Objects with adaptive accessors to avoid STM barriers

F. Miguel Carvalho and João Cachopo

Bern, Switzerland, April 10, 2012
General Goal
Shared data
atomic
Overheads?
STM Barriers
Reduce Runtime Overheads

• Redo-log <vs> undo-log
• Eager <vs> lazy ownership acquisition
• Transactional versioning
• No ownership records
• Metatada in place
• Multi-versioning (e.g. JVSTM)
• ...
Good for read-only
Memory overheads
Winding path
Can we suppress these overheads?
AOM
Adaptive Object Metadata

...implemented in the JVSTM
box

:Counter
current:

:VBox
body:
versions’ history

The most recent committed value.
Transaction

```
:Counter
  current:

:VBox
  body:

:VBoxBody
  next:  ------
  value:  2
  version: 19

:VBoxBody
  next:  ------
  value:  1
  version: 17

:VBoxBody
  next:  null
  value:  0
  version: 13

:lastCommitted  23

:Transaction
  version: 18
  ...
```
Transaction

Transaction 18 reads the version 17
Shared Data
No contention
AOM

Compact

Extended
AOM

### Compact

<table>
<thead>
<tr>
<th>:SomeType</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>field x</td>
<td>32767</td>
</tr>
<tr>
<td>field y</td>
<td>34.7</td>
</tr>
<tr>
<td>field z</td>
<td>2147483647</td>
</tr>
</tbody>
</table>

### Extended

![Superman](image-url)

- Compact
- Extended
AOM

Compact

<table>
<thead>
<tr>
<th>Header</th>
<th>null</th>
</tr>
</thead>
<tbody>
<tr>
<td>field x</td>
<td>32767</td>
</tr>
<tr>
<td>field y</td>
<td>34.7</td>
</tr>
<tr>
<td>field z</td>
<td>2147483647</td>
</tr>
</tbody>
</table>

Extended
AOM

Compact

- **Header**
  - Field x: 32767
  - Field y: 34.7
  - Field z: 2147483647

null

Extended

- **Header**
  - Field x: 32767
  - Field y: 34.7
  - Field z: 2147483647

- **VBoxBody**
  - Version: 23
  - Values: Value of x, Value of y, Value of z

- **VBoxBody**
  - Version: 19
  - Values: Value of x, Value of y, Value of z
AOM

Compact

1. null

Extended

2. value of x value of y value of z

<table>
<thead>
<tr>
<th>Header</th>
<th>version values next</th>
</tr>
</thead>
<tbody>
<tr>
<td>32767</td>
<td></td>
</tr>
<tr>
<td>34.7</td>
<td></td>
</tr>
<tr>
<td>2147483647</td>
<td></td>
</tr>
</tbody>
</table>

field x
x of version 23

field y
y of version 23

field z
z of version 23
1. Extending

<table>
<thead>
<tr>
<th>header: null</th>
</tr>
</thead>
<tbody>
<tr>
<td>x: 3</td>
</tr>
<tr>
<td>y: 7</td>
</tr>
</tbody>
</table>
1. Extending

```
header:
  x: 3
  y: 7

:VBoxBody
  next: null
  version: 0
  value:
    :Integer
      value: 3
    :Integer
      value: 7
```

replicate()
1. Extending

```
header: null
x: 3
y: 7

VBoxBody
next: null
version: 17
value:

:Integer
value: 11

:Integer
value: 7

VBoxBody
next: null
version: 0
value:

:Integer
value: 3

:Integer
value: 7
```

replicate()
1. Extending
2. Reverting

```java
boolean tryRevert (AdaptiveObject o , VBoxBody body){
    if(o.readHeader() == body){
        o.toCompactLayout(body.value);
        return o.casHeaderWithNull(body);
    }
    return false;
}
```
abstract class AdaptiveObject<T extends AdaptiveObject<T>>{

    private VBoxBody<T> header;

    public abstract void toCompactLayout(T from);

    public VBoxBody<T> readHeader(){
        return header;
    }

    public boolean casHeaderWithNull(VBoxBody<T> expected){
        return UtilUnsafe.UNSAFE.compareAndSwapObject(this,header__ADDRESS__, expected, null);
    }
}

boolean tryRevert (AdaptiveObject o , VBoxBody body){
    if(o.readHeader() == body){
        o.toCompactLayout(body.value);
        return o.casHeaderWithNull(body);
    }
    return  false;
}
abstract class AdaptiveObject <T extends AdaptiveObject {

    private static final long header__ADDRESS__;
    private VBoxBody<T> header;

    public abstract T replicate();
    public abstract void toCompactLayout(T from);
    public VBoxBody<T> readHeader(){
        return header;
    }

    public boolean casHeaderWithNull(VBoxBody<T> expected){
        return UtilUnsafe.UNSAFE.compareAndSwapObject(this,header__ADDRESS__, expected, null);
    }

    public boolean casHeader(VBoxBody<T> expected, VBoxBody<T> newBody){
        return UtilUnsafe.UNSAFE.compareAndSwapObject(this, header__ADDRESS__, expected, newBody);
    }
}

AdaptiveObject
hierarchy
hierarchy

Object

AdaptiveObject

...
AOM

• 1st release (Multiprog 12)
  – implemented with the JVSTM lock based
  – reversion and extension operations specified by an AdaptiveObject interface

• 2nd release:
  – Implemented with the JVSTM lock free
  – AdaptiveObject as the root base class
  – provides a Transparent API (like Deuce STM)
AOM with JVSTM lock based

- increases the speedup between 13% and 35%
  (* Multiprog12)
new AOM with JVSTM lock free

- increases the speedup between 5% and 36%
STAMP Vacation, low++ & long trxs & RO

• Low contention
• ++, large data sets
• -n = 256, longer transactions, instead of the recommendation 2 or 4
• 3 kinds of transactions:
  – Delete and create items: car, flight or room
  – Remove defaulter clients (bill > 0)
  – Query and reserve an item: car, flight or room

Splitted in 2 transactions: RO + RW
STAMP Vacation, low++ & long trxs & RO

- increases the speedup between 18% and 37%
- Maximum speedup = 4.32
Comparing with the Deuce STM...

and enhancing the AOM with a transparent API
STAMP Vacation, low++ & long trxs & RO

- Maximum speedup = 3,83  (< 4,32 with a non-transparent API)
- Still better than the Deuce STM with TL2
- Maximum speedup = 1.92
- Still better than JVSTM and the Deuce TL2
Future Work
Future Work

• An improved reversion algorithm

• New design for AOM that keeps the contention-free execution path without any barrier or validation

• Integrate the AOM compiler in the implementation of the Deuce STM