

Transactional Memory, linking Theory and Practice

Srivatsan Ravi
TU Berlin
Deutsche-Telekom Labs
ravi@net.t-labs.tu-berlin.de



Vincent Gramoli
EPFL
University of Neuchâtel
vincent.gramoli@epfl.ch



Victor Luchangco
Sun Labs, Oracle
victor.luchangco@oracle.com



Abstract

Transactional memory appears promising for democratizing concurrent programming. Its potential lies in producing code that is extensible as atomicity is preserved under composition of transactions. This new abstraction raises several challenges such as the compliance of transactions with alternative synchronization techniques of legacy code and memory consistency models that favor transactional programming. The objective of the Second Workshop on the Theory of Transactional Memory was to discuss new theoretical challenges and recent achievements in the context of transactional computing. We report on some of the issues raised during this workshop.

1 Introduction

The Second Workshop on the Theory of Transactional Memory, organized by Vincent Gramoli and Victor Luchangco, was held on 16 Sep 2010, in conjunction with the DISC 2010 in Cambridge, Massachusetts, USA. It consisted of a panel session on high-level issues about research on the theory of transactional memory (TM), and three technical sessions consisting of talks grouped along common themes. The main goal of the workshop was to foster discussion about the issues raised in these sessions. To achieve this goal, talks were kept brief, and over half the time was spent in vigorous discussion among the speakers and the audience.

The tone of the workshop was set by the panel discussion moderated by Michael Scott, which featured brief talks by Ali-Reza Adl-Tabatabai, Rachid Guerraoui and Mark Moir. The speakers were encouraged to present their points in a provocative manner to stimulate discussion, and a few themes emerged.

Scott emphasized in his introduction that the attraction of TM lay in great part on its perceived simplicity, and that a theory of TM must retain that essential simplicity. While Guerraoui agreed with Scott, he argued that for TM to become popular, it must be acceptable to expert programmers who require more control. Thus, more sophisticated models are needed, which requires that we respect users and trust them to tune the model appropriate to the level of programming.

TM theory should also encompass the interaction of TM with other mechanisms, such as locks and exceptions. Adl-Tabatabai noted that in a recent study with Victor Pankratius [11], students using TM in a project also used other means of synchronization, so a TM theory must specify the behavior of programs with

transactions and other synchronization primitives. Models for TM are also important for designing tools to aid in writing programs that use TM. Currently, there is not even a rough performance model to help guide the choice of programming approaches for a programmer trying to use TM. In addition to a performance model, TM-aware debuggers and profilers are necessary if TM is to be widely adopted. Adl-Tabatabai noted that the lack of such tools was a source of frustration for the students in their study.

One area of controversy among the speakers (and the audience) was the degree to which TM theory should consider practical issues. Guerraoui argued that for TM theory to mature, we must treat it as a “first-class citizen” and grant it the same status as other theoretical topics in the distributed systems community. In contrast, Moir argued that although an abstract model does elide some relevant details of the underlying hardware, it should not impose artificial restrictions on implementations, and that encouraging research on TM models with artificial restrictions saps resources from more important research. One cited example of such a restriction was strict disjoint-access-parallelism, a property that is violated by the most efficient conflict-free algorithms as observed at the former workshop [8]. Unfortunately, conclusions drawn from research on unrealistic models could be, and have been, interpreted wrongly to apply to other settings because the assumptions are not always spelled out clearly, a problem that is exacerbated by the differing uses of terminology between theoreticians, systems researchers and implementors.

The discussion from the invited speakers provided the impetus for subsequent discussion of workshop talks covering topics like the question of composition and concurrency, automated proofs and verification of TM, relaxed memory and message passing models.

2 Composition and concurrency

Composition is an appealing feature of transactions which make them reusable. Although transactional composition guarantees intuitively that the semantics of transactions is preserved despite their concurrency [10], the properties of this semantics are unclear and it is crucial to identify them. For instance, livelock-freedom cannot be preserved under composition. Michael Spear presented a barrier counter-example, derived from [14], involving n threads waiting for each other by first incrementing a counter in a transaction t_1 , and then reading the counter value and retrying until the counter value equals n in a subsequent transaction t_2 . This program terminates provided that one thread, while executing t_2 , sees that the counter value has been changed by another thread incrementing it while running t_1 . The cooperation between transaction t_1 of one thread and transaction t_2 of another is impossible if one tries to encapsulate them into a composite transaction t , using a simple closed-nesting technique (a technique discussed by Sathya Peri and Vidyasankar [12]). More specifically, the inherent isolation of t prevents threads from seeing the concurrent counter increments, forcing them to retry t_2 forever.

Composition, as defined by Mihai Letia in his work with Gramoli and Guerraoui [9], is the ability to encapsulate two transactional operations π_1 and π_2 into a transactional operation $\pi_3 = \pi_1 \circ \pi_2$ that preserves their atomicity and deadlock-freedom—but not livelock-freedom. Livelock-freedom cannot even be ensured in presence of contention if transactions are naively scheduled—this is the task of the contention manager to schedule transactions adequately to avoid livelocks. On the topic of contention management, Gokarna Sharma presented his work with Busch on the tight bounds of a randomized algorithm for workloads in which update and read-only transactions are balanced [13]. With any contention manager, relaxed transactions, which enable greater concurrency, tend to violate this composition definition. For example, the elastic model must comprise both elastic and regular transactions to ensure composition [6]. Adam Morrison presented his work on *view transactions* with Afek and Tzafrir [1]. In contrast with elastic transactions, such relaxed transactions use view pointers to indicate which memory locations must remain consistent for

the transaction to commit. Two view transactions that insert and delete can be encapsulated into a move transaction if the programmer extends the scope of the original view pointers to the context of the outermost move transaction. This prevents a programmer from reusing the insert and delete code without modifying it, hence violating composition as defined in [9]. This observation outlines a potential tradeoff between high concurrency and composition of transactions.

3 Verifying TM

Verifying concurrent programs is notoriously difficult because the number of possible interleavings of operations of different threads is exponential in the number of operations. TM can make this easier because a transaction can be treated as a single atomic operation, greatly reducing the number of interleavings that must be considered. However, even verifying the safety of TM algorithms is a complex task due to some (potentially) unbounded parameters like transaction delays and behavior of aborted transactions. Indeed, even specifying what it means for a TM to be correct is not entirely settled: there are several correctness conditions that differ in various ways.

Justin Gottschlich presented his work with Siek and Vachharajani on verifying their InvalSTM [7], using I/O automaton to model the STM and conflict graphs to specify the correctness condition. Their model currently treats commit as an atomic event, and after they complete the proof for that model, they intend to refine it to consider the actual commit procedure.

Victor Luchangco presented work with Doherty and Moir on developing completely formal (i.e., machine-checkable) specifications and verifications of transactional memory, also using I/O automaton, in this case expressed using the PVS language so that the proofs can be checked by the PVS prover [5]. They formalized two specifications for TM, one that captures the guarantees of TM in a very abstract and general way and one that is closer to an implementation, modeling read and write sets explicitly, and proved that the latter implements the former. Next they intend to write formal models of TM algorithms such as *NOrec* [3] and *TL2* [4], and prove that these implement the specifications.

An alternative to having a TM system that guarantees the atomicity of a transaction regardless of the operations done within the transaction is to support “coarse-grained transactions”, which capture methods on an abstract data structure that appear to be atomic. Coarse-grained transactions provide potential for improved performance but they increase the programmer’s burden because conflicts among these transactions must be correctly specified. Trek Palmer presented work with Eliot Moss introducing the *ACCLAM* language to specify the application-level abstractions of the concurrent program. The resulting specification helps in verifying automatically that low-level conflicts do not trigger false conflicts at the application level and that, upon abort, the involved abstractions are rolled back to a consistent state even though low-level operations are not compensated.

4 Weakening semantics

Several models have been proposed to relax the semantics of transactions to allow more efficient implementations, and also to make explicit the guarantees between transactional and nontransactional operations. Lock-based semantics are inadequate because they do not specify, for example, the behavior of “zombie” transactions that are doomed to abort due to conflicts, but continue to run for some time before the conflict is discovered. These doomed transactions are rescued by TwilightSTM [2], a software TM presented by Annette Bieniusa that aims at relaxing transaction semantics and uses an enriched interface to accept irrevocable twilight regions inside transactions. Luke Dalessandro also suggested to enrich the TM interface in his work with Scott, but for exposing speculation independently from atomicity.

Relaxation issues related to the semantics of nontransactional stores and more generally to the specification of AMD's Advanced Synchronization Facility were discussed by Sean White in a joint work with Spear, and by Stephen Diestelhorst in a joint work with Hohmuth and Polhack. Should the level of consistency be adjustable to resolve contention points and cater to application specific needs? What are the alternatives to "abort on inconsistency"? Can the programmer be oblivious to speculation and rollback? Ideally, the semantics should be specified and formalized in a way that encompasses both hardware and software TM implementations, with differing levels of detail depending on the needs of the users and implementors. Hierarchical specifications could be useful for this.

5 Extensions to message passing

Moving from multicore to manycore systems requires reconsideration of our computational models and in particular, communication through message passing rather than shared memory.

Junwhan Kim and Bo Zhang argued in their work with Ravindran that a distributed message-passing system exporting a TM interface may help avoiding inherent synchronization issues of existing RPC-based distributed control-flow programming models. They investigated the support and progress of a data-flow distributed TM where a non-replicated object is moved from node to node.

Alternatively, Eric Koskinen proposed with Herlihy to treat the TM abstraction as a fully-replicated distributed system that allows every thread to optimistically apply single-operation transactions on local copies of transactional objects, and achieves consistency across the threads through an atomic broadcast protocol.

6 Conclusion

To conclude, two main themes emerged repeatedly throughout the workshop. First, it was argued that TM should provide greater flexibility and control to the programmer, and various extensions to do so were proposed. This control could enhance concurrency by letting users give hints to the TM, to exploit best-effort hardware TM, to tolerate irrevocable actions within transactions, or to help differentiate nontransactional stores, all while ensuring composition. Second, greater precision and rigor is necessary to specify the guarantees of TM and the behavior of TM implementations. Several people presented work that advanced the current state, but more work is necessary, particularly to handle the extensions to TM that are contemplated. Although additional control might benefit advanced programmers, many expressed the need to preserve the appealing simplicity of the transactional programming paradigm to facilitate its adoption. Thus, one challenge is to produce a model for TM that satisfies both advanced programmers, who can spend a lot of time and effort to achieve high performance, and novice programmers, who want off-the-shelf solutions that are easy to understand and can be applied to aid in writing correct, efficient and scalable concurrent programs.

Acknowledgements

We wish to thank the contributors to the workshop: Ali-Reza Adl-Tabatabai, Yehuda Afek, Annette Beniussa, Konstantin Busch, Luke Dalessandro, Stephan Diestelhorst, Simon Doherty, Justin Gottschlich, Rachid Guerraoui, Maurice Herlihy, Michael Hohmuth, Junwhan Kim, Eric Koskinen, Ranjeet Kumar, Mihai Letia, Mark Moir, Adam Morrison, Eliot Moss, Trek Palmer, Sathya Peri, Martin Pohlack, Binoy Ravindran, Michael L. Scott, Gokarna Sharma, Jeremy G. Siek, Michael Spear, Moran Tzafrir, Manish Vachharajani, Sean White and Bo Zhang. We are grateful to Petr Kuznetsov for comments on earlier versions of this report.

References

- [1] Y. Afek, A. Morrison, and M. Tzafrir. View transactions: Transactional model with relaxed consistency checks. In *PODC '10: Proceedings of the 29th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, 2010.
- [2] A. Bieniusa, A. Middelkoop, and P. Thiemann. Brief announcement: Actions in the twilight - concurrent irrevocable transactions and inconsistency repair. In *PODC '10: Proceedings of the 29th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 71–72, New York, NY, USA, 2010. ACM.
- [3] L. Dalessandro, M. F. Spear, and M. L. Scott. NOrec: streamlining stm by abolishing ownership records. In *PPoPP '10: Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 67–78, 2010.
- [4] D. Dice, O. Shalev, and N. Shavit. Transactional locking II. In *DISC '06: Proceedings of the 20th International Symposium on Distributed Computing*, pages 194–208, 2006.
- [5] S. Doherty, L. Groves, V. Luchangco, and M. Moir. Towards formally specifying and verifying transactional memory. *Electronic Notes in Theoretical Computer Science*, 259:245 – 261, 2009. Proceedings of the 14th BCS-FACS Refinement Workshop (REFINE 2009).
- [6] P. Felber, V. Gramoli, and R. Guerraoui. Elastic transactions. In *DISC '09: Proceedings of the 23rd International Symposium on Distributed Computing*, volume 5805 of *LNCS*, pages 93–107, sep 2009.
- [7] J. E. Gottschlich, M. Vachharajani, and J. G. Siek. An efficient software transactional memory using commit-time invalidation. In *CGO '10: Proceedings of the 8th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, pages 101–110, New York, NY, USA, 2010. ACM.
- [8] V. Gramoli. What theory for transactional memory? *SIGACT News*, 40(4):79–81, 2009.
- [9] V. Gramoli, R. Guerraoui, and M. Letia. The many faces of transactional software composition. Technical Report EPFL-REPORT-150654, EPFL, 2010.
- [10] T. Harris, S. Marlow, S. Peyton-Jones, and M. Herlihy. Composable memory transactions. In *PPoPP '05: Proceedings of the 10th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 48–60, New York, NY, USA, 2005. ACM.
- [11] V. Pankratius, A.-R. Adl-Tabatabai, and F. Otto. Does transactional memory keep its promises? Results from an Empirical Study. Technical Report 2009-12 IPD, University of Karlsruhe, Germany, September 2009.
- [12] S. Peri and K. Vidyasankar. Correctness of concurrent executions of closed nested transactions in transactional memory systems. In *ICDCN'11: Proceedings of the 12th International Conference on Distributed Computing and Networking*, 2011. to appear.
- [13] G. Sharma and C. Busch. A competitive analysis for balanced transactional memory workloads. In *Proceedings of the 14th International Conference On Principles Of Distributed Systems*, 2010.
- [14] Y. Smaragdakis, A. Kay, R. Behrends, and M. Young. Transactions with isolation and cooperation. In *OOPSLA '07: Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems and Applications*, pages 191–210, New York, NY, USA, 2007. ACM.