Transactional memory-From theory to practice

Srivatsan Ravi
TU Berlin
Deutsche-Telekom Labs

Vincent Gramoli
EPFL

Victor Luchangco
Sun Labs, Oracle

March 3, 2011

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 conjuction 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 [12], 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 rough 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 statu quo as other theoretical topics in the distributed systems community. In contrast, Moir argued that although an abstract model does gloss over some relevant details of the underlying hardware, it should not impose artificial restrictions on implementations. Moir was of the opinion that encouraging research on TM models with artificial restrictions saps resources from more important research. One cited example was a model satisfying strict disjoint-access-parallelism, which may paradoxally provoke higher time complexity as noticed at the former workshop edition [8]. Unfortunately, conclusions drawn from research on unrealistic models could be interpreted wrongly to apply to other settings because the assumptions are not always spelled out clearly in a way accessible to nontheoreticians, 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 the 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, similar to the one of [15], 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 made impossible if one tries to encapsulate them into a composition transaction t, using a simple closed-nesting technique (a technique discussed by Sathya Peri and Vidyasankar [13]). More specifically, the inherent isolation of t will prevent threads from seeing the concurrent counter increments and will force them to retry t_2 forever.

Composition, as precisely 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 [14]. Interestingly and with any contention manager, relaxed transactions, which enhance 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 extend the scope of the original view pointers to the context of the outermost move transaction. This prevents the third-party 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 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. (Because algorithms and specifications are modeled as automata, the proofs are hierarchical, and it is sufficient to prove that the algorithms implement the lower-level specification.)

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

Lock-based STM's adopt weak atomicity techniques proposed like disjoint lock atomicity (*DLA*), asymmetric lock atomicity (*ALA*). This is in contrast to have transactions behave as if they were protected by a *single global lock (SGLA)*. While weakening the semantics can alleviate the significant costs incurred by SGLA to provide safety guarantees, it does not remove them [11]. The semantics of SGLA does not specify the behavior of (zombie) transactions that are doomed to abort because of synchronization conflicts, but continue to run for some duration 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 non-transactional stores and more generally to the specification of the AMD's Advanced Synchronization Facility were also 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, hierarchical semantics and specifications need to formalized to keep in mind the different requirements of hardware/software TM implementations while abstracting the programmer from low-level details.

5 Extensions to message passing

Moving from the multicore to the manycore era implies to reconsider our computational models and in particular, communication through message passing rather than shared memory exclusively.

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, the general tendency is now to provide the programmer with more and more control over the Transactional Memory. This control could enhance concurrency by letting the user expose semantic hints to the TM, it could also tolerate irrevocable actions inside transactions, or even help differentiating non-transactional stores, while still ensuring concurrent code composition. Although additional control would certainly benefit to advanced programmers, it is necessary to preserve the appealing simplicity of the programming paradigm, to guarantee its wider adoption. The real challenge here is to revise the TM model so that it satisfies not only advanced programmers who accept to spend time to achieve high performance but also novice programmers that want an off-the-shelf solution to program rapidly various concurrent applications. The intricacy of such TM models will raise further questions in terms of correctness and automated verification.

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. 3

[2] A. Bieniusa, A. Middelkoop, and P. Thiemann. Actions in the twilight - concurrent irrevocable transactions and inconsistency repair. In *PODC '10: Proceeding of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 71–72, New York, NY, USA, 2010. ACM. 4

[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, New York, NY, USA, 2010. ACM. 4

[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. 4

[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. REFINE '09: Proceedings of the 14th BCS-FACS Refinement Workshop. 3

[6] P. Felber, V. Gramoli, and R. Guerraoui. Elastic transactions. In *DISC '09: Proceedings of the 23rd International Symposum on Distributed Computing*, volume 5805 of *LNCS*, pages 93–107, sep 2009. 3

[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. 3

[8] V. Gramoli. What theory for transactional memory? *SIGACT News*, 40(4):79–81, 2009. 2

[9] V. Gramoli, R. Guerraoui, and M. Letia. The many faces of transactional software composition. Technical Report EPFL-REPORT-150654, EPFL, 2010. 3

[10] T. Harris, S. Marlow, S. Peyton-Jones, and M. Herlihy. Composable memory transactions. In *PPoPP '05: Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 48–60, New York, NY, USA, 2005. ACM. 2

[11] V. Menon, S. Balensiefer, T. Shpeisman, A.-R. Adl-Tabatabai, R. L. Hudson, B. Saha, and A. Welc. Single global lock semantics in a weakly atomic STM. *SIGPLAN Not.*, 43(5):15–26, 2008. 4

[12] 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. 2

[13] 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. 3

[14] G. Sharma and C. Busch. A competitive analysis for balanced transactional memory workloads. In *OPODIS '10: Proceedings of the 14th International Conference On Principles Of Distributed Systems*, 2010. to appear. 3

[15] 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. 2