

“Creating and Preserving Locality of Java Applications at Allocation and Garbage Collection Times”

Authors:

Yefim Shuf, Manish Gupta, Hubertus Franke,
Andrew Appel, Jaswinder Pal Singh

Conference:

OOPSLA 2002

Presentation: Matthias Gauger

CS 8803G: Object-Oriented Systems and
Languages

Georgia Institute of Technology

Motivation

- Growing gap between processor and memory speeds
- Need for optimization strategies that improve data locality and thereby improve memory behavior
- Major challenge: techniques suitable for pointer-intensive applications
 - Java: pointer-intensive applications with dynamic memory allocation

General idea

- 1st idea: Co-locating objects together at allocation time
- 2nd idea: Memory-friendly traversal algorithm during garbage collection
- two natural points during program execution when the placement of data can be improved

Creating Locality at Allocation Time (1)

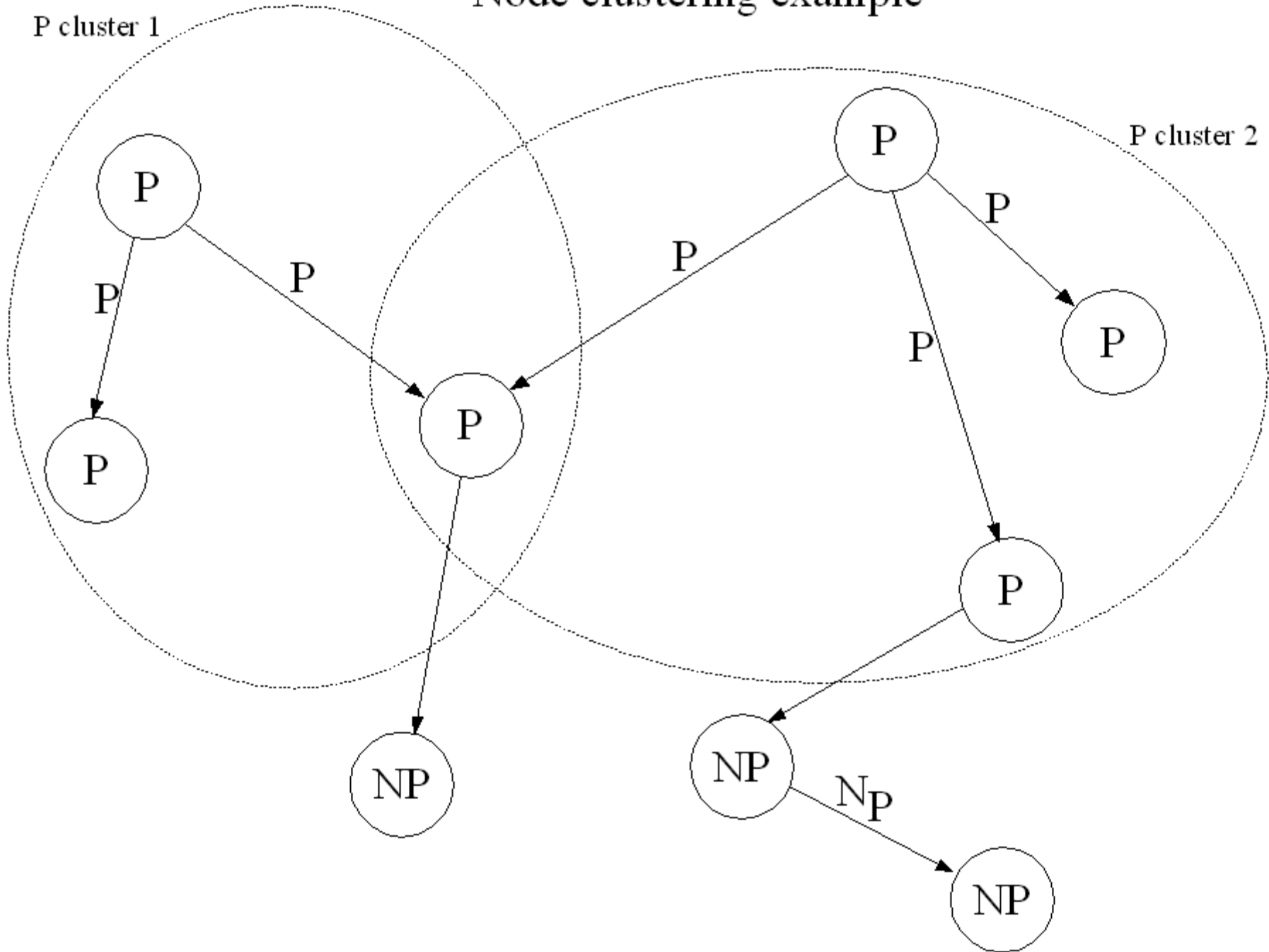
Prolific types

- For most programs a small number of types account for a large fraction of objects allocated at runtime – prolific types
- Properties:
 - short-living
 - small sized
 - children tend to be prolific, too
 - often leafs of the inheritance hierarchy

Creating Locality at Allocation Time (2)

- Co-locate objects of prolific types
- Node clustering algorithm
 - Nodes = types; directed edges = references
 - labeled with P (prolific) or NP (non-prolific)
 - cluster together nodes chained by P-edges
- Representative type in each cluster
- Create enough space for all objects in the cluster when an object of the representative type is created.

Node clustering example



- Advantages of this algorithm:
 - It improves the spatial locality of applications
 - It reduces GC time
 - It often reduces memory fragmentation

- Some interesting special cases:
 - Child object created before parent object.
 - Size of a child object unknown
 - Type of child unknown

Preserving Locality at GC Time (1)

- Divide memory into equal sized chunks
- Visit objects of only one chunk at a time.
- Differentiate between local pointers (LP) and non-local pointers (NLP).
- Visit the LP-locations first.
- When no LP-pointers are left visit next chunk.

Preserving Locality at GC Time (2)

- ▶ exploits the inherent locality of data structures
- ▶ different heuristics to choose the next chunk
- ▶ aims to achieve shorter GC delays

Discussion (1)

- Non-copying GC
 - Object co-allocation useful, especially given poor locality of the base configurations
 - Locality-based traversal only in combination with object co-allocation reasonable
- Copying GC
 - Object co-allocation useless – already good locality of new objects.
 - Locality-based traversal useful – preserves the given good locality.

Discussion (2)

- Relatively simple enhancements – no deep program analysis necessary.
- Performance evaluated with different Java-benchmarks (up to 20% performance improvements) on the Jikes RVM
- Object co-location:
 - Some weak points, but „works well in practice“
 - Improvement depends on the given locality
 - Strength: No correctness problems

References

- [1] Y. Shuf, M. Gupta, H. Franke, A. Appel, and J. Singh. Creating and Preserving Locality of Java Applications at Allocation and Garbage Collection Times. In Proc. of the 17th ACM conference on Object-oriented programming, systems, languages, and applications (OOPSLA 2002), pages 13-25, Seattle, Washington, November 2002
- [2] Y. Shuf, M. Gupta, R. Bordawekar, and J.P. Singh. Exploiting prolific types for memory management and optimizations. In Proc. of the 29th SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2002), pages 295-306, Portland, Oregon, January 2002