Reference Object Processing in On-The-Fly Garbage Collection

Tomoharu Ugawa, Kochi University of Technology
Richard Jones, Carl Ritson, University of Kent
Weak Pointers

- Weak pointers are a mechanism to allow mutators to communicate with GC
- `java.lang.ref.Reference`
  - Specification requires us to process weak references atomically from the view point of mutators
- Fully concurrent (on-the-fly) GC never stops all mutators
Java reference types

- **Strong** - usual references.
- **Soft** - used for caches that the GC can reclaim.
- **Weak** - used for canonicalize mappings (e.g., interned strings) that do not prevent GC from reclaiming their keys or values.
- **Phantom** - used for scheduling pre-mortem cleanup actions more flexibility than finalisers.
Java reference types

- **Strong** - usual references.
- **Soft** - used for caches that the GC can reclaim.
  - reduce to strong/weak references
- **Weak** - used for canonicalize mapping (e.g., interned strings) that do not prevent GC from reclaiming their keys or values.
- **Phantom** - used for scheduling pre-mortem cleanup actions more flexibility than finalisers.
  - no interaction with mutators
Reachability

Strongly - can be reached without traversing any other references.
Weakly - not strongly reachable but can be reached by traversing weak references.

• No formal specification
  • Specification is written in English.
  • There are errors in implementations
• We formalised the specification
The GC finds all strongly reachable objects and reclaims others.

The GC “clears” references whose referents are weakly reachable.

- Weak reference to Strongly - to be retained
- Weak reference to Weakly - to be cleared
Reference.get() - returns a strong reference to its target or null if the GC has cleared.

get() may make some objects that were weakly reachable strongly reachable.
Reference.get() - returns a strong reference to its target or null if the GC has cleared.

get() may make some objects that were weakly reachable strongly reachable.
Race
Race

- Collector clears weak reference $A$
Race

- Collector clears weak reference $A$
• Collector clears weak reference $A$
• Mutator makes $O$ strongly reachable by creating a strong reference to the upstream
from the OpenJDK mailing list

“I've been tuning a Java 7u51, Solaris 10, T4 system with 24G heap. My customer is not very happy with the remark pauses of up to 2 seconds.” Thomas Viessmann

“It looks like the application is using a lot of Reference objects. The time spent in remark is dominated by reference processing.” Bengt Rutisson
Solutions

- **Stop the world** - Pauseless GC [Click et al., 2005], Staccato [McCloskey et al., 2008]
  - Stop all mutators and process references

- **Lock** - “On-the-fly” GC [Domani et al., 2000]
  - Block any mutator that calls get()

- **On-the-fly** - Metronome-TS [Auerbach et al., 2008]
  - Implementation technique is not public
Global GC State

GC and mutator race in TRACING
- GC want to finish tracing and then start clearing
- Mutator force GC to do more work

Diagram:
- NORMAL \(\xrightarrow{\text{start tracing}}\) TRACING
- REPEAT \(\xrightarrow{\text{by collector}}\) TRACING
- REPEAT \(\xrightarrow{\text{by collector (atomic)}}\) TRACING
- REPEAT \(\xrightarrow{\text{by mutator (atomic)}}\) TRACING
- TRACING \(\xrightarrow{\text{start tracing}}\) TRACING
- TRACING \(\xrightarrow{\text{no tracing work}}\) CLEARING
- CLEARING

GC traces strongly reachable objects
- GC traverses strong references
  - to colour strongly reachable objects black
- Write barrier
- insertion barrier [Dijkstra]
- deletion barrier (a.k.a. snapshot) [Yuasa]
- Read barrier for Reference.get()
TRACING State

- GC traverses strong references
  - to colour strongly reachable objects black
- Write barrier
  - insertion barrier [Dijkstra]
  - deletion barrier (a.k.a. snapshot) [Yuasa]
- Read barrier for Reference.get()
Insertion Barrier

INVARIANT: Root is grey (allows root to refer to white objects)

- GC repeat tracing until it finds root black
- Reference.get() changes the state to REPEAT to notify the GC that the root may not be black
Deletion Barrier

INVARIANT: Root is black --- root is never rescanned

- Reference.get() colours target grey
- Reference.get() changes the state to REPEAT to notify the GC that the root may not be black
Once GC enters CLEARING state

- Reference.get() returns null if its target is white
- no more objects become strongly reachable
- GC clears weak references whose targets are white
Evaluation

- Jikes RVM
  - Sapphire on-the-fly copying collector
  - Trigger GC immediately after the previous GC completes
- Configuration
  - Core i7-4770 (4-core, 3.4 GHz)
  - 1 GB heap
  - 2 collector threads
Pause Time distribution

- Stop-the-world GC, or
- Block mutator.get() with “lock”
Pause Time distribution

Pause time

STW
lock ins
lock del

xalan2009
Reference Processing

Phase Time

Mutators blocked

Mutators running

STW
lock ins
lock del

avrora2009

lusearch2009

xalan2009

Mutators blocked

STW
lock ins
lock del

avrora2009

lusearch2009

xalan2009

Mutators running

OTF ins
OTF del

avrora2009

lusearch2009

xalan2009
Execution times

- xalan9
- sunflow9
- pmd9
- lusearch9
- luindex9
- avrora9
- jython6

OTF del
LOCK del
STW
Conclusion

- Reference types are frequently used in a significant number of programs.
  - *On-the-fly GC must not ignore reference types.*
- Formalised the definition of reference types.
- On-the-fly reference processing.
  - Model checked with SPIN.
  - Implemented in Jikes RVM.
  - *On-the-fly reference processing phases are longer in the worst case, but with deletion barrier, not by much.*
  - *Overall execution time is not increased significantly by processing references on-the-fly, and is often reduced.*

http://github.com/perlfu/sapphire
Questions?
Reference type usage

- xalan2009: T = 3790ms
- sunflow2009: T = 4480ms
- pmdd2009: T = 3005ms
- lusearch2009: T = 4139ms
- luindex2009: T = 2651ms
- Jython2006: T = 13639ms
- avrora2009: T = 5112ms

x-axis: Normalised execution time

Y-axis: Calls to get() per msec x 10^3
Model Checking

- Model checked with SPIN
- Correctness: appears to mutators to be processed by GC atomically
- Terminates only with deletion barrier
Properties

- No dangling pointer is created
- If a variable is not null, its target has not been reclaimed

\[ P1 \quad \square((x \neq \text{NULL}) \implies (\text{mark}[x] \neq \text{RECLAIMED})) \]

- Once get() of a Reference returns null, it will never returns its target

\[ P2 \quad \square(\text{RETNULL}_i \implies \neg \Diamond(x = i)) \quad (i = 1, 2, 3) \]

![Diagram showing reference and normal objects]

- Reference objects
- Normal objects

\[ x \quad \text{root} \]
Race

- Mutator makes an object strongly reachable
- Collector clears weak reference
Answer

- Handshake ensure that GC changes to TRACING after the get() that change the state to REPEAT returned.
- Second handshake ensures all mutators acknowledge GC is in TRACING.
- CAS tells which thread won the race.