Fun Ideas and Thoughts
PLDI 2009
Decentralized Garbage Collection

Eric Hennigan
eric.hennigan@uci.edu
University of California, Irvine

May 15, 2009

1 Motivation

What techniques are useful for scaling the heap size of garbage collected languages to hundreds of Gigabytes? Are the same techniques appropriate for distributed systems (the most likely architecture for for hundred Gigabyte heaps)? We propose a method of garbage collection that uses inter-object references to compute a graph centrality measure that determines when each object is safe for collection.

2 System Design

The critical model of decentralized garbage collection is not to use the conventional approach of solving graph reachability using a global monitor, but to gift each node with the ability to identify *itself* as being disconnected from root nodes. In this approach we think of each object as being its own process, with the ability to discover, for itself, when it should be reclaimed. To accomplish this we use distributed calculation of graph centrality.

We consider two types of nodes which need garbage collection: (1) root nodes which appear on the call stack, and become collected automatically when stack frames are popped, and (2) heap nodes which are allocated in the garbage collected heap. Only those heap nodes which are reachable from a root node are considered necessary for program execution, all other heap nodes can be collected.

Each object in the system has two tags, *self worth* W_s and *computed worth* W_c, that keep track of its connectedness to a root node. Root nodes are initialized with both self worth and computed worth set to a large number (`MAXINT`), and do not update their computed-worth. Heap nodes are initialized with self worth set to `0` and computed worth set to a large number (`MAXINT`). After creation, an object participates in a centrality calculation that determines their reachability from a root node.

Objects will send messages to each other, continuously updating their computed worth using a centrality measure. When the computed worth of a node

dips below a threshold it will select itself as a candidate for collection. The computed worth of an object i is determined by a function of the computed worth of neighboring nodes and its own self worth. An example of such a function is:

$$W_c(i) = \frac{W_s(i) + \sum_{j \in neighbors(i)} W_c(j)}{1 + degree(i)} \quad (1)$$

Together with the initial conditions, the important properties of this measure are twofold:

1. Root nodes never wish to collect themselves, because their computed worth is set to a large number, and is never updated.

2. Heap nodes, which have a self-worth of `0`, maintain their computed worth solely as a result of their reachability from root nodes.

To demonstrate when objects select themselves as canditates for collection, consider a subgraph of heap nodes which becomes disconnected from the root. These nodes will continuously update their computed worth, but, because every node in the heap has a self worth of `0`, the computed worth will begin to decay. After some period of time, this value will dip below a threshold value, at which point it is safe for the object (and every node in its subgraph) to be collected. The decay rate, which determines the speed with which objects are collected, is dependant on both the connectivity of nodes within the disconnected subgraph, and the threshold value.

3 Value of the Idea

We recognize that distributed calculations in the real world result in much network overhead. We think that our system can minimize this by processing heap nodes using an event queue. We believe it is possible to construct the system in such a way that the distributed calculation can be performed in a separate garbage collection thread. The tags associated with each object can be placed in memory locations disjoint from the objects themselves, and the distributed calculation is robust with respect to spontaneous additions and removals of both edges and nodes in the graph.

The primary implementation difficulty is to ensure that at least one node from every disconnected subgraph is in the event queue, otherwise nodes of that subgraph will stop participating in the distributed calculation and never will dip below the collection threshold. The distinct advantage to our system is that it can act completely independently of all other calculations, there is no need to stop the world, or lock objects to perform a reachability analysis.