

# Fractional Weighted Reference Counting

Erik Klintskog<sup>1</sup>, Anna Neiderud<sup>1</sup>, Per Brand<sup>1</sup>, and Seif Haridi<sup>2</sup>

<sup>1</sup> Swedish Institute of Computer Science, Box 1263, SE-164 29 Kista, Sweden,  
<http://www.sics.se/>

<sup>2</sup> Royal Institute of Technology, Department of Teleinformatics, Electrum 204, S-164  
40 Kista, Sweden, <http://www.it.kth.se/>

**Abstract.** We introduce a scheme for distributed garbage collection that is an extension of Weighted Reference Counting. This scheme represents weights as fractions. It solves the problem of limited weight, preserves the property of third-party independence, and does not induce extra messages for reference merging.

## 1 Introduction

When deciding on how to manage memory in a distributed application it is important to study how references to different objects or entities are spread and how long they are expected to live. Many traditional applications use a rather static client-server setup, where the client will have a few references to entities at the server during a session and then disconnect. For more novel applications following a dynamic peer-to-peer design pattern, a programming platform needs to provide faster means to collect garbage since many references will only live for a short period of time. In such applications references will also tend to spread more wildly over the network, and it is important that a garbage collection algorithm does not impose any third party dependencies. Allowing multiple instances of references to the same remote entity from one process is undesirable in a distributed programming platform since it complicates distributed entity-consistency protocols, and may require large amounts of memory. This puts demands on the garbage collection algorithm to handle merging of references.

As stated in numerous papers [4], Weighted Reference Counting (WRC) (presented independently by Watson and Watson [6] and Bevan [1]) is a distributed GC algorithm that efficiently and correctly collects garbage without requiring global synchronization. A problem with WRC is the weight underflow. Proposals for solving this problem exist but have some shortcomings. Piquer [3] creates indirection cells for weight extensions. Goldberg's [2] solution may lead to large tables at the distributed objects process. The algorithms have problems with reference merging. In a termination detection algorithm Mattern [5] proposed using rational numbers for the weight rather than integers. When a reference is shared the weight is divided by two, which gives unlimited weight, but requires a representation of the growing rational number. With long-lived and mergeable references the size of the numbers that must be stored will be unbounded.

We will present a version of WRC called Fractional Weight that solves the problem of weight underflow and is able to merge multiple instances of references. This is done in bounded memory without imposing any new third party dependencies. We will also show how Fractional Weight can be parameterized to handle different sharing patterns.

## 2 Fractional Weighted Reference Counting

All distributed entities have a home process. At the home process there exists an entry (stored in a table) that points to the entity. This is called the owner reference. At other processes there may exist one and only one remote reference pointing at the owner reference. As long as there exists at least one remote reference, the owner reference must not be reclaimed. A distributed garbage collector is responsible of maintaining the consistency of remote references to an entity.

### 2.1 Representing Fractions

In Fractional Weight, the total weight is defined to be one. All partial weights are represented as a fraction with a denominator  $D$  or a power of  $D$ , where  $D$  is a static integer. A fractional number is represented as a linked list of pairs, where the first value is the enumerator (1) and the second is the power of the denominator. The current weight of a reference can then be written as the sum of all its pairs (2).

$$D \geq N_k \geq 0 \tag{1}$$

$$W = \sum_{k=1}^{\infty} \frac{N_k}{D^k} \tag{2}$$

To avoid unnecessary calculations and large messages when sending fractions between different processes, sharing weight is taken from one single pair. This results in unit fractions representable as  $1/D^k$ . Sharing weight is done by giving out *GiveSize* such unit fractions. When an enumerator to be shared has reached the undivisible number one, it can be extended by equation (3) resulting in the creation of a new pair with the denominator  $k+1$  and  $D$  smaller fractions.

$$\frac{1}{D^k} = \frac{D}{D^{k+1}} \tag{3}$$

When receiving a fraction pair with enumerator  $M$  and denominator power  $k$  it must be inserted into the linked list of pairs. If no pair exist with denominator power  $k$  a new pair is created with the received numbers. Otherwise, the enumerators of the received and the found pairs are added together. The property of (1) must still hold. In the case of overflow,  $N_k$  is set to  $M + N_k - D$  and a new insert operation is performed with  $1/D^{k-1}$ .

By using a large denominator base  $D$ , we will have a large number of weight fractions of each denominator (each  $D^k$ ) to share. How long these fractions will last before an extension must be performed, depends on how the reference is shared and on how many are shared each time.

This approach forces the Owner reference to be able to store an arbitrary number of pairs. A remote reference has the choice of how many pairs to store, and weight of other pairs can be returned. Returning weight minimizes memory usage but increases the network traffic to the owner reference. Therefore  $D$ , and *GiveSize* should be chosen carefully to minimize the traffic (see below).

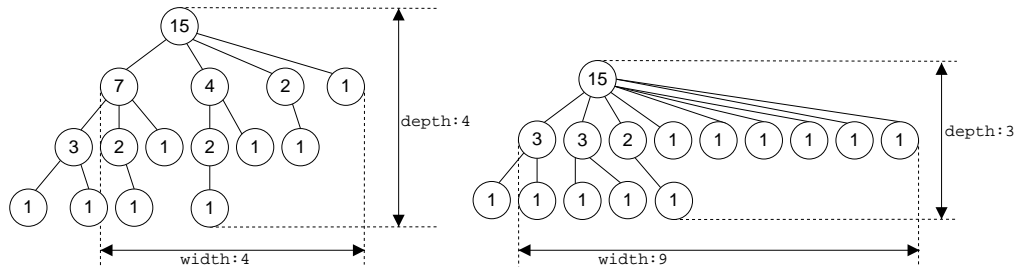
## 2.2 Sharing Fractions

Wide sharing without extensions, as needed in client server applications with many clients, can be achieved by giving out only single unit fractions, and thus make the weight last long. Deep sharing without extensions can be achieved by giving all but a single unit fraction and may be needed in large peer-to-peer applications. In cases where the sharing pattern is random or not known to the programming platform, we define a default algorithm below for finding a *GiveSize* optimized for a combined sharing pattern. Instead of using fixed numbers, we use a fix divisor  $\alpha$ , where  $\alpha > 0$ , to define how much of the available weight should be shared.

Recall that *GiveSize* is the number of unit fractions to give out. In order to have a system that strives towards larger fractions, we take unit fractions from the term of the sum (2) that has the smallest  $k$  where  $N_k > 0$ . If this  $N_k = 1$ , we start by extending it using equation (3). We then define *GiveSize* as:

$$GiveSize = \begin{cases} N_k \mathbf{div} \alpha, & N_k > \alpha \\ 1 & , N_k \leq \alpha \end{cases} \quad (4)$$

Consider the maximum possible sharing of references without performing any merging or extending the range that is depicted in Figure 1 as a tree. A low  $\alpha$  results in a skewed tree with a deep left leg. A slightly higher  $\alpha$  changes the characteristics of the tree, the tree becomes wider and more shallow. There are three interesting properties of sharing: max width, max depth and average depth. Table 1 shows some sharing patterns for a set of different  $\alpha$ . As can be seen, 'max depth' and 'max width' are inversely proportionally. A higher 'max depth' implies a lower 'max width'. The relation is not linear though, since  $\alpha = 2$  has a 6 times higher max depth than  $\alpha = 1000$ ,  $\alpha = 1000$  has a 526 times higher max width than  $\alpha = 2$ . The 'average depth' shows an even smaller difference between the smallest and the greatest  $\alpha$ . An application that needs a good depth as well as a good width can therefore use an  $\alpha$  in the range of 10 - 50 and obtain a max depth four times lower than  $\alpha = 2$ , but have a max width ten times greater than what  $\alpha = 2$  to gives. The loss in depth is even smaller when looking at the average depth, in this case the loss would only be a factor of three.



**Fig. 1.** The shape of the sharing tree using  $D = 15$  and  $\alpha$  2 and 4. The top circle represents the owner reference and the rest represent remote references. The numbers denote the initial weight as number of fractions ( $N_1$ ) at that node. Note that the number of nodes is always equal to  $D$ .

**Table 1.** Shape of the tree for some different  $\alpha$  with  $D = 2^{32} - 1$ . This value of  $D$  corresponds to using a 32-bit word for storing it.

$\alpha$	max depth	max width	average depth
2	32	32	18
3	21	55	13
4	17	77	11
8	12	161	9
10	11	203	8
50	7	981	6
100	6	1905	5
1000	5	16841	4

### 3 Conclusions

We have presented an extension to Weighted Reference Counting, called Fractional Weighted Reference Counting. The new algorithm solves the problem of limited weight by a fractional representation of weights. The algorithm preserves the property of third-party independence, and does not induce extra messages for reference merging.

We have also shown how the algorithm can be parameterized to adapt to different types of reference graphs in distributed applications including client-server and peer-to-peer.

### References

- [1] D I Bevan. Distributed garbage collection using reference counting, 1987.
- [2] Benjamin Goldberg. Generational reference counting: A reduced-communication distributed storage reclamation scheme. In *Proceedings of the SIGPLAN '89 Conference on Programming language design and implementation*, Portland, OR (USA), June 1989.
- [3] José M. Piquer. Indirect reference counting: a distributed garbage collection algorithm. In *PARLE'91—Parallel Architectures and Languages Europe*, number 505 in Lecture Notes in Computer Science, pages 150–165, Eindhoven (the Netherlands), June 1991. Springer-Verlag.
- [4] David Plainfossé and Marc Shapiro. A survey of distributed garbage collection techniques. In *Proc. Int. Workshop on Memory Management*, Kinross, Scotland (UK), September 1995.
- [5] Gerard Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 1994.
- [6] Paul Watson and Ian Watson. An efficient garbage collection scheme for parallel computer architectures. In *PARLE Parallel Architectures and Languages Europe*, June 1987.