Detecting Snapshot Isolation Anomalies in Transactional Memory

Ricardo Dias

1 Abstract

Using relaxed isolation levels, such as Snapshot Isolation (SI) [1], is a well known and long used strategy in database transactional systems to increase performance. Transactional Memory [7, 3] (TM) usually requires full serializability but, in principle, can also use SI, although with different semantic guaranties.

Unlike full-fledged TM systems that provide strict isolation between transactions, TM systems providing relaxed isolation levels allow for transactions to interfere, and generate non-serializable execution schedules. Such interferences are commonly known as *serializability anomalies* [1].

We introduce a static analysis technique and corresponding algorithm that mechanically asserts that a given transactional memory application executing under Snapshot Isolation behaves as if executed under full serializability. More precisely, we detect the kind of anomalies of Snapshot Isolation that are known to lead to non-serializable histories [1]. We build on related work directed to database transactions [2, 4] and extend it to the very different domain of transactional memory. Our technique allows for the automatic identification of anomalous code patterns which were previously detected only by ad-hoc code inspection. To the best of our knowledge, SI has only been applied to TM in semantically harmless situations [5], where anomalies did not occur. Our technique allows SI to be used as the default isolation level for TM applications by signaling conflicting programs and allowing for explicit code corrections.

We take a standard approach to define our analysis, and use a simple core imperative language with limited use of pointer indirection and support for heap allocated data. Applications written in main stream programming languages (e.g., Java) are translated to the core language and then analyzed. For each transaction in the original application we create a separate program in the core language. We assume that all transactions in the application can run concurrently, and define an intra-procedural data-flow analysis that extracts the information necessary to detect SI anomalies.

The analysis starts by extracting compact read/write sets and defining static dependencies between programs, thus creating a static dependency graph. We then apply an algorithm to that graph to determine if the concurrent execution of such transactional programs is serializable under SI.

The analysis of heap allocated data follows a modified shape analysis technique [6], that combines shape graphs with sets of read and written data items. We define a set of new properties for shape graphs in order to avoid the state explosion that would result from the application of the original definition to our setting. Unlike standard shape analysis procedures, termination in our approach does not depend on the comparison of shape graphs but rather on the collected read/write-sets, whose computation converges faster. A drawback of our definition is that it can only be applied to acyclic data structures, but we foresee that the model can be extended with backward pointers to support more general cases.

Like in any other static analysis procedure, we allow for some kind of false positive results. Nevertheless, we guarantee that if the analysis procedure does not detect any serializability anomaly, then all possible executions will be anomaly-free and correspond to a possible interleaving of the transactions. On the other hand, if some anomalies are found, they should be considered as potentially harmful. At this stage, analysis results can be further refined and code can be modified to avoid undesired interferences.

References

[1] H. Berenson, P. Bernstein, J. N. Gray, J. Melton, E. O'Neil, and P. O'Neil. A critique of ANSI SQL isolation levels. In *SIGMOD '95: Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, pages 1–10, New York, NY, USA, 1995. ACM.

[2] A. Fekete, D. Liarokapis, E. O'Neil, P. O'Neil, and D. Shasha. Making snapshot isolation serializable. *ACM Trans. Database Syst.*, 30(2):492–528, 2005.

[3] M. Herlihy, V. Luchangco, M. Moir, and I. William N. Scherer. Software transactional memory for dynamic-sized data structures. In *PODC '03: Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 92–101, New York, NY, USA, 2003. ACM.

[4] S. Jorwekar, A. Fekete, K. Ramamritham, and S. Sudarshan. Automating the detection of snapshot isolation anomalies. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 1263–1274, Vienna, Austria, 2007. VLDB Endowment.

[5] T. Riegel, C. Fetzer, and P. Felber. Snapshot isolation for software transactional memory. In *TRANSACT'06: First ACM SIGPLAN Workshop on Languages, Compilers, and Hardware Support for Transactional Computing*, Ottawa, Canada, June 2006.

[6] M. Sagiv, T. Reps, and R. Wilhelm. Solving shape-analysis problems in languages with destructive updating. *ACM Trans. Program. Lang. Syst.*, 20(1):1–50, 1998.

[7] N. Shavit and D. Touitou. Software transactional memory. In *PODC '95: Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing*, pages 204–213, New York, NY, USA, 1995. ACM.