Collecting Cyclic Distributed Garbage

Umesh Maheshwari (umesh@lcs.mit.edu)

M.I.T. Laboratory for Computer Science
Cambridge, MA 02139

1 Introduction

Systems that store objects on networked computers (nodes) reclaim unusable objects by either global marking [HK82] or distributed reference counting [Bis77]. Global marking requires the cooperation of all nodes before it can collect any garbage. Distributed reference counting is preferred for systems with large number of nodes because it is more fault-tolerant and scalable. However, it cannot collect multi-node cycles of garbage objects. In long-lived systems such as persistent object stores, even small amounts of uncollected garbage can cause significant storage loss over time.

The problem can be solved either by using a complementary marking scheme, or by migrating objects so that cyclic garbage ends up in a single node and is collected by the local collector [Bis77, SGP90, GF93]. Migration-based schemes, like distributed reference counting, are decentralized and fault-tolerant: The collection of a cycle requires the cooperation of only those nodes that contain it, and progress is made even if other nodes or other parts of the network fail.

Existing migration schemes have some practical problems. Most schemes migrate all locally unreachable objects, although they may be reachable from roots on other nodes [Bis77, GF93]. Migration of live objects is undesirable because it wastes processor and network bandwidth. Worse, it interferes with object placement and load balancing.

This paper presents simple techniques to avoid unnecessary migration. We use a heuristic to detect objects that are highly likely to be cyclic garbage and migrate only those. Further, we avoid migrating objects multiple times by estimating a destination node where a garbage cycle can converge.

2 The Environment

Our algorithm is designed for use in the Thor object database, and it is applicable to other systems that store objects at multiple nodes. Objects contain references to other objects, which may be on other nodes. An object is reachable from another if there is a path of references from the second to the first. Liveness of objects is determined by reachability from the persistent root objects. If two garbage objects on different nodes are reachable from each other, they are said to be on a multi-node garbage cycle.

In distributed reference counting schemes, each node does an independent local collection based on tracing from a root set that includes the local persistent root objects as well as local objects that are referenced from other nodes. We use a variant called reference listing [Bis77, ML94]. Here, a node N_1 keeps, for every other node N_2, an *inlist* of objects in N_1 that N_2 may hold references to. When a new internode reference is created from node N_2 to an object x at node N_1, N_1 enters a reference to x in its inlist for N_2. The local collector at N_1 uses inlist entries as roots. As it traces, it records all references to remote objects, say on N_3, in an *outlist* for N_3. At the end of collection, N_1 sends the outlist to N_3, which then uses the outlist to replace its inlist for N_1.

3 Distance Heuristic

We detect cyclic garbage based on the *distances* of objects. The distance of an object is the minimum number of internode references in any path from a persistent root to that object. The distance of an object unreachable from the persistent roots is infinity. Figure 1 illustrates the notion of distance. Object r is a persistent root and therefore has zero distance.

Estimates of object distances are computed as follows. A distance is associated with each reference in the root set: The distance of a persistent root is implicitly zero, and each reference in an inlist has a distance field. When a new inlist entry is created, its distance is simply set to one. The estimated distance of an object is implicitly the minimum distance of any reference in the root set it is reachable from.

The local collector propagates distances from roots to outlists, setting the distance of an outlist entry to one plus the minimum distance of any root it is reachable from. The

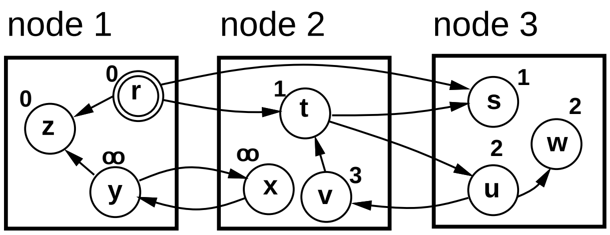

Figure 1: Distances of objects.

This research was guided by Prof. Barbara Liskov. It was supported in part by the Advanced Research Projects Agency of the Department of Defense, monitored by the Office of Naval Research under contract N00014-91-J-4136.

collector accomplishes this by tracing from the roots in the increasing order of their distances and tracing completely from one root before going on to the next. (This is similar to timestamp propagation in [Hug85].) As before, a node uses an outlist received from another node to replace its inlist for that node.

Propagation through local collections and outlist messages causes the distances of cyclic garbage to increase without bound. It can be shown that if the nodes containing a garbage cycle do local collections and exchange outlists periodically, the distance of the garbage objects will increase at the rate of one per round of collection, on the average.

Since the distance estimates of live or non-cyclic garbage objects do not increase indefinitely, it is possible to select a *threshold* distance, T, such that all objects with a greater distance are highly likely to be cyclic garbage. Only these objects are migrated.

The choice of the threshold depends on the expected distances of live objects. Since estimated distances of live objects may deviate temporarily from their actual distances, the threshold should be chosen to be a small multiple of the expected maximum distance. Fortunately, the penalty on misjudging the threshold is not severe: If it is low, some live objects may be migrated but safety is not compromised since the objects would not be deleted. If the threshold is high, all cyclic garbage will still be detected eventually.

Distance propagation needs the cooperation of only the nodes that the garbage lies on. Specifically, if a node is crashed, partitioned, or otherwise slow in doing local collection, the collection of only the garbage that is reachable from its objects will suffer.

Further, the scheme has little overhead. Distance fields are only associated with inlist and outlist entries, not with all objects. No extra messages are required over those already sent for collecting non-cyclic distributed garbage.

4 Migration

Consolidating a distributed cycle by migrating objects presents some practical problems. First, migrating a remotely referenced object that contains references to local objects is likely to create more remotely referenced objects that may have to be migrated later. Our scheme makes it simple to batch objects: Once the distance of the root being traced is above the threshold, all objects traced from the same root reference are migrated together.

Second, most existing schemes migrate objects to nodes that reference them. To ensure that all objects in a cycle converge on the same node instead of following each other in circles, nodes are totally ordered by ids and migration is allowed only in, say, the increasing order of node ids [SGP90]. However, objects on a multi-node cycle may still require multiple migrations before converging on the maximum node the cycle passes through.

To avoid multiple migrations, we propagate estimates of the maximum node. Inlist and outlist entries with distances greater than the threshold have an associated destination field. When the collector creates such an outlist entry, the destination is set to the higher of the target node id, the local node id, and the destination field (if any) of the inlist entry begin traced. To propagate the maximum node id, all inlist entries above the distance threshold are traced in the decreasing order of their destination fields. Before migrating objects, nodes wait until the destination information is likely to have propagated around the cycle. Even if some objects are migrated before all nodes agree on the destination node, a cycle would still eventually converge on the same node.

Any migration-based scheme has some limitations. If there are too many objects in a garbage cycle, there may be no space for them in the destination node. Our scheme helps by avoiding unnecessary migration, but further improvements are desirable. For example, our scheme migrates objects on garbage chains going out from garbage cycles, although such chains could be better collected by distributed reference counting after the garbage cycle is collected.

5 Conclusions

We have presented a simple and efficient way of using object migration to collect cyclic distributed garbage. Our approach is to limit migration to the bare minimum. We use the distance heuristic to avoid migrating objects that are not cyclic garbage. Further, we migrate objects directly to the selected destination node to avoid multiple migrations. Our scheme is decentralized and adds little overhead to the system. More about our scheme is found in [ML95].

References

[Bis77] P. B. Bishop. Computer Systems with a Very Large Address Space, and Garbage Collection. *Technical Report MIT/LCS/TR-178*, MIT Lab for Computer Science, Cambridge MA, May 1977.

[GF93] A. Gupta and W. K. Fuchs. Garbage Collection in a Distributed Object-Oriented System. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 5, No. 2, April 1993.

[HK82] P. Hudak, and R. Keller. Garbage Collection and Task Deletion in Distributed Applicative Processing Systems. *ACM Symposium on Lisp and Functional Programming*, pages 168–178, August 1982.

[Hug85] J. Hughes. A Distributed Garbage Collection Algorithm. *Functional Programming and Computer Architecture (Lecture Notes in Computer Science 201)*, pages 256–272, Springer-Verlag, September 1985.

[ML94] U. Maheshwari, and B. Liskov. Fault-Tolerant Distributed Garbage Collection in a Client-Server Object-Oriented Database. *Proceedings of the Third International Conference on Parallel and Distributed Information Systems*, pages 239-248, September 1994.

[ML95] U. Maheshwari, and B. Liskov. Collecting Cyclic Distributed Garbage By Controlled Migration. To appear in *Proceedings of Principles of Distributed Computing*, August 1995.

[SGP90] M. Shapiro, O. Gruber, and D. Plainfosse. A Garbage Detection Protocol for a Realistic Distributed Object-Support System. *Research Report 1320*, INRIA–Rocquencourt, November 1990.