Short-term Memory for Self-collecting Mutators

Martin Aigner, Andreas Haas, Christoph M. Kirsch, Michael Lippautz, Ana Sokolova, Stephanie Stroka, Andreas Unterweger

University of Salzburg

October 12, 2010
Heap Management
Heap Management

heap needed
Heap Management
Heap Management

needed

reachables

heap
Heap Management

- Needed
- Reached
- Unreachable

Heap Management Diagram:
Heap Management

- Explicit memory management deallocates here.
- Needed and reachable areas within the heap.
- Unreachable areas outside the heap.
Heap Management

- memory leaks

explicit memory management deallocates here

needed reachable

unreachable

heap
Heap Management

- memory leaks
- dangling pointers
Heap Management

- memory leaks
- dangling pointers
Heap Management

- memory leaks
- dangling pointers
- tracing
Heap Management

- memory leaks
- dangling pointers
- tracing
- reference counting
Heap Management

- memory leaks
- dangling pointers

- tracing
- reference counting
- reachable memory leaks
Persistent Memory Model

- Allocated memory objects are guaranteed to exist until deallocation.
- Explicit deallocation is not safe (dangling pointers) and can be space-unbounded (memory leaks).
- Implicit deallocation (unreachable objects) is safe but may be slow or space-consuming (proportional to the size of live memory) and can still be space-unbounded (memory leaks).
Short-term Memory

- Memory objects are only guaranteed to exist for a finite amount of time
- Memory objects are allocated with a given expiration date
- Memory objects are neither explicitly nor implicitly deallocated but may be refreshed to extend their expiration date
With short-term memory programmers or algorithms specify which memory objects are still needed and not which memory objects are not needed anymore!
Short-term Memory

- Needed
- Reached
- Not expired
- Unreachable
Short-term Memory

- needed
- reachable
- not expired

heap

conservative refresh
Short-term Memory

conservative expiration

conservative refresh

heap

not expired

needed

reachable
Short-term Memory - Sources of Errors

- Memory leaks
  - When not-needed objects are continuously refreshed
  - When time does not advance

- Dangling Pointers
  - When needed objects are not refreshed
Self-collecting Mutators
Programming Model

- Explicit memory management
  - The programmer (or an algorithm) adds memory management calls to the program code

- Hybrid approach for backward compatibility
  - Per default objects are allocated as persistent and managed by the existing memory management (malloc/free, garbage collection)
  - A refresh-call makes an object short-term, e.g. the objects get an expiration date
Programming Model - Expiration Date

- An object gets an expiration date when it gets refreshed and is then managed by our system
- A programmer refreshes objects explicitly
  - Every refresh-call creates a new expiration date for an object
  - The object expires when all its expiration dates are expired
A software clock is used for object expiration

- An integer counter which is increased by tick-calls
- An expiration date has expired when its value is less than the time of the software clock

Every thread has its own thread-local clock

- Expiration dates expire according to the clock of the thread which created the expiration date
Example - Monte Carlo

```java
monteCarlo(int repetitions) {
    Vector results = new Vector(repetitions);
    for (int i = 0; i < repetitions; i++) {
        RandomWalk walk = createRandomWalk();
        results.add(doCalculation(walk));
    }
    evaluateResults(results);
}
```
Example - Monte Carlo

```java
monteCarlo(int repetitions)
{
    Vector results = new Vector(repetitions);
    for(int i = 0; i < repetitions; i++)
    {
        RandomWalk walk = createRandomWalk();
        SCM.refresh(walk, 0);
        results.add(doCalculation(walk);
        SCM.tick();
    }
    evaluateResults(results);
}
```
Example - x264 Video Encoder

input frames

concurrent reference buffers

output

1. malloc
2. refresh
3. processing unit
4. push_unused
5. pop_unused
6. tick
7. or concurrent

Note: The diagram shows a sequence of operations involving frames, buffers, and a processing unit, illustrating the flow of data in an x264 video encoder.
### Other Use Cases

<table>
<thead>
<tr>
<th>benchmark</th>
<th>LoC</th>
<th>tick</th>
<th>refresh</th>
<th>free</th>
<th>aux</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpg123</td>
<td>16043</td>
<td>1</td>
<td>0</td>
<td>(-)43</td>
<td>0</td>
<td>44</td>
</tr>
<tr>
<td>JLayer</td>
<td>8247</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>Monte Carlo</td>
<td>1450</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>LuIndex</td>
<td>74584</td>
<td>2</td>
<td>15</td>
<td>0</td>
<td>3</td>
<td>20</td>
</tr>
</tbody>
</table>

**Table:** Use cases of short-term memory: lines of code of the benchmark, number of tick-calls, number of refresh-calls, number of free-calls, number of auxiliary lines of code, and total number of modified lines of code.
Implementation
Our implementation is called self-collecting mutators (SCM)

- The threads (mutators) of a program collect their expired objects by themselves
- At memory management calls a constant number of expired objects are collected

We have implementations in C, Java and Go

- The C implementation is based on ptmalloc2
- The Java implementation is based on the Jikes RVM
- For the Go implementation we extended the 6g Go runtime

Available at: tiptoe.cs.uni-salzburg.at/short-term-memory
An Object can have multiple expiration dates

- An expiration date is represented by a descriptor, which stores the expiration date and a pointer to the object

- For each expiration date of an object there exists one descriptor

- Every object contains a descriptor counter (1 word) in its header which counts the number of descriptors pointing to it
Implementation - Descriptors Buffer

time=4

expired descriptor

not-expired descriptor
Implementation - Descriptor Buffer

time=4

expired descriptor list

4
5
6
7
Implementation - Descriptor Buffer

time = 4
maximal expiration extension = 3

4 mod 4 = 0
5 mod 4 = 1
6 mod 4 = 2
7 mod 4 = 3

expired descriptor list
The malloc/new call of SCM increases the requested amount of memory by one word and uses the underlying memory management then to allocate the memory.

- In C it is `ptmalloc2`
- In Java and Go it is a mark-sweep garbage collector

The additional header word of an object is used for the descriptor counter.

The descriptor counter is initialized with zero.
Implementation - refresh

**Definition**

refresh(object, extension)

- The refresh-call creates a new descriptor for the considered object.
- The refresh-call consists of four operations:
  1. Increase the descriptor counter of the specified object.
  2. Create a new descriptor and store it in the descriptor buffer which corresponds to the given expiration extension.
  3. (Self-collection) Remove one descriptor from the expired_descriptors_list.
  4. Decrease the descriptor counter of the object which corresponds to that descriptor.

  - If the descriptor counter gets zero, the object is deallocated.
Before refresh

time=4

refresh(object, 0);

4 mod 4 = 0
5 mod 4 = 1
6 mod 4 = 2
7 mod 4 = 3

expired descriptor list
After refresh

time=4

refresh(object, 0);

4 mod 4 = 0
5 mod 4 = 1
6 mod 4 = 2
7 mod 4 = 3

add one descriptor

remove one descriptor
Implementation - tick

Definition

tick()

- The tick-call increases the thread-local time
- The descriptor list which expires by that time advance is appended to the expired descriptor list
Before tick

time=4

4 mod 4 = 0
5 mod 4 = 1
6 mod 4 = 2
7 mod 4 = 3

expired descriptor list
After tick

time=5

8 mod 4 = 0
5 mod 4 = 1
6 mod 4 = 2
7 mod 4 = 3

expired descriptor list

[Diagram of descriptors expiring]
Implementation - free

**Definition**

free(object)

- In C it is still possible to use the free-call for explicit deallocation
- When the descriptor counter of the object is zero, the object is deallocated
- Otherwise nothing is done
  - The object will be deallocated when its last descriptor expires
Implementation - Garbage Collector

- All objects are considered to compute reachability
- Short-term objects are not deallocated
  - They will be deallocated when their last descriptor expires
Experiments
What we want to show

- **In C**
  - Short-term memory is easier to use while not losing temporal performance and with low memory overhead

- **In Java and Go**
  - Short-term memory is more difficult to use but improves temporal performance (e.g. reduce the number of garbage collection runs)
Experiments - System Configuration

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>2x AMD Opteron DualCore, 2.0 GHz</td>
</tr>
<tr>
<td>RAM</td>
<td>4GB</td>
</tr>
<tr>
<td>OS</td>
<td>Linux 2.6.32-21-generic</td>
</tr>
<tr>
<td>Java VM</td>
<td>Jikes RVM 3.1.0</td>
</tr>
<tr>
<td>C compiler</td>
<td>gcc version 4.4.3</td>
</tr>
<tr>
<td>C allocator</td>
<td>ptmalloc2-20011215 (glibc-2.10.1)</td>
</tr>
</tbody>
</table>

Table: System configuration.
Monte Carlo - Total Runtime

Figure: Total execution time of the Monte Carlo benchmarks in percentage of the total execution time of the benchmark using self-collecting mutators.
Monte Carlo - Latency and Free Memory

Figure: Free memory and loop execution time of the fixed Monte Carlo benchmark.
Monte Carlo - Tick Frequency - Latency

Figure: Loop execution time of the Monte Carlo benchmark with different tick frequencies. Self-collecting mutators is used.
Monte Carlo - Tick Frequency - Free Memory

Figure: Free memory of the Monte Carlo benchmark with different tick frequencies. Self-collecting mutators is used.
Table: Total execution times of the mpg123 benchmark averaged over 100 repetitions. Here, SCM($n, m$) stands for self-collecting mutators with a maximal expiration extension of $n$ and descriptor page size $m$. 

<table>
<thead>
<tr>
<th>Service</th>
<th>Execution Time</th>
<th>Runtime %</th>
</tr>
</thead>
<tbody>
<tr>
<td>ptmalloc2</td>
<td>895.25ms</td>
<td>100.00%</td>
</tr>
<tr>
<td>ptmalloc2 through SCM</td>
<td>899.43ms</td>
<td>100.47%</td>
</tr>
<tr>
<td>SCM(1, 256B)</td>
<td>890.18ms</td>
<td>99.43%</td>
</tr>
<tr>
<td>SCM(10, 256B)</td>
<td>898.28ms</td>
<td>100.34%</td>
</tr>
<tr>
<td>SCM(1, 4KB)</td>
<td>892.18ms</td>
<td>99.66%</td>
</tr>
<tr>
<td>SCM(10, 4KB)</td>
<td>892.28ms</td>
<td>99.67%</td>
</tr>
</tbody>
</table>
mpg123 - Memory Consumption

Memory overhead and consumption of the mpg123 benchmark. Again, SCM($n$, $m$) stands for selfcollecting mutators with a maximal expiration extension of $n$ and descriptor page size $m$. We write space-overhead($n$, $m$) to denote the memory overhead of the SCM($n$, $m$) configurations for storing descriptors and descriptor counters.
Thank You

check out:

eurosys2011.cs.uni-salzburg.at