Input Acceptance of Time-Warping Transactional Memory

Nuno Diegues and Paolo Romano
INESC-ID / Instituto Superior Técnico, Universidade de Lisboa, Portugal
ndiegues@gsd.inesc-id.pt romano@inesc-id.pt

***Abstract*—Existing Transactional Memory (TM) algorithms typically abort many transactions that could be safely committed. The extent to which a TM allows such spurious aborts is captured by the theory of Input Acceptance. Recently, the Time-Warping Multi-version (TWM) algorithm was proposed to minimize spurious aborts without hampering practical performance. In this work we seek to theoretically assess the input acceptance of TWM and compare it with other TMs.**

I. Introduction

The growing interest in TM research has led to the development of STMs designed to maximize single-threaded performance and reduce book-keeping overhead [1], [2], [3], [4]. As a consequence, these algorithms are optimized for uncontended scenarios and end up rejecting a large number of serializable schedules (i.e., creating spurious aborts). Conversely, several (mostly theoretical) proposals [5], [6], [7], [8], [9], [10] were designed with the main concern of reducing spurious aborts, and ultimately achieving permissiveness [8].

We have recently identified a sweet spot between efficiency (i.e., avoiding costly book-keeping operations) and the ability to avoid spurious aborts, proposing [11] the Time-Warping Multi-version (TWM) algorithm: (1) TWM deterministically accepts many common patterns rejected by practical TM algorithms, by tracking only *direct* conflicts between transactions; and (2) it exploits multi-versioning to further reduce aborts and achieve mv-permissiveness [12].

The key idea in TWM is to allow a write transaction to commit "in the past", by ordering the data versions it produces before those generated by already committed, concurrent transactions. In this case we say that T performs a *time-warp* commit. The validation scheme of TWM detects a specific pattern named *triad*. A triad exists whenever there is transaction T that is both the source and target of anti-dependency edges [13] from two concurrent transactions T' and T'' (where, possibly, $T' = T''$). We call T a *pivot*, and define the TWM validation scheme as follows: a transaction fails its validation if, by committing, it would create a triad whose *pivot* time-warp commits.

In this work we seek to compare TWM with other existing TMs from a theoretical perspective. For this we shall use the theory of Input Acceptance [6], summarized in the following section.

II. Using the theory of Input Acceptance

The key idea of input acceptance is to identify a sequence of input events (an input pattern) that, when fed to a TM, leads to aborting at least one transaction. In such case the input pattern is rejected by the TM. If the pattern results in no aborts, then it is accepted. An input class is a set of input patterns, which is rejected only when all its input patterns are rejected. Conversely, a class is accepted only when all the patterns it comprises are accepted. Next, we conduct our analysis using the notation of input acceptance [6]. By comparing the input class of different TM algorithms (also called designs, in this context), it is possible to define a hierarchy of TMs that captures their relative ability of avoiding spurious aborts. We note that, in contrast, it would not be possible to achieve such a fine-grained classification by using solely the permissiveness concept [8]. To the best of our knowledge, the only opaque-permissive (online) TM is AbortsAvoider [7]. Every other online TM is simply not permissive. Yet, in practice there exist significant performance differences between such non-permissive TMs.

We briefly summarize the notation used in the input acceptance framework. Its idea is to depict the patterns using events Γ_t^x, where Γ denotes a read (r) or write (w) event of transaction t over datum x. Start and commit input events are respectively denoted by s, c. Moreover, an event π^* means any of the previous events (including itself). This set of primitives is complemented with the Kleene closure '*', complement operator '¬', and the choice operator '|'. This allows denoting $\neg(w_p^x \mid w_i^y)^*$ as any sequence of events except the writes of transaction T_p to x or T_i to y. To simplify the new classes that we present next, we complement the previous notation with the following. We assume the inputs are well formed accordingly to Gramoli et al. [6] and also that a commit event for i cannot occur as a replacement of a π^*, if c_i is explicitly demanded in the input sequence. Further, we use the permutation operator ε as a suffix for a sequence of events to indicate any possible permutation of those events.

Finally, we represent dependencies between transactions [13] according to their execution history as follows: (1) $A \xrightarrow{wr} B$ when B read-depends on A because it read at least one of A's updates; (2) $A \xrightarrow{ww} B$ when B write-depends on A because it overwrote at least one of A's updates; (3) $A \dashrightarrow B$ when A anti-depends on B because A read some version of a variable for which B commits a new version.

A. Invisible Writes Invisible Reads

The Invisible Writes Invisible Reads (IWIR) design is characterized read operations that do not modify shared memory and by write operations that are buffered in private memory until the transaction is deemed successful (e.g., in TL2 [1]). In the the work of Gramoli et al. [6], it was shown that the IWIR

$$\mathcal{C}3 = \pi^*(r_i^x \neg c_i^* w_j^x)^\varepsilon \neg c_i^* c_j \pi^*$$

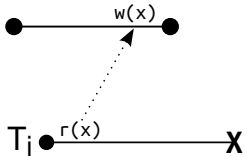

Fig. 1: Class $\mathcal{C}3$ and a history output by a IWIR TM for an example pattern included in the class.

design rejects all the patterns in a class named $\mathcal{C}3$. We present this class in Fig. 1, along with an example of a possible history output by a IWIR TM when fed with a pattern included in $\mathcal{C}3$. The essence of this class is to capture the classic validation that ensures data items read are still up-to-date at commit time. As we can see, it only enforces the existence of a single anti-dependence between concurrent transactions $T_i \dashrightarrow T_j$, where T_j commits before T_i.

The other designs corresponding to Visible Reads Visible Writes (VWVR) (e.g., SXM [14]), and Visible Writes Visible Reads (VWIR) (e.g., TinySTM [4]), were shown to have a lower input acceptance than IWIR in [6].

B. Time-Warp Multi-version

To study TWM we create new classes to capture the patterns that it rejects, reported in Fig. 2. These classes are governed by the detailed algorithm presented in [11]. Briefly, they account for the rule that a transaction aborts only if it completes a triad in which the pivot time-warp commits.

We first present $\mathcal{C}_{nsr}$ to capture degenerate triads that are composed of just two transactions; this is the simple case where any of the two transactions in the triad can be seen as the pivot. For that, both T_i and T_j read and write two variables, creating a cycle due to the anti-dependencies $T_i \dashrightarrow T_j \dashrightarrow T_i$. An example history of this class is shown in the left side of Fig. 2. Note that this class is more restrictive than $\mathcal{C}3$, i.e., $\mathcal{C}_{nsr} \subseteq \mathcal{C}3$.

Then we present class $\mathcal{C}_{tw}$ to capture the case where the triad is composed of three different transactions. For this, we identify an explicit transaction T_p in the patterns, which can be seen as the pivot of the triad. The essence of $\mathcal{C}_{tw}$ is to enforce two anti-dependencies between the three transactions in the triad: $T_i \dashrightarrow T_p \dashrightarrow T_j$. We divide the expression of $\mathcal{C}_{tw}$ in two parts, $\mathcal{C}_{tw1}$ and $\mathcal{C}_{tw2}$, to enhance readability — they are very similar to each other, and $\mathcal{C}_{tw2}$ differs only in that it generates sequences of events where w_p^y takes place before T_i starts. The example history shown on the right side of Fig. 2 is derived using the component of $\mathcal{C}_{tw1}$.

Finally, we show $\mathcal{C}_{twall}$, which is comprised of all the previous classes. In the following, we prove that this class represents all input patterns rejected TWM.

Lemma 1. Class $\mathcal{C}_{twall}$ is an upper bound of the patterns rejected by TWM.

Proof: To prove this, we observe that every abort in TWM happens only when the algorithm is fed with an input pattern captured by $\mathcal{C}_{twall}$.

TWM aborts a transaction T_i when: (1) T_i is validating; (2) there is some T_p whose write T_i misses, i.e., $T_i \dashrightarrow T_p$; and (3) T_p time-warp committed because it missed the write of another transaction, say T_j, such that $T_p \dashrightarrow T_j$. The set of conditions that trigger this abort are thus captured by patterns contained in class $\mathcal{C}_{tw}$. Namely, by instantiating $\mathcal{C}_{tw}$, with either $\mathcal{C}_{tw1}$ or $\mathcal{C}_{tw2}$, and specifying that c_p happens before c_i.

We are left with aborts in the following conditions: (1) T_p is validating; (2) there is some T_i that missed a write of T_p, i.e., $T_i \dashrightarrow T_p$; and (3) T_p missed the write of another transaction, say T_j, such that $T_p \dashrightarrow T_j$. In this case it is possible that T_i and T_j are the same transaction, which is captured by class $\mathcal{C}_{nsr}$ where any of the transactions can be seen as our pivot T_p. Otherwise, when we actually have three transactions, this is also captured by $\mathcal{C}_{tw}$, but with c_i happening before c_p.

Consequently we have that all aborts of TWM are captured by $\mathcal{C}_{twall}$, which makes it an upper bound on the patterns rejected by it. ■

We can now use this result to start placing TWM in the hierarchy.

Theorem 2. *TWM has a larger input acceptance than the IWIR design.*

Proof: By Gramoli et al. [6] we have that the IWIR design rejects all the patterns contained in class $\mathcal{C}3$. By Lemma 1 we have that $\mathcal{C}_{twall}$ captures all patterns rejected by TWM. It is straightforward to see, by examining the regular expressions defining these two classes, that $\mathcal{C}_{twall} \subseteq \mathcal{C}3$. Therefore we can conclude that TWM has a larger input acceptance than IWIR. ■

C. Interval-based

This design typically maintains a lower and upper bound for each transaction T ($T.lb$ and $T.ub$), which denotes the

$$\mathcal{C}_{nsr} = \pi^*(r_i^y \pi^* w_i^x \pi^* r_j^x \pi^* w_j^y)^\varepsilon (c_i c_j)^\varepsilon \pi^*$$

$$\mathcal{C}_{tw1} = \left(r_p^x \neg (w_p^x)^* w_j^x\right)^\varepsilon c_j \neg \left(w_p^x\right)^* s_i \left(r_i^y \neg (w_i^y \mid w_p^x)^* w_p^y\right)^\varepsilon$$

$$\mathcal{C}_{tw2} = \left(r_p^x \neg (w_p^x)^* w_j^x w_p^y\right)^\varepsilon c_j \neg (w_p^x)^* s_i \neg (w_p^x \mid w_i^y)^* r_i^y$$

$$\mathcal{C}_{tw} = \pi^*(\mathcal{C}_{tw1} \mid \mathcal{C}_{tw2})(c_p c_i)^\varepsilon \pi^*$$

$$\mathcal{C}_{twall} = \mathcal{C}_{tw} \mid \mathcal{C}_{nsr}$$

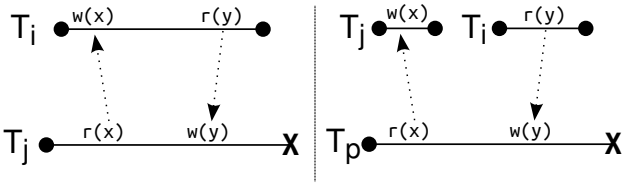

Fig. 2: Class $\mathcal{C}_{twall}$ and its decomposition. We also show two example output histories: on the left, a degenerate triad with only two transactions, derived from $\mathcal{C}_{nsr}$; and on the right a possible triad obtained from $\mathcal{C}_{tw}$, by deriving from $\mathcal{C}_{tw1}$ in the choice operator.

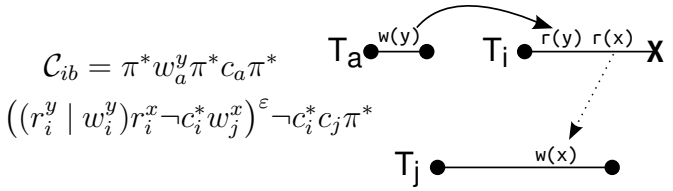

Fig. 3: Class $\mathcal{C}_{ib}$ and an example history output by a TM using the IB design for a pattern included in the class.

interval of possible serialization points for T (e.g. AVSTM [8] and TSTM [9]). A transaction T is aborted in this design if its interval is empty, i.e., whenever $T.lb > T.ub$. Otherwise, it commits by choosing one point among the possible ones in the interval, which is assigned to $T.ser$. Briefly, the idea is that these bounds reflect the dependencies and anti-dependencies of T: (i) when T reads a value committed by transaction T', it enforces $T.lb \leftarrow max(T.lb, T'.ser)$; (ii) when T' commits, for each T whose footprint coincides on at least a datum, it enforces $T.ub \leftarrow min(T.ub, T'.ser)$.

Next, in Fig. 3, we present a new class $\mathcal{C}_{ib}$ by restricting class $\mathcal{C}3$. The idea is to have a transaction T_i that may abort (as in the original $\mathcal{C}3$); for this to happen, we must restrict both $T_i.lb$ and $T_i.ub$ according to the rules defined above. We present an example pattern in Fig. 3, where $T_a \xrightarrow{wr} T_i$ enforces $T_a.ser < T_i.lb$, and $T_i \dashrightarrow T_j$ enforces $T_i.ub < T_j.ser$. Note that this class is stricter than $\mathcal{C}3$, meaning $\mathcal{C}_{ib} \subseteq \mathcal{C}3$, as it can be derived by adding constraints to $\mathcal{C}3$. We can now use this new class to study the power of the IB design.

Lemma 3. Class $\mathcal{C}_{ib}$ is an upper bound of the patterns rejected by the IB design.

Proof: We start by showing that the IB design may reject only patterns contained in $\mathcal{C}_{ib}$. We observe that a transaction T_i can only be aborted when $T.lb > T.ub$. This scenario occurs only if the following conditions hold:

1) $\exists\ T_a : T_a \xrightarrow{wr} T_i \vee T_a \xrightarrow{ww} T_i$
2) $\exists\ T_j : T_i \dashrightarrow T_j$
3) $T_a.ser > T_j.ser$.

where T_a and T_b are two transactions such that with $T_a \neq T_b \neq T_i$. As a result of the third condition, and using the rules described above for the IB design, we obtain $T_i.lb > T_i.ub$, which results in T_i aborting. It is straightforward to see that the necessary conditions 1 and 2 are exactly embodied by class $\mathcal{C}_{ib}$: it specifically dictates the existence of transactions T_a and T_j creating the required dependencies, and no more than that. If condition 3 is also verified, then this set of conditions is both sufficient and necessary for aborting a transaction in IB.

By noting that $\mathcal{C}_{ib}$ always ensures conditions 1 and 2, we have that it is an upper bound on the input patterns rejected by IB. ■

Theorem 4. *The IB design has a larger input acceptance than the IWIR design.*

Proof: Once again, by Gramoli et al. [6], we have that IWIR rejects all the patterns in $\mathcal{C}3$. By Lemma 3 we have that $\mathcal{C}_{ib}$ captures all patterns rejected by IB. By examining the two classes we can see that $\mathcal{C}_{ib} \subseteq \mathcal{C}3$. Then, it follows that IB has a larger input acceptance class than IWIR. ■

D. Comparing TWM and IB

By using Theorems 2 and 4, we can correctly place TWM and IB above the IWIR design in terms of input acceptance. The question now is to assess the relative power of TWM and IB. For that, we study the acceptance of TWM with regard to $\mathcal{C}_{ib}$ and the acceptance of IB with regard to $\mathcal{C}_{twall}$. In doing so, we arrive at the conclusion that they are incomparable.

Lemma 5. The IB design does not have larger input acceptance than TWM.

Proof: To prove this we use a counter-example input pattern that is rejected by the IB design but accepted by TWM.

The first task is to find such pattern by deriving it from $\mathcal{C}_{ib}$. For that we build a pattern $\mathcal{P}$, contained in $\mathcal{C}_{ib}$, where $T_a.ser$ is arbitrarily increased by adding transactions to $past(T_a)$. We define $past(T)$ analogously to [10], as the transitive closure of the set of transactions on which T depends, and removing any anti-dependencies.

In other words, there exists a finite number of such transactions in $past(T_a)$ that makes $T_a.ser > T_j.ser$ for any choice of serialization points for T_i, thus causing the abort of T_i. As a result, the IB design rejects $\mathcal{P}$. This pattern $\mathcal{P}$ is easily derived from $\mathcal{C}_{ib}$ by: (1) replacing the first π^* with the extension described; (2) replacing the last π^* with c_i; and (3) removing the other wildcards.

Now we show that $\mathcal{P}$ is accepted by TWM. From the validation rule described for TWM, it follows that no transaction is aborted if it commits without completing a triad where the pivot time-warp commits. But the pattern $\mathcal{P}$ contains only one anti-dependency, that of $T_i \dashrightarrow T_j$, which is insufficient to generate a triad. Consequently TWM accepts $\mathcal{P}$. As a result, $\mathcal{P}$ serves as the counter-example that proves the Lemma. ■

Lemma 6. TWM does not have a larger input acceptance than the IB design.

Proof: We can prove this by showing an example pattern $\mathcal{P}$ accepted by IB but rejected by TWM. We derive $\mathcal{P}$ from $\mathcal{C}_{twall}$, by: (1) expanding $\mathcal{C}_{tw1}$; and (2) removing all wildcards. A possible ordering of events when expanding the permutation operator (ε) is that reported on the right side in Fig. 2. That pattern was already shown to be rejected by TWM in Lemma 1.

We now argue that $\mathcal{P}$ is accepted by IB. For that, we note that $T_i.lb$ is constrained when T_i performs reads, which in $\mathcal{P}$ happens only with r_i^y. Note that no transaction writes to y in $\mathcal{P}$. We can consider that a special transaction initially writes every variable with a default value and serializes on point 0. This means that $T_i.lb$ is unconstrained. Consequently, no matter what is the value of the upper bound, constrained by $T_i.ub < T_p$, $T_i.lb$ will always be lower and thus allow T_i to commit successfully. As a result, IB accepts $\mathcal{P}$ and it serves as a counter-example that proves the Lemma. ■

Theorem 7. *TWM and the IB design are incomparable with regard to input acceptance.*

Proof: This follows trivially from Lemmas 5 and 6, as neither approach has larger input acceptance than the other. ■

E. Serializability Graph Testing

Finally, we describe a design with higher input acceptance than TWM and IB. To do so, we identify a design called Serializability Graph Testing (SGT). This design draws its key idea from the Serializability Theorem [15]. This theorem states that a history $\mathcal{H}$ is serializable if and only if the corresponding Direct Serialization Graph DSG($\mathcal{H}$) is acyclic. The AbortsAvoider TM [7] explores this technique to ensure online-opaque-permissiveness by maintaining an explicit DSG. On the other hand, SSTM [6] also uses SGT, albeit it scatters the corresponding metadata across transactions. This means there is no centralized (or even explicit) DSG; but this is merely an implementation detail.

III. Revised Input Acceptance Hierarchy

We can finally revise the hierarchy originally presented in [6] according to our results, as shown in Fig. 4. By Theorems 2 and 4 we place TWM and IB above IWIR, and by Theorem 7 we place them side by side.

So far, in our analysis, we considered solely write transactions. This was done because we compare both single-versioned and multi-versioned TM designs, where the latter can deterministically avoid aborting read-only transactions [12]. By accounting also for read-only transactions in our input acceptance analysis, we introduce an additional design called MV-IWIR. This design includes mv-permissive TMs based on IWIR algorithms and, aside from this, uses the same validation rule to commit update transactions. Both JVSTM [2] and SMV [16] exploit this design by using invisible reads, deferred writes, and multi-versions to guarantee that read-only transactions never abort.

The only difference from IWIR to MV-IWIR is that read-only transactions do not abort. This means that, by considering only write transactions, MV-IWIR also rejects all the patterns included in $\mathcal{C}3$. If read-only transactions are identified a priori, i.e., the s event of a read-only transaction is annotated, then MV-IWIR obtains more input acceptance than IWIR. This is trivially shown by verifying that those TMs always commit T_i successfully in the execution shown in Fig. 1. As a result, MV-IWIR only rejects a subset of the patterns contained in $\mathcal{C}3$, and thus has a larger input acceptance than IWIR. The reason why TWM has a larger input acceptance than MW-IWIR is that, despite both being mv-permissive, the class of update transactions rejected by TWM is a subset of that rejected by MV-IWIR.

We additionally observe that IB is incomparable with MV-IWIR. On one hand, IB accepts the example execution in Fig. 1, which is rejected by MV-IWIR if T_i is a write transaction. On the other hand, MV-IWIR accepts the example execution in Fig. 3 if T_i is a read-only transaction, which was shown to be rejected by IB. TWM remains incomparable with IB even with read-only transactions, as the considerations concerning the existence of input patterns accepted by IB and rejected by TWM (and vice versa) still apply.

This concludes our theoretical analysis of TWM, with the resulting hierarchy justifying (together with our experimental study [11]) the prominent place of time-warping in efficiently minimizing spurious aborts in TM applications.

References

[1] D. Dice, O. Shalev, and N. Shavit, "Transactional Locking II," in *DISC '06*.

[2] S. M. Fernandes and J. Cachopo, "Lock-free and scalable multi-version software transactional memory," in *PPoPP '11*.

[3] A. Dragojević, R. Guerraoui, and M. Kapalka, "Stretching transactional memory," in *PLDI '09*.

[4] P. Felber, C. Fetzer, and T. Riegel, "Dynamic performance tuning of word-based software transactional memory," in *PPoPP '08*.

[5] H. E. Ramadan, I. Roy, M. Herlihy, and E. Witchel, "Committing conflicting transactions in an STM," in *PPoPP '09*.

[6] V. Gramoli, D. Harmanci, and P. Felber, "On the Input Acceptance of Transactional Memory," *Parallel Processing Letters*, vol. 20, no. 1, pp. 31–50, 2010.

[7] I. Keidar and D. Perelman, "On avoiding spare aborts in transactional memory," in *SPAA '09*.

[8] R. Guerraoui, T. A. Henzinger, and V. Singh, "Permissiveness in Transactional Memories," in *DISC '08*.

[9] U. Aydonat and A. T., "Serializability of transactions in software transactional memory," in *TRANSACT '08*.

[10] T. Crain, D. Imbs, and M. Raynal, "Read invisibility, virtual world consistency and probabilistic permissiveness are compatible," in *ICA3PP '11*.

[11] N. Diegues and P. Romano, "Brief Announcement: Enhancing Permissiveness in Transactional Memory via Time-Warping," in *DISC '13*.

[12] D. Perelman, R. Fan, and I. Keidar, "On maintaining multiple versions in STM," in *PODC '10*.

[13] A. Adya, "Weak consistency: a generalized theory and optimistic implementations for distributed transactions," Ph.D. dissertation, Massachusetts Institute of Technology, 1999, aAI0800775.

[14] R. Guerraoui, M. Herlihy, and B. Pochon, "Polymorphic contention management," in *DISC'05*.

[15] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency control and recovery in database systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1987.

[16] D. Perelman, A. Byshevsky, O. Litmanovich, and I. Keidar, "SMV: Selective Multi-Versioning STM," in *DISC '11*.

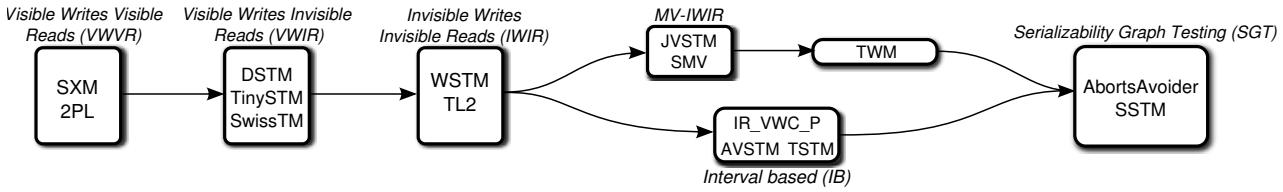

Fig. 4: Comparing the input acceptance of the Time-Warp approach and of some well-known TM designs.