

# Optimizing Consistency Checking for Memory-Intensive Transactions

Justin E. Gottschlich

Daniel A. Connors

Department of Electrical and Computer Engineering, University of Colorado at Boulder  
 {gottschr, dconnors}@colorado.edu

## ABSTRACT

Transactional memory (TM) reduces parallel programming complexity while maintaining multi-threaded performance benefits. Consistency checking, the way memory conflicts are found in TM, is critical to performance and is executed through either *validation* or *invalidation*. We present a unique invalidation algorithm which boasts high transactional throughput for memory-intensive transactions.

**Categories and Subject Descriptors:** D.1.3 [Concurrent Programming]: Parallel Programming.

**General Terms:** Algorithms, Performance, Theory.

**Keywords:** Consistency Checking, Transactional Memory.

## 1. CONSISTENCY CHECKING

*Validation* ensures consistency by comparing transactions' read and write sets against global memory using versioning of memory to verify consistency [1]. Using worst-case analysis, given a non-unique series of  $M$  committing transactions, and  $w_i$  write and  $r_i$  read set sizes of the  $i$ th committing transaction, the commit-time operations needed for validation are:

$$c(M) = \sum_{i=1}^M w_i + r_i.$$

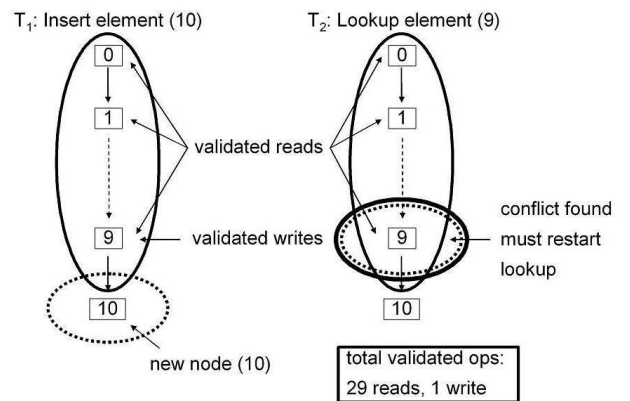
Our *invalidation* algorithm ensures consistency by invalidation, committing transactions search in-flight transactions for conflicts, in conjunction with thread-level locking and search-optimized containers. Using worst-case analysis, given a non-unique series of  $M$  committing transactions,  $w_i$  write set size of the  $i$ th committing transaction,  $F_i$  in-flight transactions during the  $i$ th transaction's commit, and  $w_j$  write and  $r_j$  read set sizes of the  $j$ th in-flight transaction, the commit-time operations needed for our algorithm are:

$$c(M) = \sum_{i=1}^M \left( \sum_{j=1}^{F_i} w_i (\lceil \log_2(w_j) \rceil + \lceil \log_2(r_j) \rceil) \right)$$

A comparison of validation and our invalidation algorithm is shown in Figure 1, where transactions  $T_1$  (insert) and  $T_2$  (lookup) operate on a  $10^1$  sized linked list. Our invalidation algorithm outperforms validation by  $7.5x$  (30/4), yet, the performance differential between the consistency

checking models are not constant. As transactional memory increases, the rate of change widens superlinearly, yielding a  $300x$  performance difference for a  $10^3$  sized list and a  $17,650x$  performance difference for a  $10^5$  sized list. These savings in conjunction with reduced atomic operational overhead achieve high performance for memory-intensive transactions (full paper: <http://rogue.colorado.edu/draco/>).

### Validation – Insert Commits First



### Invalidation – Insert Commits First

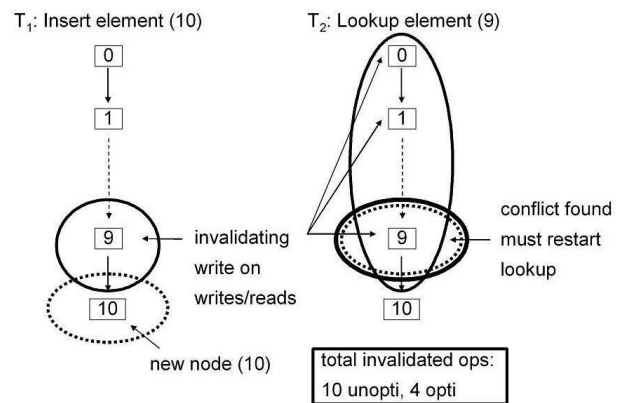


Figure 1: Consistency Checking Comparison.

## 2. REFERENCES

- [1] J. R. Larus and R. Rajwar. *Transactional Memory*. Morgan & Claypool, 2006.