed in the last paper, linearizability restricts that each method call should appear to “take effect” instantaneously at some points between the invocation and response. It emphasizes a more restrictive principle that the real-time behavior of operations need to be preserved, which contributes to its locality. Linearizability also permits programmers to specify and reason about concurrent objects using known techniques from the sequential domain. However, no detailed implementations of linearizability are discussed in this paper. In fact, similar to sequential consistency, linearizability also constraints several hardware and compiler optimizations, making it a little bit impractical in multiprocessor systems. [2]

2 Merits

Compared to the sequential consistency, linearizability stands out because of its locality, which is also known as the composability. Sequential consistency only requires that operations should take effect in program order but linearizability requires that the history’s precedence ordering should also be preserved. This makes the sequential consistency not a local property but linearizability a local property. If a concurrent system is based on a nonlocal correctness property, it demands a centralized controller for all shared objects or other additional mechanisms placed upon objects to realize common protocols. However, if the correctness condition is a local property, it turns to be compositional and allows the system to be built in a modular design. That is to say, the result of composing multiple linearizable objects itself is linearizable and these objects can be implemented and verified independently, making linearizability a good way to describe components of large systems.

3 Drawbacks

But linearizability has its deficiencies. Nowadays, most practical systems have its hardware optimizations, for example, the memory reads and writes may typically reordered for better performance. Also, there might be compiling optimizations for particular programming languages. However, both kinds of optimizations may cause violations to the sequential consistency or linearizability, making these two correctness conditions not popular in practical use. For instance, the Java programming language does not guarantee linearizability or sequential consistency when reading or writing on shared objects. [1]

4 Personal Thoughts

I think the use of axioms giving pre- and postconditions to describe the meaning of concurrent object’s operations is a good choice. Each of pre- and postconditions describes the object state before and after the call of methods. With this technique, we don't need to consider details about interactions between different methods. I also find this approach adopted in the artificial intelligence field, researchers utilize pre- and postconditions to define actions/predicates. This technique makes their life easier to conveniently add new operations without changing descriptions of old ones.

References

[1] Nir Shavit Maurice Herlihy. The Art of Multiprocessor Programming. *Chapter 3*, 2012.

[2] Jeannette M. Wing Maurice P. Herlihy. Linearizability: A Correctness Condition for Concurrent Objects. *ACM Transactions on Programming Languages and Systems*, 1990.