Prioritized Garbage Collection
Using the Garbage Collector to Support Caching

Diogenes Nunez, Samuel Z. Guyer, Emery D. Berger

Tufts University, University of Massachusetts Amherst

November 2, 2016
Caches are everywhere
Caches are everywhere
Caches are everywhere
Caches are everywhere
Caches are a space-time tradeoff.
Caches are a space-time tradeoff.
Caches are a space-time tradeoff.
Caches are a space-time tradeoff?
Garbage collection dominates runtime.
Perfect Cache Size
Elements can be too small.

- The cache can use more space.
Elements can be too big.

- The cache needs to use less space.
Evicted elements remain in memory until next GC.
Evicted elements remain in memory until next GC.
Evicted elements remain in memory until next GC.
Problem

Can we . . .

- Keep the garbage collector from dominating program run time
- Handle entries of various sizes
Soft References

- Removed prior to an `OutOfMemory` exception
- JVMs can remove them during normal operation
  - Hotspot removes them over time in a Least-Recently-Used (LRU) order
All soft references are equal.

- All Soft References placed in a global queue
- Queue is cleared in an LRU policy over time
- Results in removal of cache values from cold code
Our Solution

Make the Garbage Collector aware of the cache and its contents
Our Solution

- Prioritized Garbage Collector
  - Separate the references into logical spaces with unique eviction policies
  - Compute how much memory each reference is responsible for

- Sache
  - Use the Prioritized Garbage Collector to limit the memory used by stored values
Prioritized Garbage Collection: Using PrioSpaces

// Specify memory limit in number of bytes...
PrioSpace<Tree> space = VM.getSpace(500);

// .. or percentage of heap
PrioSpace<Tree> space = VM.getSpace(0.6);
Register objects into a PrioSpace

```
Tree t;
PrioReference<T> ref = space.newReference(t);
```
Updates the reference’s priority every use.

```
PrioReference<Tree> ref;
Tree t = ref.get();
/* Results in a call to
   * space.updateReference(t);
   */
```
Compare two PrioReferences

```java
int compare(PrioReference<T> a, PrioReference<T> b);
```

- Determines the eviction policy of the space given by programmer

Higher priority → More likely to not be evicted
Compare two PrioReferences

```c
int compare(PrioReference<T> a, 
PrioReference<T> b);
```

- Determines the eviction policy of the space given by programmer
- Determines the priority of a reference in the space
  - Higher priority → More likely to *not* be evicted
For each PrioSpace, garbage collector calculates and stores the amount of memory used by structures

Garbage collector frees structures according to priority of the references
Sache (Space Aware Cache)

- User puts a limit on size of Sache in bytes
- Tell the prioritized garbage collector what values are in the cache and the order of eviction
  - Current eviction policy is LRU
- Measure the amount of memory used by these values during garbage collection
- If keeping a value means the Sache exceeds the size limit, evict that value
Sache: Interface

class Sache <K,V>
    extends HashMap<K, PrioReference<V>>
{
    protected int priority;
    protected PrioSpace<V> priospace;

    public Sache(long maxSize);

    public boolean put(K key, V value);
    public V get(K key);
    public V remove(K key);

    private void update();
}
Sache: Using Sache

```java
Sache map = new Sache(60);
map.put("1", d);
map.put("2", a);
map.put("3", h);
map.get("1");
//Garbage collection occurs
```

Marked by Mark Sweep

Marked by Prioritized GC
Sache: Using Sache

Sache map = new Sache(60);
map.put("1", d);
map.put("2", a);
map.put("3", h);
map.get("1");
//Garbage collection occurs

limit = 60 bytes
space:

Marked by Mark Sweep
Marked by Prioritized GC
Sache: Using Sache

```java
Sache map = new Sache(60);
map.put("1", d);
map.put("2", a);
map.put("3", h);
map.get("1");
//Garbage collection occurs
```

Limit = 60 bytes

Space:

Marked by Mark Sweep

Marked by Prioritized GC
Sache: Using Sache

Sache map = new Sache(60);
map.put("1", d);
map.put("2", a);
map.put("3", h);
map.get("1");
//Garbage collection occurs

limit = 60 bytes
space: d

Marked by Mark Sweep
Marked by Prioritized GC
Sache: Using Sache

```java
Sache map = new Sache(60);
map.put("1", d);
map.put("2", a);
map.put("3", h);
map.get("1");
//Garbage collection occurs
```

limit = 60 bytes
space: `d`

Marked by Mark Sweep
Marked by Prioritized GC
Sache: Using Sache

```java
Sache map = new Sache(60);
map.put("1", d);
map.put("2", a);
map.put("3", h);
map.get("1");
//Garbage collection occurs
```

Limit = 60 bytes

Space:

```
a
d
```

Marked by Mark Sweep

Marked by Prioritized GC
Sache: Using Sache

Sache map = new Sache(60);
map.put("1", d);
map.put("2", a);
map.put("3", h);
map.get("1");
// Garbage collection occurs

limit = 60 bytes
space: [a d]

Marked by Mark Sweep
Marked by Prioritized GC
Sache: Using Sache

Sache map = new Sache(60);
map.put("1", d);
map.put("2", a);
map.put("3", h);
map.get("1");
//Garbage collection occurs

limit = 60 bytes
space: h a d

c
map
a
b
d
f
e
h
i
g

Marked by Mark Sweep
Marked by Prioritized GC
Sache: Using Sache

Sache map = new Sache(60);
map.put("1", d);
map.put("2", a);
map.put("3", h);
map.get("1");
//Garbage collection occurs

limit = 60 bytes
space: [h, a, d]

Marked by Mark Sweep
Marked by Prioritized GC
Sache: Using Sache

Sache map = new Sache(60);
map.put("1", d);
map.put("2", a);
map.put("3", h);
map.get("1");
//Garbage collection occurs

limit = 60 bytes
space: [d h a]

Marked by Mark Sweep
Marked by Prioritized GC
Sache: Using Sache

Sache map = new Sache(60);
map.put("1", d);
map.put("2", a);
map.put("3", h);
map.get("1");
//Garbage collection occurs

limit = 60 bytes
space: [d, h, a]

Marked by Mark Sweep

Marked by Prioritized GC
Prioritized GC: Measuring Sizes

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

limit = 60 bytes
current-total = 0 bytes

space: d h a

Marked by Mark Sweep
Marked by Prioritized GC
Mark entries in space
Mark roots
Depth-first search from roots
Depth-first search from space entries
Free all unmarked objects

Marked by Mark Sweep
Marked by Prioritized GC

Marked by Mark Sweep
Marked by Prioritized GC

limit = 60 bytes
current-total = 0 bytes
space: [d h a]

D. Nunez, S. Guyer, E. Berger
Prioritized GC
November 2, 2016
Prioritized GC: Measuring Sizes

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

Limit = 60 bytes
Current-total = 0 bytes
Space: d h a

Marked by Mark Sweep
Marked by Prioritized GC
Prioritized GC: Measuring Sizes

- **Mark entries in space**
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

Marked by Mark Sweep

Marked by Prioritized GC

limit = 60 bytes
current-total = 0 bytes
space: [d h a]

D. Nunez, S. Guyer, E. Berger
Prioritized GC: Measuring Sizes

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

Limit = 60 bytes
Current-total = 0 bytes
Space: d h a

Marked by Mark Sweep
Marked by Prioritized GC
Prioritized GC: Measuring Sizes

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

Marked by Mark Sweep

Marked by Prioritized GC
Mark entries in space

Mark roots

Depth-first search from roots

Depth-first search from space entries

Free all unmarked objects

limit = 60 bytes
current-total = 0 bytes

Marked by Mark Sweep

Marked by Prioritized GC
Prioritized GC: Measuring Sizes

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

limit = 60 bytes
current-total = 0 bytes

space: [d h a]

Marked by Mark Sweep
Marked by Prioritized GC
Prioritized GC: Measuring Sizes

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

Marked by Mark Sweep

Marked by Prioritized GC

limit = 60 bytes
current-total = 0 bytes

space: d h a
Prioritized GC: Measuring Sizes

- Mark entries in space
- Mark roots
- **Depth-first search from roots**
- Depth-first search from space entries
- Free all unmarked objects

Marked by Mark Sweep
- Marked by Prioritized GC

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```

```
limit = 60 bytes
current-total = 0 bytes
space: d h a
```
Prioritized GC: Measuring Sizes

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

Marked by Mark Sweep
Marked by Prioritized GC

group limit = 60 bytes
current-total = 0 bytes
space: \begin{array}{c}
d \\
h \\
a \\
\end{array}

D. Nunez, S. Guyer, E. Berger
Prioritized GC: Measuring Sizes

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

| Marked by Mark Sweep | Marked by Prioritized GC |

Limit = 60 bytes
Current-total = 12 bytes
Space: d h a

Diagram:
- d
- h
- a
- b
- e
- f
- g
- c
- i
- map

D. Nunez, S. Guyer, E. Berger
Prioritized GC: Measuring Sizes

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

Marked by Mark Sweep

Marked by Prioritized GC

Limit = 60 bytes
Current-total = 24 bytes
Space: d h a

Diagram:
- Marked nodes: a, d, h, f, e
- Unmarked nodes: b, c, g, i

D. Nunez, S. Guyer, E. Berger
November 2, 2016 27 / 47
Prioritized GC: Measuring Sizes

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

Marked by Mark Sweep
Marked by Prioritized GC

limit = 60 bytes
current-total = 36 bytes
space: \[ \text{d h a} \]

D. Nunez, S. Guyer, E. Berger
Prioritized GC
November 2, 2016 27 / 47
Prioritized GC: Measuring Sizes

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

limit = 60 bytes
current-total = 48 bytes
space: [d h a]

Marked by Mark Sweep
Marked by Prioritized GC
Prioritized GC: Measuring Sizes

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

Limit = 60 bytes
Current-total = 60 bytes
Space: d h a

Marked by Mark Sweep
Marked by Prioritized GC
Prioritized GC: Eviction

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

Marked by Mark Sweep
Marked by Prioritized GC

limit = 60 bytes
current-total = 72 bytes
space: d h a

D. Nunez, S. Guyer, E. Berger

November 2, 2016 28 / 47
Prioritized GC: Eviction

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

Marked by Mark Sweep

Marked by Prioritized GC

limit = 60 bytes
current-total = 60 bytes
space: [d h a]

D. Nunez, S. Guyer, E. Berger
Prioritized GC: Eviction

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

Marked by Mark Sweep
Marked by Prioritized GC

limit = 60 bytes
current-total = 60 bytes
space: | d | h | c |

D. Nunez, S. Guyer, E. Berger

November 2, 2016
Prioritized GC: Sweep

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

Marked by Mark Sweep
Marked by Prioritized GC

limit = 60 bytes
current-total = 60 bytes
space: [d h]
Sache evicts upon next access

- Mark entries in space
- Mark roots
- Depth-first search from roots
- Depth-first search from space entries
- Free all unmarked objects

Marked by Mark Sweep
Marked by Prioritized GC

limit = 60 bytes
current-total = 60 bytes

space: d h

D. Nunez, S. Guyer, E. Berger
November 2, 2016 31 / 47
Experiments: Traces Used

- Traces are list of (string, int) pairs
  - string: unique string identifier
  - int: size of structure to create
    - drawn from a predetermined range
- Request sequence follows Pareto distribution\(^1\)

---

Experiments

- **Trace driven**
  - Program reads requests: (string, int) pairs
  - Program looks the string up in the cache
  - If the read fails, create and store a tree of the requested size in the cache

- **Change the cache used**
  - Google Guava with LRU policy at fixed sizes
  - Sache at fixed sizes

- **Implemented Prioritized GC in Jikes Research Virtual Machine**
Results: Workloads (50KB – 100KB)
Results: Workloads (100KB – 200KB)
Our Solution: Sache (Space Aware Cache)

- User puts a limit on size of Sache in bytes
- Tell the garbage collector what values are in the cache
- Measure the size of these values during garbage collection
- If keeping a value means the Sache exceeds its size limit, evict that value
Our Solution: Adaptive Sache (Space Aware Cache)

- User puts a limit on size of Sache in bytes
- Tell the garbage collector what values are in the cache
- Each collection, calculate limit on size of Sache in bytes
- Measure the size of these values during garbage collection
- If keeping a value means the Sache exceeds its size limit, evict that value
Adaptive Sache: Calculating Size Limits

Live data
Adaptive Sache: Calculating Size Limits

<table>
<thead>
<tr>
<th>Live data</th>
<th>Reserved for computation</th>
<th></th>
</tr>
</thead>
</table>

D. Nunez, S. Guyer, E. Berger
Adaptive Sache: Calculating Size Limits

Live data  Reserved for computation  Sache use
Experiments: Memory Pressure

- **Trace driven**
  - Use the workload with sizes in 50KB – 100KB range
  - After a third of requests read, increase memory pressure
  - After another third, decrease memory pressure

- **Change the cache used**
  - Google Guava with LRU policy fixed at 350 entries
  - Adaptive Sache with reserve of 50%
Results: Pressured Workload Trace: Total Run Time

![Graph showing the relationship between Time (ms) and Memory pressure (MB)](image)

Guava LRU
Sache

D. Nunez, S. Guyer, E. Berger
Prioritized GC
November 2, 2016

41 / 47
Results: Pressured Workload Trace: Total GC Time

[Graph showing the relationship between Memory pressure (MB) and GC Time (ms) for Guava LRU and Sache.]
Results: Adaptive Sache on Pressured Workload Trace

Add lines to note when memory pressure starts and ends

Number of Entries in Sache After Each Collection

Number of entries vs. GC Number
Conclusion

- Prioritized Garbage Collector allows programmers to handle references in application-specific ways
- Sache uses the Prioritized Garbage Collector to enforce memory limits
- Sache can adapt its size to memory pressure, improving robustness
Questions?
Backups
Saches and SoftReference Caches: Sache

![Graph showing the ratio of cache hit rates vs heap size for different frequencies (Freq = 1 to 10). The x-axis represents heap size in MB, ranging from 0 to 100, and the y-axis represents the ratio of cache hit rates, ranging from 0 to 100. The graph demonstrates a steady trend with some variations at different heap sizes and frequencies.]