A Formalism for Distributed Transactional Memory Models

Konrad Siek

October 10, 2014

COST Action: IC1001

STSM title: A Formalism for Distributed Transactional Memory Models

STSM dates: from 07-09-2014 to 14-09-2014

Host: Prof. Rachid Guerraoui, École Polytechnique Fédérale de Lausanne, rachid.guerraoui@epfl.ch

Participant: Konrad Siek, Poznań University of Technology, konrad.siek@cs.put.edu.pl

Reference: ECOST-STSM-IC1001-070914-050072

1 Purpose of the STSM

Deterministic scheduling problems are classified using a system of notation [5, 2] that facilitates presentation and discussion of scheduling problems. Each deterministic scheduling problem can be described using a triple $\alpha|\beta|\gamma$, where α describes the processor environment of the problem, β describes the characteristics of the tasks and resources involved in the problem and γ describes the performance measure. Hence, for instance, a problem of *scheduling non-preemptable, arbitrary-length tasks with precedence constraints on an arbitrary number of identical processors with the intent of minimizing schedule length* can be denoted simply as described as $P|prec|C_{max}$. The notation is succinct and unambiguous, and therefore aids communication. In addition, it allows to describe the entire domain of scheduling problems; and if all scheduling problems are known *a priori*, it is easy to find ones which lack solutions.

In a similar vein to the system of classification of scheduling problems, a framework called ACTA [3] was introduced by the researchers in the field of database transactions. The framework attempts to unify and describe the spectrum of transactional models in terms of a set of characteristics: visibility, consistency, recovery, and permanence. These are captured by specifying what type of effects transactions can have on each other and on shared objects. I.e., with respect to effects on other transactions, transactions can be commit-dependent or abort-dependent upon one another (a transaction is required to wait for a previous transaction to commit or abort before it can do so too). This can be further qualified by adding requirements, e.g., an exclusively-abort-dependent transaction must not develop more than one abort dependency, and transitive-commit-dependent transaction is dependent on the committed transactions of its parent. Furthermore, specific formal conditions for transaction aborts can be expressed within the ACTA notation. Transactions' effects on shared objects are given in terms of a view set, a set of all objects potentially accessible to a transaction, and an access set, the set of objects actually accessed by the transaction. When a transaction commits, the values of all objects in its access set become persistent. This system also allows to define delegation, by moving objects from one transaction's access set to another's. Limiting what can be within a transaction's view and access sets and under what conditions allows comprehensively to specify transactional behavior. The authors of [3] suggest that ACTA may be used to specify the behavior and structure of any transactional model and show this by specifying a the nested, split, joint, and cooperative transactional models. ACTA allowed the researchers to find a new model by combining the nested and split transactional models. The framework also allows to understand the nature of the models better.

However, there is a distinct lack of any tool that would allow formally and precisely (but also briefly) to describe transactional memory (TM) models. This is especially true for distributed TM, which is a nebulous term describing very different systems (e.g., both replicated and non-replicated ones). Therefore, the goal of this STSM is to look into the properties of TM systems and attempt to develop a formal and comprehensive taxonomy that could be used as a tool for their classification.

2 Description of the work carried out during the STSM

Due to the brevity of the STSM, the researchers focused on trying to identify the key characteristics of TM systems that need to be taken into account when creating a comprehensive taxonomy. In this way, the researchers intended to create a basis for future work on the formalization of the taxonomy. In particular, the STSM allowed to take a basic survey of the field, however, this survey needs to be broadened to ensure that the results are comprehensive.

3 Description of the main results obtained

As preliminary results from the STSM, we identified some key characteristics that describe TM models.

System Model One of the most fundamental differentiators among TMs is the system model in which they are used. TMs are used in a variety of system models including non-distributed multicore systems and various types distributed systems. Non-distributed TMs operate in a single machine environment—this category includes most TM systems, e.g. [4]. Distributed TMs can be either replicated, partially-replicated, or non-replicated. A replicated TM operates on a network of nodes, where each node has the same set of shared resources that are kept consistent (so from a transaction's point of view, all nodes are the same, see e.g., [7, 8]). A non-replicated TM operates on a network of nodes, each of which has a different set of unique resources (so a transaction explicitly points to nodes it wishes to access resources from, see e.g., [10, 11, 12]). A partially replicated TM is a hybrid solution, where all resources are replicated, but only on a subset of the network each. Furthermore, non-distributed TMs can either allow transactions to access the resources by sending a request to them (this is the control flow model) or by moving the object to the host of the transaction (data flow model). In the end, the system model can be briefly characterized as (where `local` is the default):

$$SM = \left\{ \texttt{local}, \texttt{repl}, \texttt{prepl}, \texttt{cflow}, \texttt{dflow} \right\}$$

Concurrency Control Method TMs tend to use the optimistic approach to concurrency control, where transactions attempt to access shared resources without synchronization and abort whenever two or more transactions conflict by accessing the same resource at the same time. This is the more prevalent approach used e.g., in [4, 10, 12]. The other approach is pessimistic, which involves preventing conflicts before they happen (see e.g., [1, 9, 11]. Hence (with `opt` is the default):

$$CC = \left\{ \texttt{opt}, \texttt{pes} \right\}$$

Nesting Some TMs allow transactions to be specified inside other transactions. Flat nesting specifies that operations in nested transactions simply become operations of the parent transaction. Close nesting allows for the children transactions to execute as transactions, and if they abort, they themselves restart, but do not cause the parent to abort. Linear nesting transactions are close nesting transactions with an additional requirement that at most one child transaction is executed per parent at any time. Open nested transactions assumes that children run at a different level of abstraction than their parents, so e.g., the parent can commit even if its children abort.

$$TM = \left\{ \texttt{none}, \texttt{flat}, \texttt{close}, \texttt{linear}, \texttt{open} \right\}$$

API TMs can allow or disallow various operation types to be performed within. Transactions can distinguish between reads and writes (assumed by default), or treat all operations uniformly. They an also allow the user to perform manual aborts. Finally, TMs can allow or forbid irrevocable operations within transactions.

$$API = \{\ \texttt{op}, \texttt{r/w}, \texttt{mabort}, \texttt{irr}\ \}$$

Update Management TMs handle executing writes and updating the shared resources in one of two ways. Some TM systems defer updating shared resources until they commit (this can be considered the default), while others update *in-place*, as soon as the write occurs.

$$DU = \{\ \texttt{inplace}, \texttt{oncommit}\ \}$$

Safety and Progress The safety and progress guarantees of a given TM may also be specified.

$$\begin{array}{rcl} \textit{Safety} & = & \{\ \texttt{opacity}, \texttt{VWC}, \texttt{TMS1}, \ldots\ \} \\ \textit{Progress} & = & \{\ \texttt{lockfree}, \texttt{waitfree}, \ldots\ \} \end{array}$$

Notation The characteristics above can be grouped and presented using a notation similar to those known in scheduling, e.g., as a triple:

$$SM \mid CC \times TM \times UM \times API \mid \textit{Safety} \times \textit{Progress}$$

Using this notation a system like Atomic RMI [11] can be described as:

$$\texttt{cflow} \mid \texttt{pes}, \texttt{inplace}, \texttt{op}, \texttt{mabort}, \texttt{irr} \mid \texttt{luopacity}, \texttt{strongprog}$$

4 Other

After discussing various research directions during the short visit at EPFL in September, the researchers involved in the STSM (i.e., Konrad Siek, Paweł Wojciechowski, and Rachid Guerraoui) agreed that the most interesting avenue for further joint work on the TM model would be to focus on weakening the consistency property through eventual consistency (EC). Since the parties involved in the STSM have already worked on the the topic of EC [6, 13], it was decided to combine our strengths and investigate how EC pertains to the theory and practice of TM. Hence, as a result of the STSM the researchers were able to discuss their ideas and, in eect, to plan a future cooperation to work on eventual consistency with a goal of creating properties that describe EC in the context of TM. While the work is still preliminary, we expect this aspect of the cooperation to yield publishable results in the near future (perhaps in 2015).

5 Confirmation by the host institution of the successful execution of the STSM

References

[1] Y. Afek, A. Matveev, and N. Shavit. Pessimistic Software Lock-Elision. In *Proceedings of DISC'12: the 26th International Symposium on Distributed Computing*, pages 297–311, Oct. 2012.

[2] J. Błażewicz, J. K. Lenstra, and A. H. G. R. Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete Appl. Math*, 5, 1983.

[3] P. K. Chrysanthis and K. Ramamritham. Acta: a framework for specifying and reasoning about transaction structure and behavior. In *Proceedings of ACM SIGMOD'90: the International Conference on Management of Data*, June 1990.

[4] D. Dice, O. Shalev, and N. Shavit. Transactional Locking II. In *Proceedings of DISC'06: the 20th International Symposium on Distributed Computing*, Sept. 2006.

[5] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan. Optimization and approximation in deterministic sequencing and scheduling theory: a survey. *Ann. Discrete Math*, 5, 1979.

[6] R. Guerraoui and E. Ruppert. A paradox of eventual linearizability in shared memory. In *Proceedings of PODC '14: the 2014 symposium on Principles of Distributed Computing*, July 2014.

[7] S. Hirve, R. Palmieri, and B. Ravindran. HiperTM: High Performance, Fault-Tolerant Transactional Memory. In *Proceedings of ICDCN'14: the 15th International Conference on Distributed Computing and Networking*, Jan. 2014.

[8] T. Kobus, M. Kokociński, and P. T. Wojciechowski. Hybrid replication: State-machine-based and deferred-update replication schemes combined. In *Proceedings of ICDCS'13: the 33rd International Conference on Distributed Computing Systems*, July 2013.

[9] A. Matveev and N. Shavit. Towards a Fully Pessimistic STM Model. In *Proceedings of TRANSACT '12: the 7th ACM SIGPLAN Workshop on Transactional Computing*, number 7437 in Lecture Notes in Computer Science, pages 192–206, Aug. 2012.

[10] M. M. Saad and B. Ravindran. HyFlow: A High Performance Distributed Transactional Memory Framework. In *Proceedings of HPDC '11: the 20th International Symposium on High Performance Distributed Computing*, June 2011.

[11] K. Siek and P. T. Wojciechowski. Atomic RMI: a Distributed Transactional Memory Framework. In *Proceedings of HLPP'14: the 7th International Symposium on High-level Parallel Programming and Applications*, pages 73–94, July 2014.

[12] A. Turcu, B. Ravindran, and R. Palmieri. HyFlow2: A High Performance Distributed Transactional Memory Framework in Scala. In *Proceedings of PPPJ'13: the 10th International Conference on Principles and Practices of Programming on JAVA platform: virtual machines, languages, and tools*, Sept. 2013.

[13] P. T. Wojciechowski and K. Siek. Having your cake and eating it too: Combining strong and eventual consistency. In *Proceedings of PaPEC 2014: the 1st Workshop on the Principles and Practice of Eventual Consistency*, Apr. 2014.