One-pager 4

Martin Vassor

1 Introduction

This introduction explains assumptions and the outline.

Tsx consists of two parts: an optimistic transaction, without locking and which is aborted in case of conflicts, and a fall-back transaction, with locking to serialize accesses.

In crew concurrency model, each object has at most one *writer* at any time. This privilege granting is done statically or dynamically (`compare-and-swap` to initially increment the version number –if it is even–, ensuring the uniqueness of writer).

A crew transaction has two parts: first optimistic reads and actual computation, then the critical section: acquiring write privileges, verifying *readSet* is unmodified, and writing back.

Section **??** and **??** study the usefulness of crew in the optimistic (resp. fall-back) part of tsx. Section **??** suggests applications for weaker property requirements.

Other cases are not studied, as different merging strategies can be reduced to the studied cases.

2 Crew in optimistic TSX

This section shows that tsx is more conservative than crew.

Claim 1. *If a transaction T in* crew *aborts, then T in* tsx *aborts.*

Proof. If the crew aborts, an object in the *readSet* has been publicly modified. Then, its cache line is invalidated, which aborts the optimistic part. □

Thus, having a crew transaction inside an optimistic section of tsx is not useful as tsx will catch all conflicts. By extension, any overlapping an optimistic tsx section with a crew transaction is not useful: either the tsx part covers all the critical part (claim **??** applies), either tsx does not cover all the critical part, hence it provides no guarantee and should be extended or removed[1].

Also, notice that having crew in an optimistic tsx increases the probability of tsx abort, as there is more cache-lines used.

3 Crew as fall-back behaviour

This section presents cases in which using a crew strategy in the fall-back section is useful.

As the fall-back section of a tsx transaction contains the default behaviour, the programmer as to explicitly define the concurrency model, which can be crew.

One might expect the benefits to be the same than in regular concurrency control. However, the usefulness of having a optimistic concurrency model is lesser, as the optimistic section of the tsx already filters deterministic conflicts. Hence, the benefits are only possible on the limits of tsx, for instance if the *dataSet* does not fit in the cache, causing tsx conservatively fails, although without conflicts. Notice that the crew semantics are preserved. Even if writer uniqueness is not explicit, tsx is conservative, that is it will abort in case of conflicts.

4 Weaker concurrent model

Crew can also benefit when relaxing semantics, as Tsx conflict detection can not be tuned for weaker requirements.

In crew, conflicts are detected and resolved by the programmer, who can use his knowledge of the program to adapt the concurrency control.

Word count: $141 + 175 + 137 + 45 = 498$, including footnotes, titles, excluding references

[1] This case explains why the critical section alone of crew can not be implemented with rtx: the rtx transaction should contain all optimistic read of the crew transaction, nullifying crew benefits.