Innovative Approach to Achieve Compositionality
Efficiently using Multi-Version Object Based
Transactional Systems *

Chirag Juyal¹, Sandeep Kulkarni², Sweta Kumari¹, Sathya Peri¹, and Archit Somani¹**

¹ Department of Computer Science Engineering, IIT Hyderabad,
(cs17mtech11014, cs15resch01004, sathya_p,
cs15resch01001)@iith.ac.in

² Department of Computer Science, Michigan State University
sandeep@cse.msu.edu

Introduction on STM: To utilize the cores of multicore processors, synchronization and
communication among them involve high cost. Software transaction memory systems
(STMs) addresses this issues and provide better concurrency in which programmer
need not have to worry about consistency issues such as locking, races and deadlocks
etc. Concurrently executing transactions access shared memory through the interface
provided by the STMs.

Another advantage of STMs is that they facilitate compositionality of concurrent
programs with great ease. Different concurrent operations that need to be composed to
form a single atomic unit is simply achieved by encapsulating all these operations as
a single transaction. Composition of concurrent programs is a very nice feature which
makes STMs very appealing to use by programmers.

Most of the STMs proposed in the literature are specifically based on read/write
primitive operations (or methods) on memory buffers (or memory registers). These STMs
typically export the following methods: t.begin which begins a transaction, t.read (or r)
which reads from a buffer, t.write (or w) which writes onto a buffer, t.commit which
validates the operations of the transaction and tries to commit. If validation is successful then it
returns commit otherwise STMs export t.abort which returns abort. We refer to these as
Read-Write STMs or RWSTMs.

On the other hand, Object-based STMs or OSTMs operate on higher level objects
rather than read & write operations on memory locations. It was shown in databases that
object-based schedulers provide greater concurrency than read-write based systems. So,
Herlihy et al. [1], Harris et al. [2], and [3, Chap 6] extended this concept to STMs. We
have consider an OSTM using hash table [4]. It exports the following methods: (1)
begin which begins a transaction, (2) insert (or i) which inserts a value for a given
key, (3) delete (or d) which deletes the value associated with the given key and returns
the current value of the key, (4) lookup (or l) which looks up the value associated with
the given key and (5) commit which validates the operations of the transaction.

Figure 1 represents a hash table which contains nodes with keys $(k_2, k_5, k_7, k_8)$
between two sentinel nodes $\infty$ and $+\infty$. We denote the RWSTMs and OSTMs operations
as layer-0 (or leaves) and layer-1 respectively in the form of transactional forest shown

* The initial work was accepted in AADDA workshop, ICDCN’18 as work in progress.
** Author sequence follows the lexical order of last names.
OSTM has been accepted as a full paper in Netys-18.
in Figure 1(b). Suppose transactions $T_1$ and $T_2$ are concurrently executing. Consider
the history at layer-0 (while ignoring higher-level operations), denoted as $H[0]$. It is not opaque
because between the two reads of $k_5$ by $T_1$, $T_2$ writes to $k_5$. In order to ensure opacity
for $H[0]$, one of the transactions among $T_1$ or $T_2$ would be aborted.

**Objective of MV-OSTM**: It was observed in databases and STMs that storing multiple
versions in RWSTMs provides better concurrency [6]. Maintaining multiple versions
of each key can ensure that more read operations succeed because the reading operation will have
an appropriate version to read. So, we motivated to develop a multi-version OSTM as MV-OSTM.

**Fig. 1**: Motivational example of OSTM s Vs RWSTMs

On the other hand, consider the history $H[1]$ at layer-1 consisting of $l$ and $d$ operations
(while ignoring the underlying read/write operations, since they do not overlap (referred to as pruning in [3, Chap 6]). These methods operate on different keys ($k_5$, $k_7$ and $k_8$), so they are not conflicting and can be re-ordered either way. Thus, we get that $H[1]$ is opaque [5] with $T_1 T_2$ (or $T_2 T_1$) being an equivalent serial history. Hence, OSTM reduces number of aborts and provides greater concurrency.

**Fig. 2**: Advantages of multi-version over single-version OSTM s

Figure 2 a) represents a history $H$ with two concurrent transactions $T_1$ and $T_2$
operating on a hash table $ht$. $T_1$ first performs a $t$lookup on key $k_2$. But due to
absence of key $k_2$ in $ht$, it gets $NULL$. After that suppose $T_2$ invokes $t$insert
method on the same key $k_2$ and inserts the value $v_2$ in $ht$. Then $T_2$ deletes the key $k_1$
from $ht$ and returns $v_3$ implying that some other transaction had previously inserted $v_3$ into $k_1$.

The second method of $T_1$ is $t$lookup on the key $k_1$. In this case the STMs has to return
abort to ensure correctness, i.e., opacity. If $T_1$ obtained a return value of $NULL$ for $k_1$,
then the history will not be serial and hence not opaque.

In order to improve concurrency, we can use multiple versions for each key. Whenever
a transaction inserts or deletes, a new version is created. Hence, consider the example
shown in Figure 2b, even after \( T_2 \) deletes \( k_1 \), the previous value of \( v_3 \) is still retained. Thus, when \( T_1 \) invokes \( t_{\text{lookup}} \) on \( k_1 \) after the delete on \( k_1 \) by \( T_2 \), it will return \( v_3 \) (as previous value) and the history is opaque with the equivalent serial history being \( T_1 T_2 \).

Thus, \( MV-OSTM \) reduce number of aborts and achieve greater concurrency than \( OSTM \)s while ensuring the compositionality. To the best of our knowledge, this is the first work to explore the idea of using multiple versions in \( OSTM \)s to achieve greater concurrency. Currently, we have developed \( MV-OSTM \) with the \( \infty \) number of versions for each key. So, we worked on garbage collection method to delete the unwanted versions of a key (omitted for lack of space). Our \textit{contributions} are as follows: a) We have proposed a new STM as \( MV-OSTM \) which providing the greater concurrency with the help of multiple versions to reduce the number of aborts and its composable too. b) \( MV-OSTM \) will never aborts a lookup only transaction because of \( \infty \) versions. c) We have developed the \textit{garbage collection} method to delete unwanted versions from \( MV-OSTM \). d) \( MV-OSTM \) satisfies \textit{opacity}.

\textbf{Theorem 1} Any history \( H \) generated by \( MV-OSTM \) is opaque iff it produces acyclic graph of \( H \).

\textbf{Conclusion and Future Work:} STMs is an alternative to provide synchronization and communication among multiple threads without worrying about consistency issues. We have proposed a new STM as \( MV-OSTM \) which provides the greater concurrency in terms of number of abort with the help of multiple version and composability. It satisfies correctness-criteria as \textit{opacity}. Further, we want to optimize \( MV-OSTM \) with limited (say \( k \)) number of versions corresponding to each key. Later on, we will implement our proposed \( MV-OSTM \) with the unlimited and limited number of version and compare its performance.

\textbf{References}


\textsuperscript{d} Find technical report link of this paper at http://arxiv.org/abs/1709.00681.