Brief Announcement: Relaxing Opacity in Pessimistic Transactional Memory

Konrad Siek and Paweł T. Wojciechowski

Institute of Computing Science
Poznań University of Technology
60-965 Poznań, Poland
{konrad.siek,pawel.t.wojciechowski}@cs.put.edu.pl

Since in the Transactional Memory (TM) abstraction transactional code can contain any operation (rather than just reads and writes), greater attention must be paid to the state of shared variables at any given time. Thus strong safety properties are important in TM, such as opacity [2], virtual world consistency [3], or TMS1/2 [1]. They regulate what values can be read, even by transactions that abort. In comparison to these, properties like serializability allow inconsistent views, so they are relatively weak. However, strong properties virtually preclude early release as a technique for optimizing TM. Early release is a mechanism that allows transactions to read from other transactions, even if the latter are still live. This can increase parallelism, and it is useful in high contention (see e.g., [4]). Thus, we introduce last-use opacity, a safety property that relaxes opacity.

Opacity consists of three core guarantees: serializability, preservation of real-time order, and consistency. We concentrate on the latter, which stipulates that non-local read operations (i.e. those that read values written by other transactions than the current one) must only read values from committed or commit-pending transactions. *Last-use opacity* relaxes this consistency criterion to only provide last-use consistency [7] and recoverability. Then, a transaction can read from another live transaction, if the latter will no longer access the variable in question. Plus, transactions must commit or abort in the order in which they access shared variables. These conditions are defined as follows:

Definition 1 (Commit-pending Equivalence). *Transaction T_i in history H is commit-pending–equivalent with respect to variable x if (a) T_i is live, and (b) there is a read or write operation op on x in $H|T_i$, s.t. for any history H_c for which H is a prefix ($H_c = H \cdot H'$) op is the last read or write on x in $H_c|T_i$.*

Definition 2 (Last-use Consistent Operation). *Given a history H, a transaction T_i and a read operation $op_r = \mathrm{r}(x)v$ on variable x returning v in subhistory $H|T_i$, we say op_r is last-use–consistent as follows: (a) If op_r is local then the latest write operation on x preceding op_r writes value v to x; (b) If op_r is non-local then either $v = 0$ or there is a non-local write operation op_w on variable x writing v in $H|T_k$ ($k \neq i$) where T_k is committed, commit-pending, or commit-pending–equivalent with respect to x.*

Definition 3 (Recoverable Last-use Consistency). *History H is recoverable last-use–consistent if (a) every read operation in $H|T_i$, for every transaction*

T_i in H is last-use–consistent, and (b) for every pair of transactions T_i, T_j such that $i \neq j$ and T_j reads from or writes after T_i, then T_i aborts or commits before T_j aborts or commit, and if T_i aborts, then T_j also aborts.

Relaxing consistency necessarily leads to some inconsistent views to be accepted. Hence, while last-use opacity prevents overwriting (releasing x and writing to it afterwards), it does not prevent zombie transactions—ones that view inconsistent state and are forced to abort. This happens if transaction T_i reads from T_j which, for whatever reason, later aborts. Even if T_i eventually aborts, it operates on stale data and, therefore, can behave unexpectedly. However, this can be rendered harmless by, e.g. sandboxing [5], or enforcing invariants.

On the other hand, using last-use opacity yields performance benefits, especially in high contention. In Fig. 1 we compare two variants of the same distributed TM [6]: last-use–opaque LSVA and opaque OSVA. In all benchmarks LSVA is able to process transactions faster, due to its ability to release early.

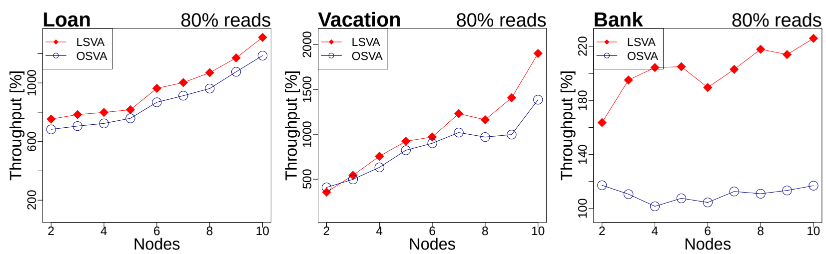

Fig. 1. Percentage improvement relative to a lock-based implementation.

Acknowledgments The project was funded from National Science Centre funds granted by decision No. DEC-2012/06/M/ST6/00463.

References

1. Doherty, S., Groves, L., Luchangco, V., Moir, M.: Towards formally specifying and verifying transactional memory. Formal Aspects of Computing 25 (Sep 2013)
2. Guerraoui, R., Kapałka, M.: On the Correctness of Transactional Memory. In: Proc. PPoPP'08 (Feb 2008)
3. Imbs, D., de Mendivil, J.R., Raynal, M.: On the Consistency Conditions or Transactional Memories. Tech. Rep. 1917, IRISA (Dec 2008)
4. Ramadan, H.E., Roy, I., Herlihy, M., Witchel, E.: Committing Conflicting Transactions in an STM. In: Proc. PPoPP'09 (Feb 2009)
5. Scott, M.: Transactional Semantics with Zombies. In: Proc. WTTM'14 (Jul 2014)
6. Siek, K., Wojciechowski, P.T.: Atomic RMI: a Distributed Transactional Memory Framework. In: Proc. HLPP'14 (Jul 2014)
7. Siek, K., Wojciechowski, P.T.: Zen and the Art of Concurrency Control: An Exploration of TM Safety Property Space with Early Release in Mind. In: Proc. WTTM'14 (Jul 2014)