Area Notes

Ken Dickey

PREFACE

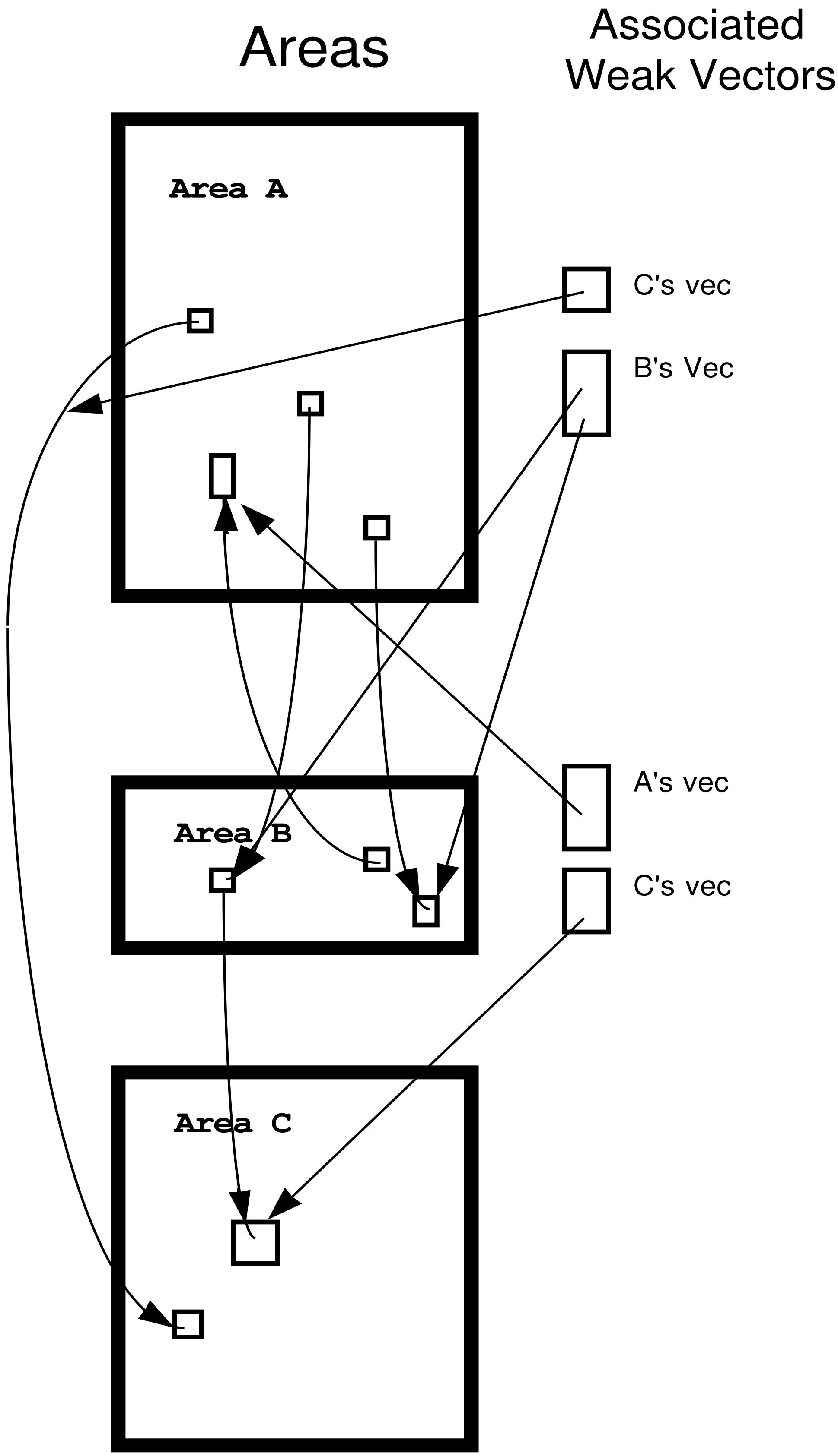
This paper presents a simple, general model for sharing objects between memory areas which may have separate storage managers. Only local synchronization is required. This model accords well with generational [LH83] and distributed (e.g. [Sha91]) garbage collectors, but also allows for interaction with manual storage managers, provided they properly manage their references.

The primary motivation for this 'Note' is the desire to see more cooperation between multiple language runtime systems in a shared memory environment (note [Dobbs]). Another usage is the shortening of GC pause times by using a number of smaller, separately gc'able areas which preserve some locality while otherwise sharing resources. The reader is assumed to have some background in automated storage reclamation technology (e.g. see [Wil92]).

Note that this is a discussion of a model protocol and that many variations and optimizations can be applied. The intent here is for clarity of understanding.

MODEL DESCRIPTION

Let's first describe a simple view of the model as might be seen in an implementation of a single language runtime which supplies multiple areas (a.k.a. heaps, spaces, regions). For each area there is a set of vectors of weak pointers. Each remote area has its own vector so that when a pointer to a remote area is assigned to the local area, the reference is also added to that remote area's local vector. Thus each of these vectors acts as a registry of objects which live in another area but are referred to by the local area. When the local area collects, its non-local vectors are part of its root set. After the main collection phase, it scans the local vectors for the other areas and sets the vector slots of references to remote objects which it no longer references to some non-pointer value (e.g. zero).



B uses 1 object from A and C; A uses 2 from B, 1 from C

What conditions minimize the need for synchronization of area collections? There are three points of interest: [1] managing the weak vector registries, [2] data migration, and [3] data access and update.

As a write barrier (e.g. [Hol93]) can be used to add weak references and as the local area removes them, there is little need for direct synchronization of reference management. The major constraint is that references to remote areas not be added while that remote area is in an inappropriate collection phase (e.g. reading the root set), which is handleable in the write barrier when an inter-area assignment is detected. For protocol purposes the assignment code is logically in the remote area, as it is the exporter of the reference. Each collector supplies its own export registration function so that it can maintain its local collection invariants when a local object is exported. Registering a newly participating area can be done lazily at the first inter-area assignment. Resizing vectors requires synchronization, but this can be be made a fairly rare event.

Data movement because of compaction can be avoided by disallowing it or by requiring access through a function which performs the appropriate synchronization. Note that today many commercial implementations of dynamic languages provide access to native data structures by static (e.g. C, C++) foreign runtimes by migrating native data objects to a static area (if required) and using a registry so that the foreign code can indicate when it is no longer referencing that object. The native object may then be directly deallocated, migrated back to a dynamic area, or collected in place (e.g. via some Variation on mark-sweep [McC60], [Coh81], treadmill [Bak92], [Hen93], or other suitable reclamation technology {e.g. [WJ93]}). If a non-copying collector is used, then it does not matter if an area is exporting its references to a static or a dynamic language as long as the area protocol is satisfied.

Concurrency protection for data access and mutation is highly system specific and cannot be covered here. E.g. see wait free and lock free methods [Mas92], [Her93], [Suk94]. It is assumed that exported data objects are manipulated through setter and getter functions which hide the access protocols.

Now, what is required to generalize this model to multiple, heterogeneous runtime systems? The basic constraint is that the representation of references between the systems either be the same, a mapping table be used, the references treated as opaque, or the same access protocol be used (e.g. mask the low 3 bits and use the resulting integer as a machine address). There must be a way to import/export objects, but this can use the area specific export registration function mentioned above.

[Again, remember that the above is a model. There are a wide variety of optimizations and synchronization primitives available to apply in various specific situations].

COSTS and BENEFITS

The basic costs are:

[A] Enlarged root sets.
[B] Local vectors of weak pointers.
[C] Concurrency protection for thread-safe data access and mutation.
[D] Static areas/non-copying collection.
[E] Write barrier for some writes (note [Hol93]); other object export protocol.
[F] Writing and testing the collector-specific export registration code.

The benefits are the ability to locally collect areas within a heterogeneous runtime environment with very low synchronization overhead and accommodating a variety of collection strategies. Collection implementation knowledge remains local to the areas being collected. Likewise, synchronization is local. The local area's collector does not care if exported objects are used by a static or dynamic system. The design choices taken here allow light weight mechanisms to be used for systems which fit the situation described.

NOTES

There are a number of programmatic situations where areas allow for local collection within a single runtime system. An area might be allocated for each spawned thread. Programmers may be given leave to specify areas where they expect good locality.

Exported objects containing only intrinsic/immediate data managed through exported setter and getter functions but not requiring object identity can be simply cloned and therefor do not need reference management.

Copying collectors can be managed by a lockout protocol where no references are allowed into the area while collecting and references have been registered in remote areas so that they can be updated at the end of the collection cycle. This might not be too onerous in a single-cpu shared memory environment which uses indirection tables with guarded access. Copying collectors could use less memory by sharing fewer spare "toSpace" regions where area locality is good.

Changes required for generalization to distributed environments is covered in [Sha91].

More general heterogeneous gc techniques which require cooperation between collectors based on shared constraint agreement is covered in [Del82].

REFERENCES

[Bak92] Henry Baker: "The Treadmill: Real-time Garbage Collection without Motion Sickness", ACM SIGPLAN Notices, 27(3), MACH 1992.

[Coh81] Jacques Cohen: "Garbage Collection of Linked Data Structures", Computing Surveys, 13(3), September 1981.

[Del82] V. Delacour: "Allocation Regions & Implementation Contracts", Proc. 1992 SIGPLAN International Workshop on Memory Management, Springer-Verlag LCNS #637, September 1992.

[Dobbs] Dr. Dobb's Special Report on Interoperable Objects {SOM, COM, CORBA, OLE, OpenDoc, AppWare, Taligent}, Miller Freeman #255, Winter 1994/95.

[Hen93] Wade Hennessey, "Real-Time Garbage Collection in a Multimedia Programming Language", 1993 OOPSLA Workshop on Memory Management and Garbage Collection, September 1993.

[Her93] M. Herlihy: "A Methodology for Implementing Highly Concurrent Data Objects", ACM Trans. Prog. Lang. and Systems (TOPLAS), 15(5), November 1993.

[Hol93] Urs Holzle: "A Fast Write Barrier for Generational Garbage Collectors", 1993 OOPSLA Workshop on Memory Management and Garbage Collection, September 1993

[LH83] Lieberman & Hewitt: "A Real-Time Garbage Collector based on the Lifetimes of Objects", Communications of the ACM, 26(6), June 1983.

[Mas92] Henry Massalin: "Synthesis: An Efficient Implementation of Fundamental Operating System Services", PhD Thesis, Columbia U., 1992.

[McC60] John McCarthy: "Recursive Functions of Symbolic Expressions and their Computation by Machine", Communications of the ACM, 6(9), April 1960.

[Sha91] Marc Shapiro: "A Fault-Tolerant, Scalable, Low-Overhead Distributed Garbage Detection Protocol", Tenth Symposium on Reliable Distributed Sytems, Pisa, Italy, September 1991, IEEE Computer Society Press.

[Suk94] Joseph Sukdlarek: "Remarks on A Methodology for Implementing Highly Concurrent Data Objects", ACM SIGPLAN Notices, 29(12), December 1994.

[Wil92] Paul Wilson: "Uniprocessor Garbage Collection Techniques", Proc. 1992 SIGPLAN International Workshop on Memory Management, Springer-Verlag LCNS #637, September 1992.

[WJ93] Wilson & Johnstone: "Real-Time Non-Copying Garbage Collection", 1993 OOPSLA Workshop on Memory Management and Garbage Collection, September 1993.

--- E O F ---

FILE	"AreaNotes.wp"
IMPLEMENTS	Notes on area based object sharing & gc.
AUTHOR	Ken Dickey
DATE	1995 January 21-2
LAST UPDATED	1995 January 26 -- added formatting & figure
STATUS	Draft; comments appreciated.

- 7 -