Critique for Linearizability: A correctness Condition for Concurrent Objects

Xin Wang

1 Summary

The paper "Linearizability: A correctness Condition for Concurrent Objects" [4] defines the concept of linearizability which is a correctness condition for linearizable objects and compares it with other correctness conditions.

History is a finite sequence which consists of operation method invocations and responses. History is used to model the execution of a concurrent system. A sebsequence of the events in a history H is called a subhistory. If a history is totally ordered and no operation intervals overlap, then the history is considered to be sequential. Otherwise, the history is considered concurrent. The linearization of a history includes two steps. In first step, incomplete invocations with no matching reponse are completed or deleted. In second step, invocations and responses are ordered again to ensure the result is legal and consistent, which means that two ordered operations in sequential history are also ordered operations in linearization.

Linearizability has the properties of locality and non-blocking. Locality means that system is linearizable if and only if all individual objects are linearizable. With this property, concurrent systems can be designed and implemented in modularized method. It also enables linearizable objects be independent from each other. Non-blocking property means that one method is not forced to wait to synchronize with one another. However, blocking can exist in conditions where programmers have planned.

Abstraction function and an interpretation of representation invariant are used to prove the correctness of linearization. A queue example is used for proof of correctness and critical region is considered. With an extended representation, the authors provide us with a continually defined abstraction function which remains the same as the original abstraction function with quiescent object. In this way, proving the correctness of the extended representation natually leads to proof of correctness of the original one.

To sum up, linearizability is beneficial for specifying, implementing and verifying concurrent objects.

2 Positive reactions

Linearizability is an essential and basic correctness requirement for a concurrent implementation in software transaction memory [5]. By showing that in multiprocessor systems, processes can perform the same transaction and behave as if only one process is running, linearizability of transactioin memory can be proved. It is so important that programmers must ensure that after an object has been released, subsequent changes made by other transactions do not violate the linearizability of the former transaction. Moreover, in replicated servers, linearizability provides a nonreplicated servers' illusion for the clients [3].

3 Negative reactions

The verification of linearizability of an object is very important and wide spread, yet to some extent complex in the way that it requires full program verification. To simplify verification of linearizability can improve the performance. One possible simplification method is to use atomicity to develop lightweight checking

tools for related correctness conditions. The proposal is provided in reference [2], yet the implementation has not been realized.

4 Extension proposal

The implementation, analysis and optimization all require linearizability as a standard. Considering the complexity of verification of linearizability, find alternatives such as lightweight checking tools to ensure atomicity maybe helpful. According to the work that compares sequential consistency and linearizability as in [1], in certain circumstances, supporting linearizability may be more costly than supporting sequential consistency. Future research about more detailed comparison can help to make a wise choice between the two.

5 Conclusions

To sum up, this paper proposes a good correctness condition for concurrent objects. Much following work has demonstrated the effectiveness of linearizability and it also inspires some innovative ideas such as making a choice between sequential consistency and linearizability in different conditions for better performance.

References

[1] Hagit Attiya and Jennifer L. Welch. Sequential consistency versus linearizability. *ACM Trans. Comput. Syst.*, 12(2):91–122, May 1994.

[2] Cormac Flanagan and Stephen N Freund. Atomizer: a dynamic atomicity checker for multithreaded programs. *SIGPLAN Not.*, 39(1):256–267, January 2004.

[3] Rachid Guerraoui and André Schiper. Software-based replication for fault tolerance. *Computer*, 30(4):68–74, 1997.

[4] Maurice P Herlihy and Jeannette M Wing. Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 12(3):463–492, 1990.

[5] Nir Shavit and Dan Touitou. Software transactional memory. *Distributed Computing*, 10(2):99–116, 1997.