Accurate and Efficient Object Tracing for Java Applications

Philipp Lengauer
Verena Bitto

2015-02-02
What if …

… we would know all there is to know about every object?

```java
public void foo() {
    String[] object = {"Tracing", "rocks"};
}
```

… then we could reproduce the entire heap for every point in time and do offline analysis!
State of the Art – An Unfair Comparison

**Instrumentation based**
- changes application behavior
- introduces large overhead
- unable to reconstruct heap in detail

avg(overhead) = 1054 %

avg(overhead) = 14 % (7 %)
Approach

```java
class A {
    ...
    new Foo();
    ...
}
```
Allocation Events

Slow allocation event

→ 8 – 16 bytes per allocation
Allocating Subsystems

![Bar chart showing the allocation of subsystems for various benchmarks.](image)

Benchmarks: aurora, baix, eclipse, top, h2, arthon, lujex, lusearch, pmid, simflow, tomcat, tradear, trademoop, xalan.

Legend: C1, C2, IR, VM, Dependencies.
Object Types

![Bar Chart of Object Types](chart-image)
Allocations in JVMs

Addresses of objects that are allocated into a TLAB are computable offline!

\[
\text{addr}(o_n) = \begin{cases} 
\text{addr}(\text{TLAB}(o_{n-1})) & \text{if } n = 1 \\
\text{addr}(o_{n-1}) + \text{size}(o_{n-1}) & \text{else}
\end{cases}
\]
Allocation Events Revisited

**Slow allocation event**

- **event type**
- **allocation site**
- **relative address**
- **space**
- **array length**
- **class**

- **mode → event type**
- **class → allocation site**
- **size → class + length**

**Fast allocation event**

- **event type**
- **allocation site**
- **array length**

- **address → previous events + TLAB information**

- → 4 bytes per allocation
- → **computable at compile-time (JIT)**
Firing a Fast Allocation Event

is as easy as ...

```c
const int event = 0xABCDEF00;

if(buffer->top + 1 <= buffer->end) {
    *(buffer->top++) = event;
} else {
    fire_event_slow(event);
}
```
Instrumenting Allocation Sites

**Java Source Code**
```java
String foo() {
    return new String();
}
```

**Scala Source Code**
```scala```
def foo() : String = new String
```

**Java Bytecode**
```
new java.lang.String
dup
invokespecial <init>
ret
```

**Machine Code**
```
Allocation
Call
Log event
```

**Sea of Nodes**
- Allocation node
- Call node
- Log event node

**Client Compiler**

**Server Compiler**

**HIR/LIR**
- Allocation instruction
- Call instruction

**Instrument**
Move Events

Slow move event

If there is no move event for an object in a collected space, it has been deallocated.
Minor GCs

Eden — Survivor to — Survivor from — Old

Move by GC-Thread 1
Move by GC-Thread 2

PLAB
Move Events Revisited (for Minor GCs)

**Slow move event**

- Event type
- From address
- To address

**Fast move event**

- Event type
- From address

**Fast narrow move event**

- Event type
- From address
Major GCs

Claim: objects live and die in groups due to their sequential allocation
Move Events Revisited (for Major GCs)

- **Slow move event**
  - Event type
  - From address
  - To address

- **Region move event**
  - Event type
  - From address
  - To address
  - Object count: ~312 objects per event
  - Size change: (3.65 Kb -> 12 b)
Average Event Size [bytes]
Overhead (DaCapo, DaCapo Scala, SPECjvm, SPECjbb)
**Conclusion**

**Very low overhead (7 %)**

**Analyse trace and rebuild heap offline**

**Compact event formats, ...**

**Proof of concept – Ant Tracks**