Brief Announcement: From Causal to z-Linearizable Transactional Memory

Torvald Riegel Christof Fetzer
Dresden University of Technology, Germany

Heiko Sturzrehm* Pascal Felber
University of Neuchâtel, Switzerland

ABSTRACT

The current generation of time-based transactional memories (TMs) has the advantage of being simple and efficient, and providing strong linearizability semantics. Linearizability matches well the goal of TM to simplify the design and implementation of concurrent applications. However, long transactions can have a much lower likelihood of committing than smaller transactions because of the strict ordering constraints imposed by linearizability. We investigate the use of weaker semantics for TM and introduce a new consistency criterion that we call z-linearizability. By combining properties of linearizability and serializability, z-linearizability provides a good trade-off between strong semantics and good practical performance even for long transactions.

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

General Terms: Algorithms, theory.

Keywords: Transactional memory, causal serializability, linearizability.

1. CONTEXT AND PROBLEM

Transactional memory (TM) has been proposed as a lightweight mechanism to synchronize threads. It alleviates many of the problems associated with locking, offering the benefits of transactions without incurring the overhead of a database. It makes memory, which is shared by threads, act in a transactional way like a database. The main goal is to simplify the development of concurrent applications, which are becoming more widespread because of the increasing shift to multicore processors and multiprocessor systems.

Many TM implementations provide strong semantics, i.e., linearizability. While most of the implementations are efficient and scalable, linearizability restricts the concurrency one can achieve, in particular, with long transactions. Indeed, as most TMs rely on the "first committer wins" rule, a commit by any concurrent short transaction may force a long transaction to abort. Furthermore, the fact that linearizability preserves the real time ordering of transactions forbids some schedules which might be valid for the application (e.g., serializable schedules) and that would let the long transaction commit. Consider for instance the three transactions in Figure 1 executing on three different threads. Filled circles correspond to write operations (i.e., new versions that will be visible to other transactions at commit time) while empty circles represent read operations. Transaction demarcation is represented by the gray triangles. Linearizability imposes an ordering of T_1 before T_2 , which prevents long transaction T_L from committing, even though there is a valid serialization. Linearizability may thus limit concurrency because it imposes a total order on transactions that is consistent with real time even when they access disjoint object sets and would not otherwise need to be ordered.

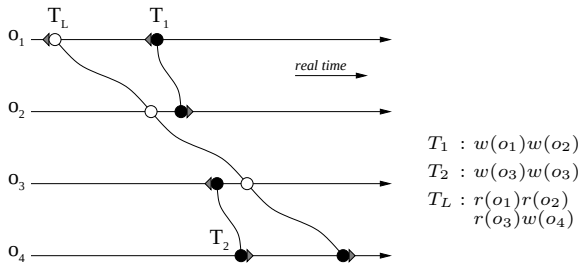

Figure 1: Linearizability schedules T_1 before T_2, which forces long transaction T_L to abort (each transaction executes in a separate thread). Serializability would allow all three transactions to commit.

2. A VECTOR TIME BASE

A reasonable way to improve the chances for long transactions to commit is to use weaker semantics to improve the concurrency of transactions. Some weaker types of semantics can be implemented with the help of vector clocks, which have the potential to increase the concurrency but also produce higher runtime overheads.

Vector clocks can be used as a time base for implementing a TM. They offer two attractive features. First, unlike time bases implemented with a single shared counter, vector clocks do not suffer from contention on the time base because each thread can have its own component in a vector clock and timestamps are exchanged between threads only when they access shared objects. This loose synchronization can be beneficial when inter-processor communication delays are not negligible (e.g., in distributed settings) or contention on the shared counter is high.

Second, unlike Lamport's logical clocks, vector timestamps

*This research was partially supported by the by the Swiss National Science Foundation under grant number 5005-67322 (NCCR-MICS).

PODC'07, August 12–15, 2007, Portland, Oregon, USA.
ACM 978-1-59593-616-5/07/0008.

allow us to identify transactions that are not causally related and do not conflict, for which it is not necessary to determine a total ordering at commit time.

Consider again Figure 1. In a time-based TM that uses a single clock (e.g., a shared counter), transaction T_1 commits before, and is thus ordered before, T_2. Hence, T_L must abort because it reads versions of o_1 and o_2 that are not valid anymore at the time T_L commits. This ordering of T_1 and T_2 is not necessary because the two transactions access disjoint sets of objects. There is a valid serialization $T_2 \rightarrow T_L \rightarrow T_1$ but one cannot take advantage of it.

Moving from single clocks to vector timestamps allow us to easily implement *causal serializability*, a consistency criterion weaker than serializability but stronger than causal consistency. Informally, causal consistency allows each processor to have its own sequential view of the execution as long as the individual views preserve the causality relation. Causal serializability adds the additional constraint that all transactions that update the same object must be perceived in the same order by all processors. In the example of Figure 1, causal serializability allows all transactions to commit. Adapting our LSA algorithm [1] to support causal serializability using vector timestamps is straightforward. By using *plausible clocks* instead of vector clocks, one can further keep the runtime overhead reasonably low.

Adding support for serializability would require two major changes to the algorithm: reads must be visible, and we need to construct a partial precedence graph to guarantee that schedules are serializable. These extensions introduce significant runtime overhead.

3. Z-LINEARIZABILITY

We propose a novel, pragmatic approach to increase the chances that long transactions can commit. The problem is that in time-based TMs, long transactions that access a large number of objects have typically a higher likelihood of interfering with other transactions (e.g., T_{L1} interferes with T_1 and T_2 in Figure 2). Moreover, when a short transaction (e.g., T_5 or T_6) updates objects that are read by a long transaction and the short transaction commits first, this will result in the abort of the long transaction.

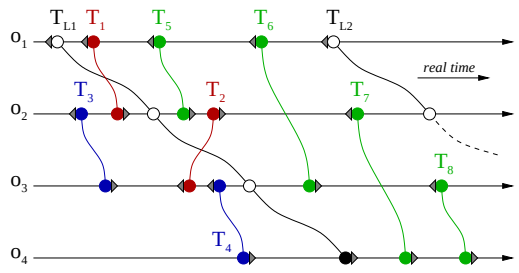

Figure 2: Long transactions have typically a higher chance of interfering with other transactions. Since the first committer wins, a commit by any of the short transactions T_5 and T_6 will result in the abort of T_{L1}.

Consider the scenario depicted in Figure 2. Long transaction T_{L1} accesses all objects (e.g., all accounts of a bank) while the remaining transactions only access a small number of objects (e.g., two accounts in case of a simple transfer). To permit T_{L1} to commit and to guarantee serializability, the system would need to abort at least transactions T_1 and T_2 because neither T_1 and T_{L1} nor T_2 and T_{L1} can be ordered. To permit T_{L1} to commit and to guarantee linearizability, one would additionally need to abort T_4 or T_5 because linearizability requires T_5 to be ordered before T_4 but T_{L1} must be ordered after T_4 and before T_5.

A pragmatic approach is to slightly weaken linearizability during a long transaction. We propose to permit an ordering of short transactions that can violate the real time ordering of transactions but only while a long transaction is executing. To explain the principle of z-linearizability, consider long transactions that access many objects (e.g., they calculate the sum of all accounts). Our basic assumption is that such long transactions are quite infrequent in comparison to short transactions. Hence, long transactions partition short transactions into different "time zones". We want to ensure that all transactions in the individual zones are linearizable and all transactions are serializable. More precisely, the semantics that we want to enforce for z-linearizability are that (1) the set of long transactions is linearizable, (2) the set of short transactions is linearizable, (3) the set of all transactions is serializable, and (4) the serialization order observes the sequential execution ordering of the individual threads.

This is a weakening of linearizability in the sense that we permit some short transactions to violate linearizability while a long transaction is in progress. Property (4) states that a thread cannot cross an active long transaction "backwards", e.g., in Figure 2 a thread t could execute T_3 and then T_5, but not T_5 and then T_4 because the latter would cross the path of T_{L1} backwards.

In our experiments, we have observed an increase in the throughput of long transactions using z-linearizability as compared to our LSA implementation [1] that provides linearizability.

4. CONCLUSION

The linearizability semantics supported by transactional memories are simple to reason about and allow for efficient and scalable implementations. Yet, enforcing linearizability also has the negative effect of restricting concurrency under some common workloads, notably when long transactions compete with many short transactions. The use of semantics weaker than linearizability can help increase the throughput of long transactions. In particular, *z-linearizability* offers a promising trade-off between strong semantics and good performance. A full version of the paper appears as [2].

5. REFERENCES

[1] Torvald Riegel, Pascal Felber, and Christof Fetzer. A Lazy Snapshot Algorithm with Eager Validation. In *20th International Symposium on Distributed Computing (DISC)*, September 2006.

[2] Torvald Riegel, Heiko Sturzrehm, Pascal Felber, and Christof Fetzer. From causal to z-linearizable transactional memory. Technical Report RR-I-07-06.1, University of Neuchatel, 2007.