Memory Bloat in the Real World

Harry Xu

UCI ISR Open Forum

05/18/2012

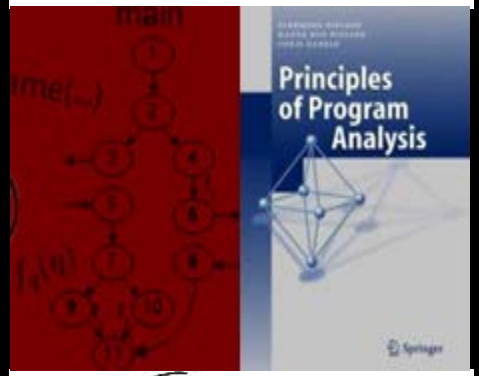
Who Am I


The Ohio State University

Sarah Marie Porter-Allen

Bachelor of Arts

In Testimony Whereof, the seal of the University and the signatures as authorized by the Board of Trustees are hereunto affixed.



- Recently got my Ph.D. (in 08/11)
- Interested in (static and dynamic) program analysis
  - Theoretical foundations
  - Applications
- Recent interest--- *software bloat analysis*

http://www.ics.uci.edu/~guoqingx

Is Today's Software Fast Enough?

- Pervasive use of large-scale, enterprise-level applications
  - Layers of libraries and frameworks
  - Object-orientation encourages excess
- No free lunch anymore from hardware advances
  - The size of software grows faster than the hardware capabilities (a.k.a. *Myhrvold's Law*)

Memory Bloat

Heaps are getting bigger

- Grown from 500M to 2-3G or more in the past few years
- But not necessarily supporting more users or functions

Surprisingly common (all are from real apps):

- Supporting thousands of users (millions are expected)
- Saving 500K session state per user (2K is expected)
- Requiring 2M for a text index per simple document
- Creating 100K temporary objects per web hit

Consequences for scalability, power usage, and performance

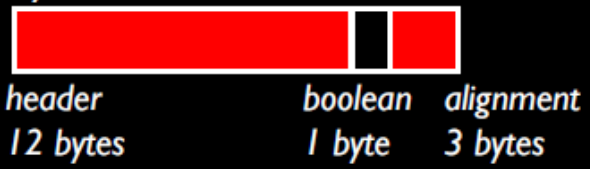
Outline

- Anecdotes
  - Costs of objects
  - Costs of fine-grained modeling
- Goals
  - Raise awareness of memory bloat
  - Give you a way to make informed tradeoffs

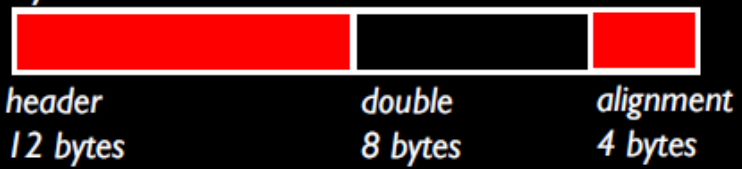
Anecdote 1: Costs of Objects

Q: are objects really cheap in memory?

Boolean
16 bytes



Double
24 bytes



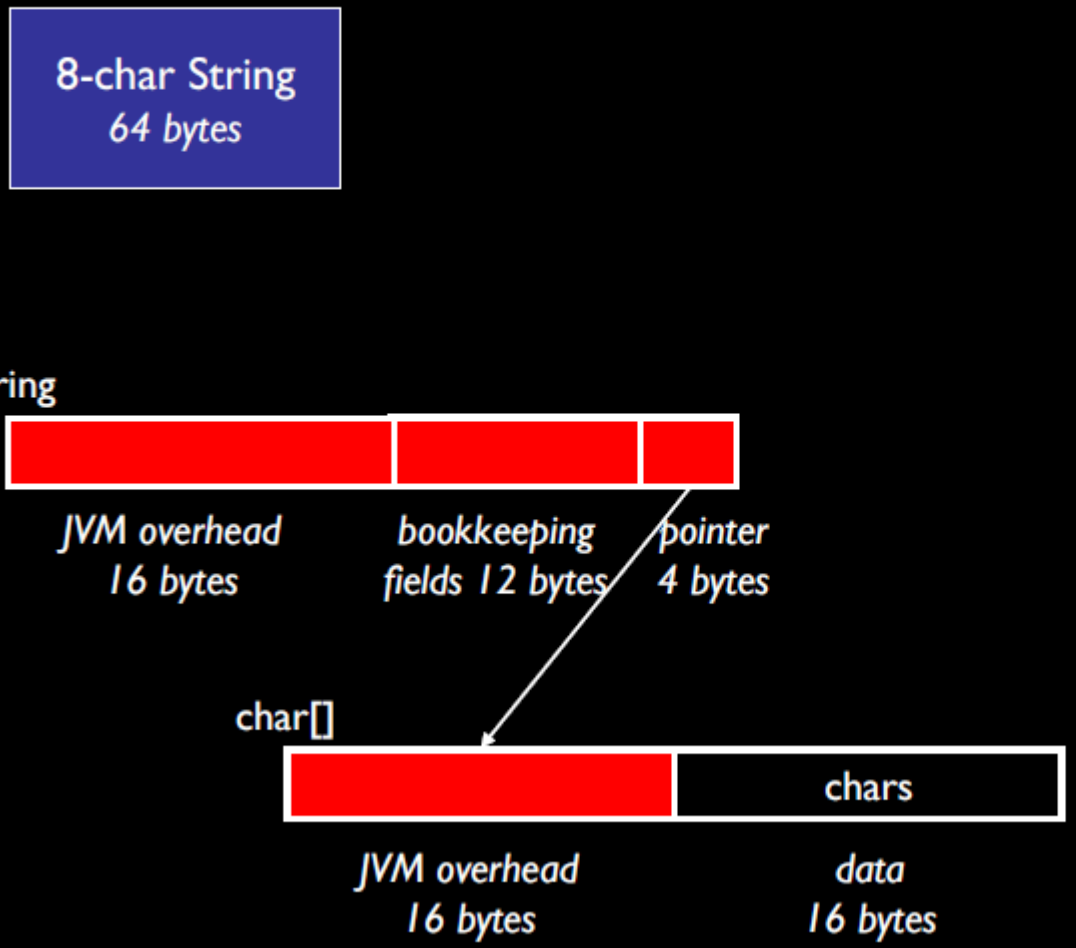
char[2]
24 bytes

From experiment on one 32-bit JVM

- JVM & hardware impose costs on objects. Can be substantial for small objects
- Headers enable functionality and performance optimizations
- 8-byte alignment in this JVM
- Costs vary with JVM, architecture

Another Example

Example: An 8-character String



- only 25% is the actual data
- 75% is overhead of representation
- would need 96 characters for overhead to be 20% or less

Consequences of Excessive Object Creation

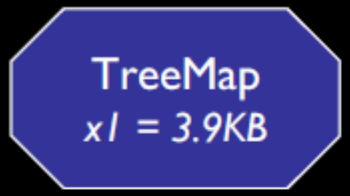
- Case study: Hyracks, a parallel data processing system written in Java
  - Extremely poor packing factor
  - Cannot process 1GB input data on a 12 GB heap if data elements are represented using objects
- Solutions
  - Release/remove objects soon after they are used
  - Reusing objects
  - Using memory in buffers (e.g., java.nio.ByteBuffer)

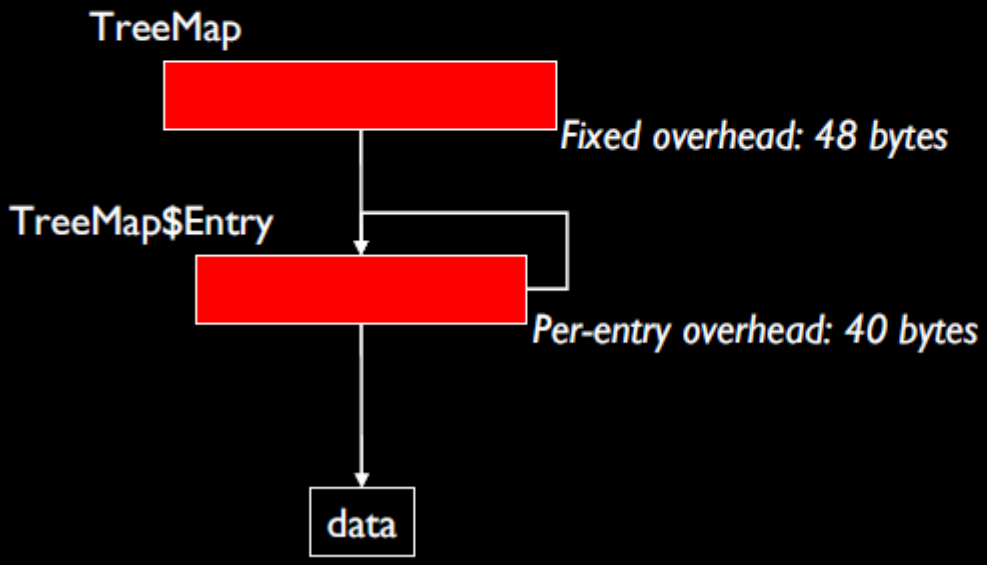
Anecdote 2: Costs of Fine-Grained Modeling

- Q: What's the cost of using a java.util.TreeMap

TreeMap

A 100-entry TreeMap

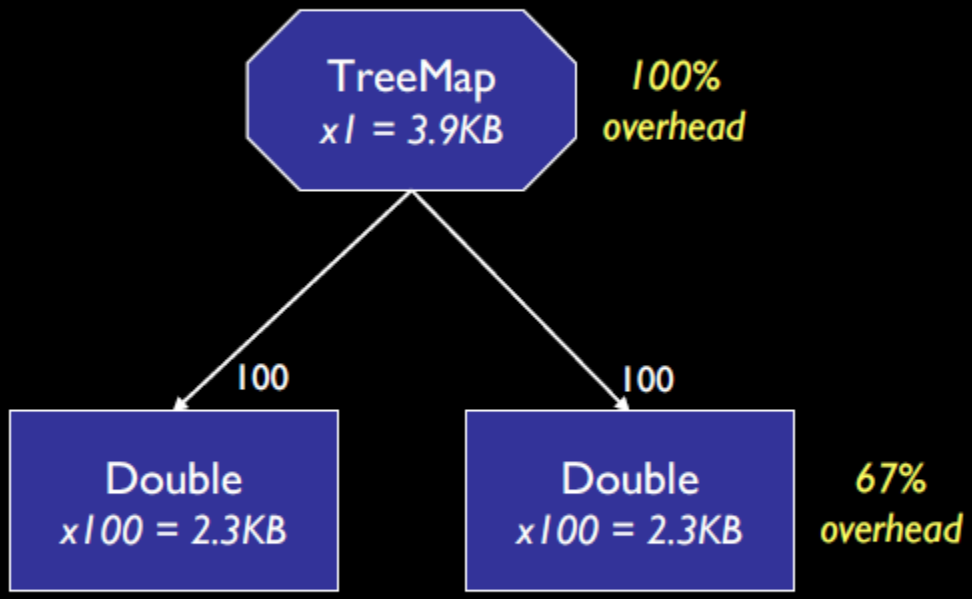




- How does a TreeMap spend its bytes?
- Collections have fixed and variable costs

TreeMap

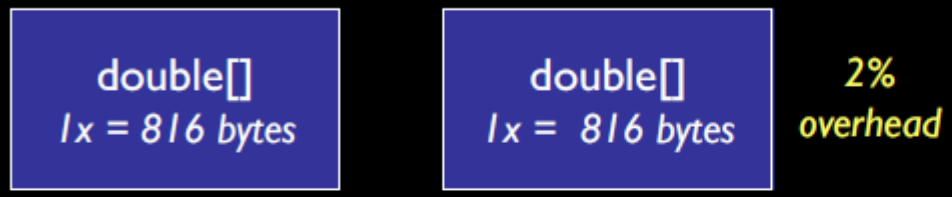
TreeMap<Double, Double> (100 entries)



- 82% overhead overall
- Design enables updates while maintaining order
- Is it worth the price?

TreeMap

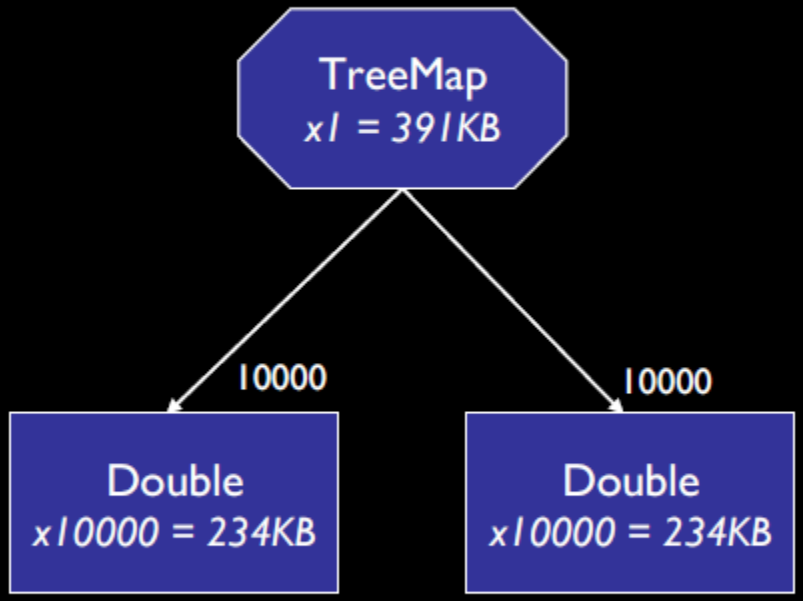
Alternative implementation (100 entries)



- Binary search against sorted array
- Less functionality – suitable for load-then-use scenario
- 2% overhead

TreeMap

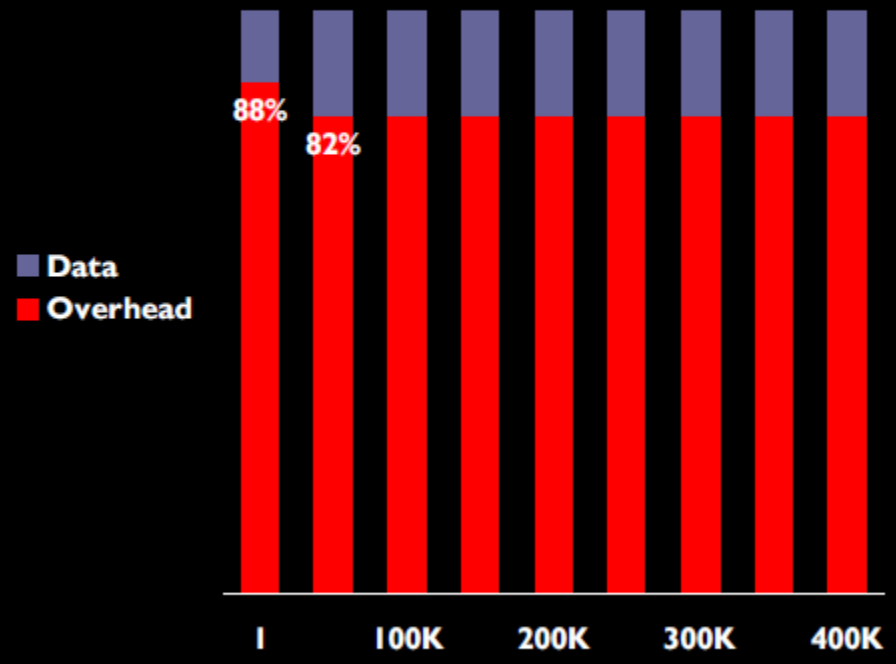
TreeMap<Double, Double> (10,000 entries)



- Overhead is still 82% of cost
- Overhead is not amortized in this design
- High constant cost per element: 88 bytes

TreeMap

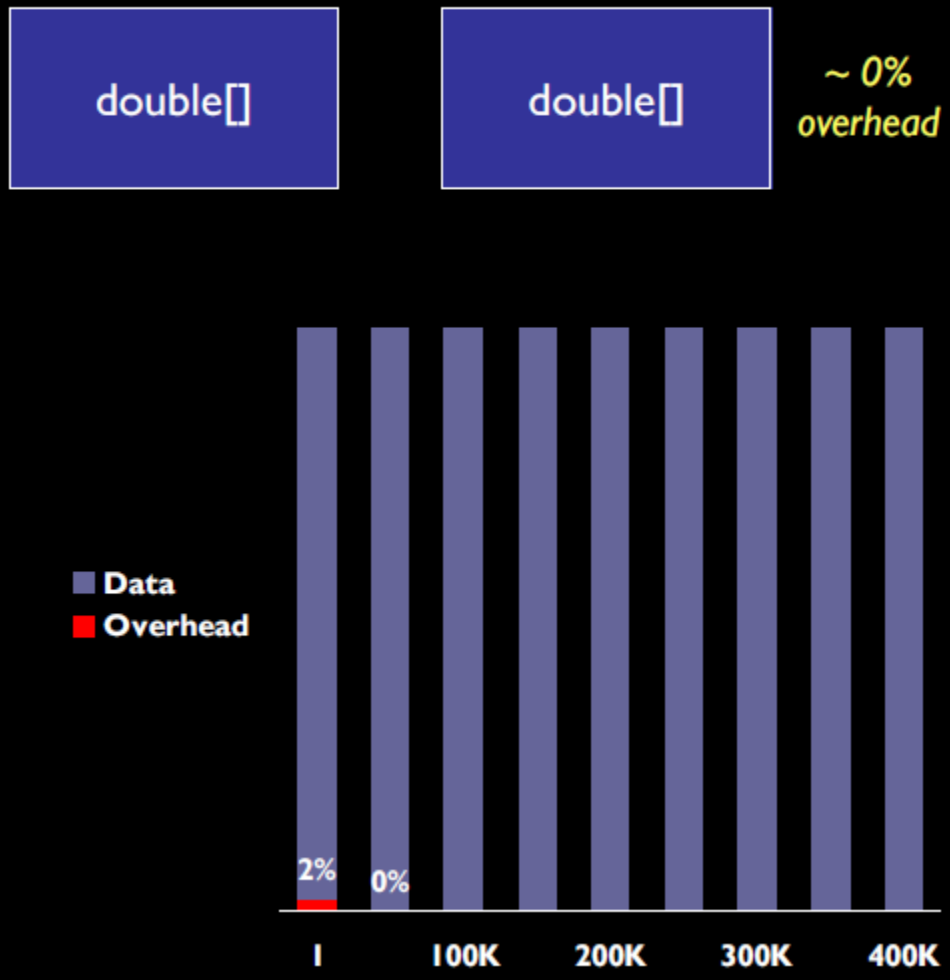
TreeMap<Double, Double>



- Overhead is still 82% of cost
- Overhead is not amortized in this design
- High constant cost per element: 88 bytes

TreeMap

Alternative implementation



- Overhead starts out low, quickly goes to 0
- Cost per element is 16 bytes, pure data

Consequences of Too Many Delegations

- Garbage collection is not free
  - Cost of a typical GC algorithm is O (|V| + |E|)
- Hyracks
  - *SELECT a, COUNT(*) AS FROM b GROUP BY c;*
  - Using a Java Hashtable for grouping leads to significantly increased GC time (47% of the total running time)
- Solutions
  - Arrays
  - Buffers
  - Customized data structures with less delegations

Conclusions

- A lot of things in object-orientation are not as cheap as we think
- Develop more specialized data types and operations
- My research targets these problems by developing language, compiler, and runtime system support

Acknowledgements

- IBM T. J. Watson Research Center
  - Nick Mitchell
  - Gary Sevitsky
  - Matthew Arnold
- UCI
  - Yingyi Bu
  - Vinayak Borkar
  - Michael Carey
- Ohio State University
  - Nasko Rountev
  - Tony Yan
- Office: 3212

Thank You