TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Transactions on Mergeable Objects

Deepthi Akkoorath[1] Annette Bieniusa[1]

[1]University of Kaiserslautern
AG Software Technology

10. Nov 2015

Overview

Software Transactional Memory

Normal	STM
acquire_lock(x)	atomic
acquire_lock(y)	{
if (x > 0)	if (x > 0)
y++	y++
release_lock(x)	}
release_lock(y)	

Software Transactional Memory

Normal	STM
acquire_lock(x)	atomic
acquire_lock(y)	{
if (x > 0)	if (x > 0)
y++	y++
release_lock(x)	}
release_lock(y)	

- Composable
- Deadlock freedom
- Semantics (often): serializability

Software Transactional Memory

Pessimistic Approach

- Acquires exclusive access to data before processing transaction

Software Transactional Memory

Pessimistic Approach

- Acquires exclusive access to data before processing transaction

Optimistic Approach

- Execute updates and buffers writes (multi-versioning)
- Consistency check during commit
- Abort if check failed

Software Transactional Memory

Pessimistic Approach

- Acquires exclusive access to data before processing transaction

Optimistic Approach

- Execute updates and buffers writes (multi-versioning)
- Consistency check during commit
- Abort if check failed

More Conflicts $\rightarrow$ Performance degrade

Mechanisms to reduce conflicts

Delayed Operations

- Explicit delaying of some operations to commit time
- Transactional "RMW"

Mechanisms to reduce conflicts

Delayed Operations

- Explicit delaying of some operations to commit time
- Transactional "RMW"

Commutativity

Transactional Boosting [Herlihy/Koskinen (PPoPP'11), Hassan et al. (PPoPP'14)]

Mechanisms to reduce conflicts

Delayed Operations

- Explicit delaying of some operations to commit time
- Transactional "RMW"

Commutativity

Transactional Boosting [Herlihy/Koskinen (PPoPP'11), Hassan et al. (PPoPP'14)]

Alternative Programming Models

- RMW [Ruan et al. (TACO'15)]
- Twilight [Bieniusa et al.(PODC'10)]
- Concurrent revisions [Burckhardt et al.(ESOP'12)]

Mechanisms to reduce conflicts

Delayed Operations

- Explicit delaying of some operations to commit time
- Transactional "RMW"

Commutativity

Transactional Boosting [Herlihy/Koskinen (PPoPP'11), Hassan et al. (PPoPP'14)]

Alternative Programming Models

- RMW [Ruan et al. (TACO'15)]
- Twilight [Bieniusa et al.(PODC'10)]
- Concurrent revisions [Burckhardt et al.(ESOP'12)]

Mechanisms to reduce conflicts

Delayed Operations

- Explicit delaying of some operations to commit time
- Transactional "RMW"

Commutativity

Transactional Boosting [Herlihy/Koskinen (PPoPP'11), Hassan et al. (PPoPP'14)]

Alternative Programming Models

- RMW [Ruan et al. (TACO'15)]
- Twilight [Bieniusa et al.(PODC'10)]
- Concurrent revisions [Burckhardt et al.(ESOP'12)]

- Serializability $\rightarrow$ performance degradation
- Weaken Serializability, but no defined semantics

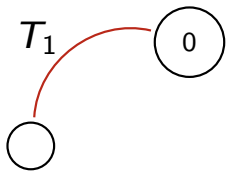
Weakly Consistent Transactions

On Shared Memory

- Shared Global Copy

Weakly Consistent Transactions

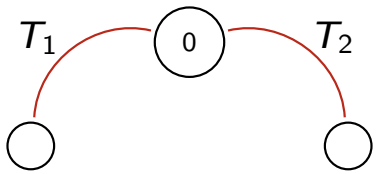
On Shared Memory



- Shared Global Copy
- Read from Consistent Snapshot

Weakly Consistent Transactions

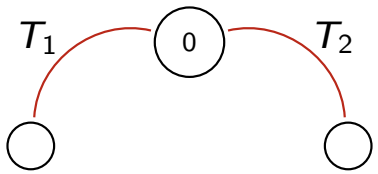
On Shared Memory



- Shared Global Copy
- Read from Consistent Snapshot

Weakly Consistent Transactions

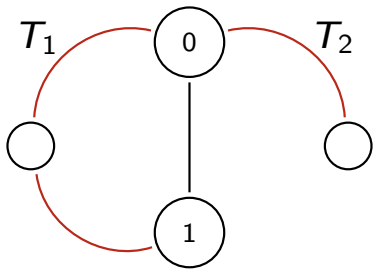
On Shared Memory



- Shared Global Copy
- Read from Consistent Snapshot
- Update thread local copy

Weakly Consistent Transactions

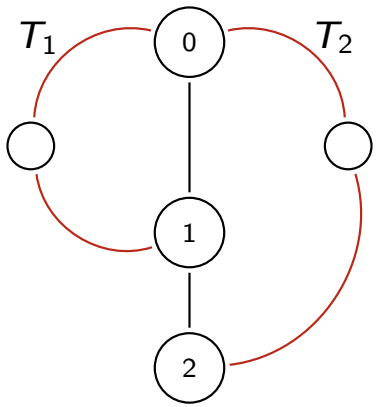
On Shared Memory



- Shared Global Copy
- Read from Consistent Snapshot
- Update thread local copy
- Commit $\rightarrow$ New Snapshot

Weakly Consistent Transactions

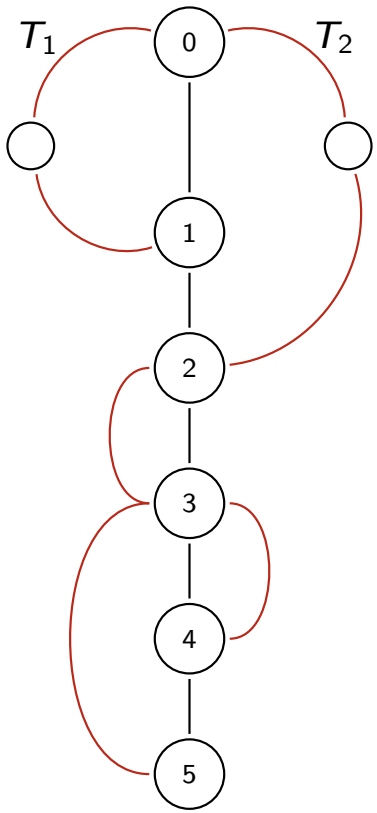
On Shared Memory



- Shared Global Copy
- Read from Consistent Snapshot
- Update thread local copy
- Commit $\rightarrow$ New Snapshot
  - Merge local to global copy

Weakly Consistent Transactions

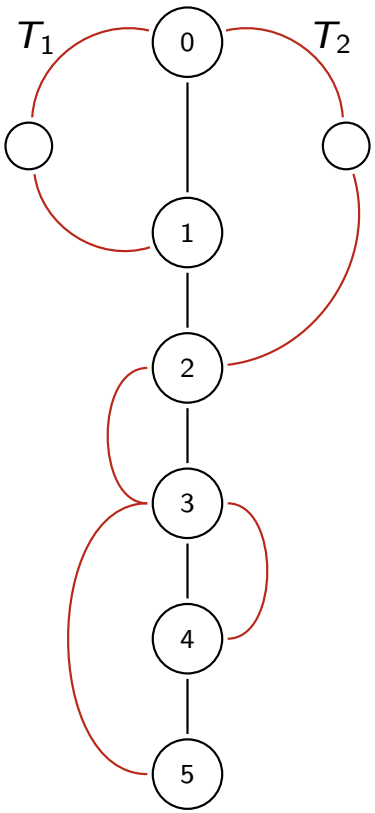
On Shared Memory



- Shared Global Copy
- Read from Consistent Snapshot
- Update thread local copy
- Commit $\rightarrow$ New Snapshot
  - Merge local to global copy

Weakly Consistent Transactions

On Shared Memory



- Shared Global Copy
- Read from Consistent Snapshot
- Update thread local copy
- Commit $\rightarrow$ New Snapshot
  - Merge local to global copy

- No aborts
  - Multi-versioning
  - Mergeable objects

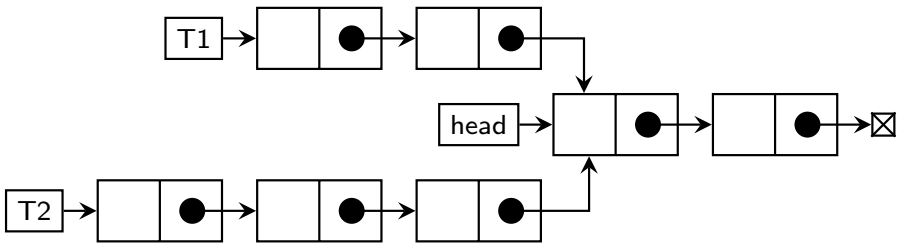
Mergeable Data Type

Mergeable Counter

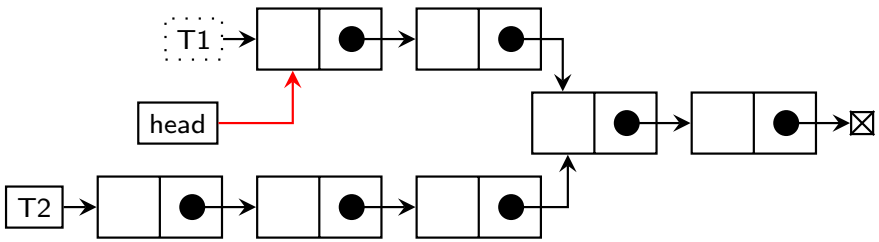
```
data Counter = Counter Int Int

incrBy :: Int → Counter → Counter
incrBy i (Counter g l) = Counter g (l+i)

value :: Counter → Int
value (Counter g l) = g+l

merge (Counter g _) (Counter _ l) = Counter (g+l) 0
```

Mergeable Bag

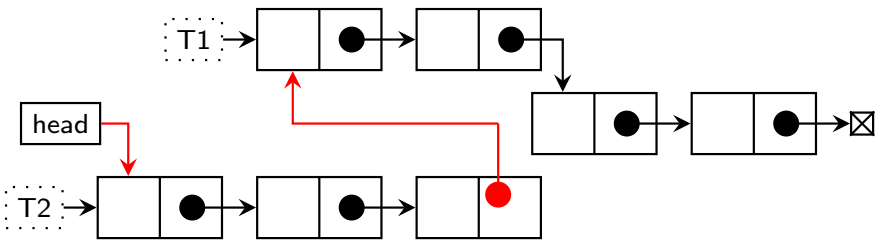
Add only



After $T1$ commits:



After $T2$ commits:



MTM: Transactions on Mergeable Objects

Semantics

```
addToBag(e, bag, size) {
   add(bag,e)
   incrBy(size,1)
}
```

Thread 1	Thread 2
startTxn {	startTxn {
addToBag(1, b, s)	addToBag(3, b, s)
addToBag(2, b, s)	addToBag(4, b, s)
print b	print b
}	}
endTxn	endTxn

MTM: Transactions on Mergeable Objects

Semantics

```
addToBag(e, bag, size) {
   add(bag,e)
   incrBy(size,1)
}
```

Thread 1	Thread 2
startTxn {	startTxn {
addToBag(1, b, s)	addToBag(3, b, s)
addToBag(2, b, s)	addToBag(4, b, s)
print b	print b
}	}
endTxn	endTxn
1, 2	3, 4

MTM: Transactions on Mergeable Objects

A Haskell Implementation

```
data MTM a = ...
data CVar a = ...

-- MTM Functions
eventually :: MTM a → IO a
newCVar :: Mergeable a ⇒ a → MTM (CVar a)
readCVar :: Mergeable a ⇒ CVar a → MTM a
modifyCVar :: Mergeable a ⇒ CVar a → (a → a) → MTM a

-- Mergeable Objects
class Mergeable a where
      merge :: a → a → a
```

MTM: Transactions on Mergeable Objects

Evaluation: Kmeans Clustering

```
function MAIN
    chunks=divide data into N chunks
    In Parallel
    for n in N do
        thread[n].CLUSTER(chunks[n])
    end for
end function
```

Atomic

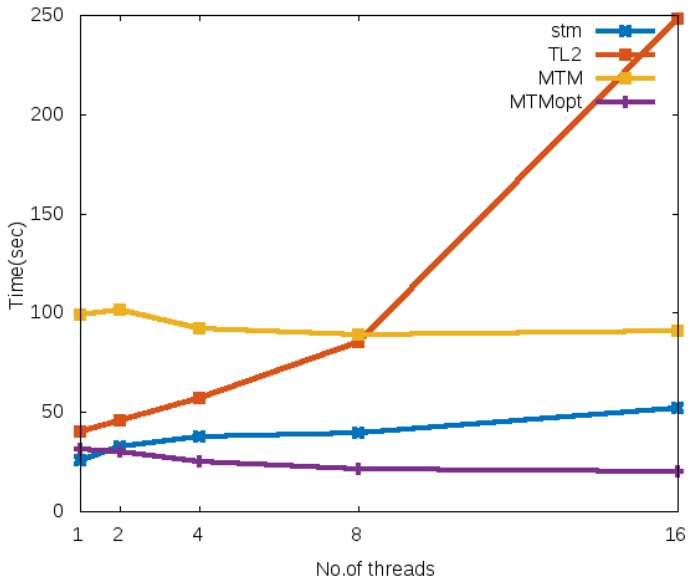
```
function CLUSTER(points)
    for p in points do
        index=findNearestCluster(clusters,p)
        beginTxn
            inc(new_centers_len[index],1)
            add(new_centers[index][0],p[0])
            add(new_centers[index][1],p[1])
        endTxn
    end for
end function
```

MTM-opt

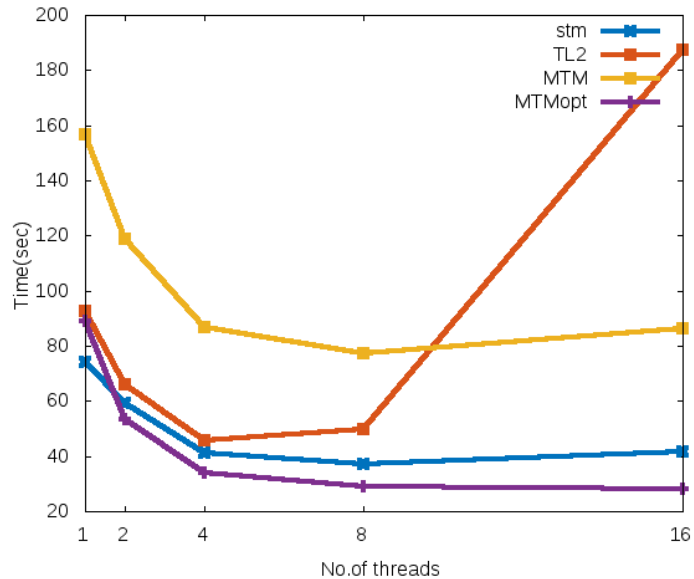
```
function CLUSTER(points)
    beginTxn
        for p in points do
            index=findNearestPoint(clusters,p)
            inc(new_centers_len[index],1)
            inc(new_centers[index][0],p[0])
            inc(new_centers[index][1],p[1])
        end for
    endTxn
end function
```

MTM: Transactions on Mergeable Objects

Evaluation: Kmeans clustering



High contention



Low Contention

MTM: Transactions on Mergeable Objects

Conclusion

- No conflicts $\rightarrow$ No aborts $\rightarrow$ More performance
- Longer transactions $\rightarrow$ Less synchronisation $\rightarrow$ More parallelism
- Weaker consistency semantics
- Requires efficient merge (space / time complexity)

Thank You!

MTM: Transactions on Mergeable Objects

A Haskell Implementation

```
data MTM a = ...
data CVar a = ...

-- MTM Functions
eventually :: MTM a → IO a
newCVar :: Mergeable a ⇒ a → MTM (CVar a)
readCVar :: Mergeable a ⇒ CVar a → MTM a
modifyCVar :: Mergeable a ⇒ CVar a → (a → a) → MTM a

-- Mergeable Objects
class Mergeable a where
      merge :: a → a → a
```

MTM: Transactions on Mergeable Objects

Example

```
addToBag :: Int → CVar (Bag Int) → CVar (Counter) → MTM [Int]
addToBag e bag size = do {
   ; b ← modifyCVar bag (add e)
   ; s ← modifyCVar size (incrBy 1)
   ; return (toList b) }
```

Thread 1	Thread 2
b1 ← eventually $ do	b2 ← eventually $ do
addToBag 1 b s	addToBag 3 b s
addToBag 2 b s	addToBag 4 b s
print b1	print b2

MTM: Transactions on Mergeable Objects

Example

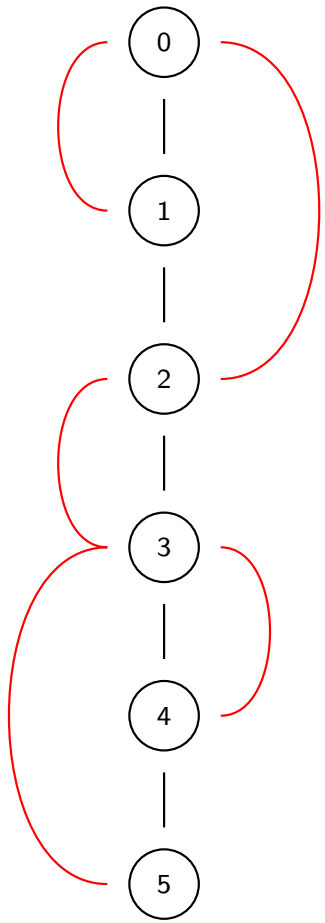
```
addToBag :: Int → CVar (Bag Int) → CVar (Counter) → MTM [Int]
addToBag e bag size = do {
   ; b ← modifyCVar bag (add e)
   ; s ← modifyCVar size (incrBy 1)
   ; return (toList b) }
```

Thread 1	Thread 2
b1 ← eventually $ do	b2 ← eventually $ do
addToBag 1 b s	addToBag 3 b s
addToBag 2 b s	addToBag 4 b s
print b1	print b2
1, 2	3, 4

MTM: Transactions on Mergeable Objects

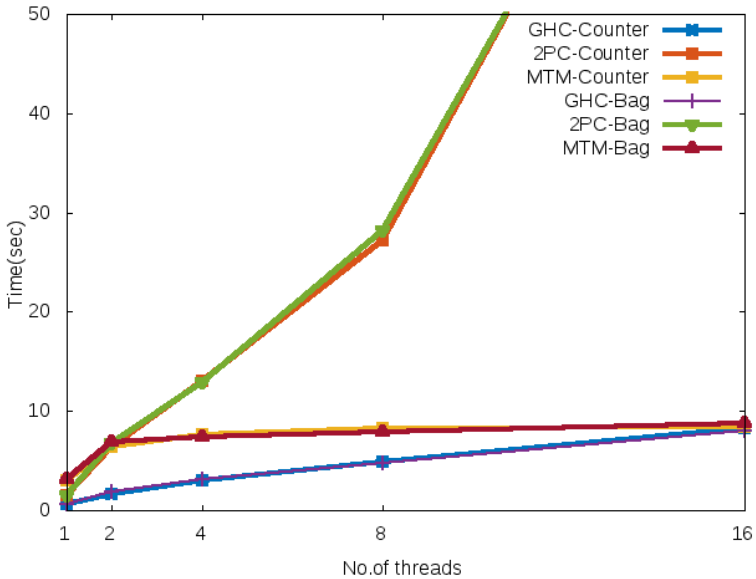
Algorithm

```
txn: {sid, Map writeset, Map readset}
var: {versions, lock}
versions : [{val, versionid}]
function BEGINTRANSACTION(txn)
    txn.sid ← globalclock
    txn.writeset ← ∅
    txn.readset ← ∅
end function
function COMMIT(txn)
    lockAll(txn.writeset)
    versionid ← globalclock++
    for all (var,val) ∈ txn.writeset do
        merge val to the latest version
    end for
    unlockAll(txn.writeset)
end function
function READVERSION(var, versionid)
    v ← var.versions
    if v.head.versionid ≥ vid then
        vr ← v.head
    else
        waituntil (not locked(var))
        vr ← var.versions.head
    end if
    while vr.versionid > vid do
        vr ← vr.next
    end while
    return vr.val
end function
```

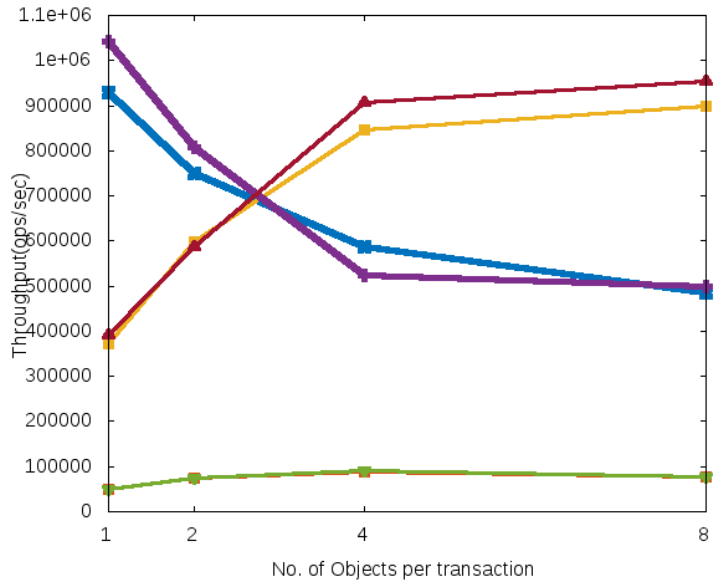


MTM: Transactions on Mergeable Objects

Evaluation: Micro benchmarks



Time vs No.of Threads



Throughput vs. No.of Objects