

Randomized Distributed Garbage Collection

Kaoutar El Maghraoui, and Travis Desell

Department of Computer Science

Rensselaer Polytechnic Institute

{elmagk,deselt}@cs.rpi.edu

December 10, 2003

Abstract

With the explosive growth of the Internet and distributed computing, distributed garbage collection (DGC) has become a very important area of research. Single-processor garbage collection is by itself one of the challenging areas of research in computer science. In distributed environments, the problem of garbage collection becomes even harder and more challenging. This is mainly due to the additional overhead that is incurred by network communication and global synchronization. While most research efforts in DGC algorithms have focused on designing deterministic and correct algorithms, this work is a first attempt to introduce randomization techniques to DGC. We propose two approaches that could be applied in active objects systems and one approach that could be applied in actors system. Our mathematical analysis has shown that randomization achieves faster algorithms without highly impeding the accuracy and correctness of garbage collection.

1 Introduction

Extensive research has been done in the area of distributed garbage collection [1]. Collecting garbage manually in large scale distributed systems can be very complex, if not impossible. Finding an efficient garbage collection algorithm for distributed systems still remains an open area of research. Distributed systems become increasingly complex due to remote references, inter-site cyclic garbage, communication latency, synchronization and scalability. Maintaining a decentralized global view of both reachable and unreachable objects is difficult mostly because of the following reasons:

- Different nodes of the computation can either fail or experience communication problems with other nodes during the operation of garbage collection. This could lead to an inconsistent state.
- Remote references have a very dynamic nature. They can be created, moved, and deleted. It is difficult to determine if an object is still reachable by any other active object in the system.
- A reference could be identified as garbage and reclaimed when it should not be, ie. while a reference to it is still in transit.
- Distributed garbage collection algorithms often necessitate maintaining a global state and hence global or at least partial synchronization: two traditional problems that are very hard to solve efficiently in distributed systems, especially as the size of the system increases.

Many algorithms have been proposed to tackle the problem of garbage collection in distributed systems. Most of them have emerged as an adaptation of the ideas developed for single-processor, single-address-space systems. In distributed reference counting [3], each public object contains a reference count, which is

incremented every time a reference to it is copied and decremented every time a reference is discarded. To handle remote references, control messages are sent anytime an increment or decrement operation needs to be performed on remote references. However these algorithms can result in the creation of dangling references. In addition to that, they are not scalable since a control message has to be sent any time a reference is created, duplicated or deleted. Many algorithms have been proposed to improve on distributed reference counting by transferring successively more state information to the reference [13]. However, indirect identification schemes do not solve memory leaks caused by cyclic garbage. The concurrent mark and sweep [7] algorithm suffers from the generation of large communication overhead to ensure concurrency of the garbage collection. It also uses a stop the world technique for global synchronization that causes delays proportional to the size of the distributed environment.

These solutions and many other ones are quite slow, mainly due to global synchronization or partitions, which highly hamper performance as all the computers in the partition must globally handshake and agree not to change their state. Slow and non-cooperative sites can also cause extremely poor delays. The indeterministic nature of communication latency and processor load cause these systems to remain very un-scalable.

Traditional distributed garbage collection algorithms focus primarily on soundness and completeness, and in doing so have greater processing and communication requirements. However, in many large-scale distributed systems the occasional failure is not only tolerated, it is the norm. We present in this paper some randomization approaches to distributed garbage collection. We propose a randomized algorithm for distributed reference counting. We also analyze some randomized techniques that attempt to sample efficiently the space of references or actors to be garbage collected based on the lifetime heuristic in case of references and the inactivity heuristic in case of actors. Similar techniques have been used in Generation Reference Counts [7] and Generation Scavenging [10], which use the observation that references usually die young. Our approach differs from them in that we use randomization techniques to achieve faster algorithms.

2 Related Work

2.1 Distributed Reference Counting

Distributed reference counting (DRC) has been the most widely adopted technique to implement distributed garbage collection. In single-processor's reference counting systems, every object is assigned a counter that keeps track of all the references to this object. Any time a new reference to the object is made, the counter is incremented and anytime a reference is destroyed, the counter is decremented. The object is considered garbage and its space is safe to reclaim when its counter becomes zero (when it is not reachable by any other object). Distributed reference counting introduces the concept of remote references. A reference becomes remote when it is sent to another node in the network. In this case, maintaining an accurate view of the reference counters would require the exchange of messages between nodes when new references are created or deleted to a remote object. In a distributed environment, additional issues emerge such as latency, delays, message losses, and synchronization. Naive extensions of single-processor RC are not enough to address all the issues associated with distributed systems. For example, a race condition between increment and decrement messages may cause the counter of an object to be decremented before it is incremented. This could result in having a counter of zero temporarily and garbage collecting the object while it is still referenced by some other remote locations.

Different algorithms have been proposed to address the issues of DRC. Lermen and Maurer [9] addressed the issue of prematurely decrementing counters by introducing create and acknowledge messages in addition to increment and decrement messages. Any time a reference is copied, a create message is sent to the reference owner. This latter increments the reference counter and sends an acknowledgement message to the sender. Similarly when a reference is deleted a delete message is sent to the reference owner. Decrement messages can only be sent when the number of references that exist is equal to the number of acknowledgements

sent. Birell et al. [5] proposed a similar algorithm that avoids the decrement race condition by using acknowledgements and more synchronization between the distributed garbage collector and mutators.

Another scheme was proposed in [4] that attempts to solve the race conditions of increment/decrement messages by using weighted reference counting. In this scheme, a weight is initially assigned to a reference. The weight of the object is the sum of all the weights of its references. When the first reference is created, the object and its reference are given the same weight. When a reference is deleted, a message is sent to the reference owner with the reference weight, which gets deleted from the total weight. When the reference is copied, the weight gets equally divided between the two copies. One problem that might occur with this algorithm is to reach the case when the reference weight cannot be divided any further. Indirect Reference Counting is another technique for garbage collection. This algorithm does not use increment messages. It relies instead on a representation of the path along which the references get propagated in a distributed system. The path is referred to as a diffusion tree where every process should maintain a reference counter for each reference it holds. The discussed algorithms for DRC result in many messages being exchanged between the different nodes in the distributed system rendering DRC a very expensive and non-scalable algorithm.

2.2 Active Objects Garbage Collection

Active objects are widely used in distributed systems. Objects and processes associated with them are both integrated in one entity called an active object or actor. Garbage collection in active object systems is a very crucial task because active objects consume not only memory but also other resources such as network, CPU, and storage resources. An object is considered garbage in these systems when it cannot potentially call a root object, nor be called by a root object. This makes detecting garbage in active objects systems a significantly more difficult task. Kafura [8] proposed a centralized algorithm to address GC in active object systems that is based on the marking and sweep algorithm. This algorithm marks objects using three different colors (white, grey, and black). White objects cannot interact with any root object. Grey objects are sleeping objects who could potentially interact with a root object if they become activated. Black objects are non-garbage. The algorithm builds a reachability graph of the different objects in the system. Initially all objects are colored white except for the root objects. Then according to a set of rules that depend on the reference structure of objects, the algorithm marks the objects with either grey or black until no possible marking is done. At the end, all white objects are considered garbage and they are safe to reclaim. To remove the bottleneck associated with centralization, Isabelle [11] proposed a distributed extension to Kafura's algorithm. In this algorithm, a set of local garbage collectors are loosely coupled with a global garbage collector that maintains a global snapshot of the references in the system's state.

3 Approach

The major problems in implementing a distributed garbage collection scheme are minimizing its impact and achieving high levels of scalability. To minimize the impact of the garbage collection scheme we must minimize network communication and avoid pausing the computation (as it is done in algorithms requiring partial or global synchronization). Eliminating synchronization between nodes in the distributed system also allows the garbage collection scheme to become much more scalable, as each additional node required to be synchronized involves a large amount of additional overhead. Coordinating a garbage collection scheme between a large number of nodes is not only extremely difficult but also very time consuming due to the amount of network communication required.

We propose three approaches to dealing with these issues through randomization: Lifetime-Based Garbage Collection (LBGC), Randomized Reference Counting (RRC), and Dormancy-Based Garbage Collection (DBGC). The first two approaches are suitable for references and active objects in general, while the third approach is more suitable for actors. LBGC tries to sample the references space using the assumption that

references usually die very young or very old. Once a sample has been determined, a deterministic distributed garbage collection algorithm can be used to collect references pertaining to this sample. This approach will improve the performance of garbage collection if the sample chosen has a very high percentage of garbage. By reducing the sample of the references to collect, we reduce the overhead of GC and also the overhead of network communication. RRC requires some inter-object (and thus network) communication, but seeks to minimize this communication through randomization and delayed partial broadcasts. RRC will provide greater accuracy and quicker removal of garbage objects, at the cost of a more complex garbage collection scheme and more network communication. RRC also allows for the collection of active cyclic garbage, a very difficult problem in current distributed garbage collection strategies. Dormancy-Based Garbage Collection uses the inactivity of an actor's mailbox to predict whether it is a garbage or not. This approach is similar to LBGC algorithm. This heuristic is also used to reduce the actor space that will be collected.

3.1 Lifetime-Based Garbage Collection (LBGC)

Generational garbage collection algorithms have improved many garbage collection algorithms using heuristics that predict the lifetime of references and improve the caching and paging behavior [12]. These algorithms are based on the assumption that young objects have a shorter life expectancy than old objects. Generational garbage collectors manage to reduce the overhead of garbage collection by cleverly selecting the subsets or generations of references that will be garbage collected. The choice of the generation to collect is very crucial. If these algorithms select a generation that happens to have a very low percentage of garbage, the performance of the algorithm decreases. In real programs, the assumption that young references have a short life expectancy is usually true. However once a reference becomes old, chances that it is a dead reference increase also. We propose a randomized garbage collection algorithm that is based on the assumption that in young and old generations of references, the probability of having garbage is high. Therefore, a good sample for garbage collection should include both young and old generations.

3.2 Randomized Reference Counting (RRC)

In RRC, each object contains a list of references to other objects (this is trivial as most objects already contain this information). Periodically, that object will send keep alive messages to the objects it has references to, additionally these keep alive messages will specify if the object is root reachable or not. Each object keeps track of the objects that have references to it, and if those objects are root reachable or not. In this manner we can determine when an object is not reachable by the root and therefore garbage.

Using a method in which each object broadcasts keep alive messages to all of its references will result in a massive amount of overhead. Race conditions must also be dealt with, adding even more overhead (as is done in normal distributed reference counting). To avoid these problems, an object periodically chooses references randomly to send keep alive messages to. Each period, an object removes references based on the probability that the reference has been removed. We know this probability due to the probability of the referee sending a keep alive every period. Additionally, we add a "fudge factor" to this probability, to deal with race conditions and errors in the network which could delay (or prevent) the reception of a keep alive message. In this manner we minimize the overhead of distributed reference counting, while still obtaining a tight bound on the probability of a reference (and thus the probability of an object) being garbage.

Additionally, RRC can also make use of a two-tiered system of garbage collection, serializing objects to disk when they reach a certain threshold of being garbage and then finally collecting them from the disk after a much more stringent threshold has been reached.

3.3 Dormancy-Based Garbage Collection (DBGC)

In the previous two approaches, we presented randomized algorithms for garbage collection for references. DBGC is a randomized algorithm for collecting garbage in active actors systems. Actors [2] are entities

that encapsulate both the process and its state. Each actor provides a communication interface to other actors. Each actor has a unique name, which is used as a reference by other actors and a message box where messages from other actors are placed to be processed later on. Communication between actors is purely asynchronous. DBGCC also tries to sample a good collection by randomly selecting a subset that is most likely to have a very high percentage of dead actors. The heuristic that is used in this algorithm is the inactivity of the mailbox of an actor. The assumption used here is that the longest time that an actor has not received any message, the higher the probability that it is a dead actor. The algorithm will be triggered in rounds once a certain threshold for the number of actors present is reached.

4 Randomized Reference Counting

4.1 The protocol

Given a set of variables:

A	An active object
G	The set of all current garbage objects
RR	The set of all objects reachable by the root
	$RR = \tilde{G}$
$S = \{s_1, s_2, \dots, s_n\}$	A set of n known source references, $s_i = 1$ if the source is root reachable $s_i = 0$ otherwise
$T = \{t_1, t_2, \dots, t_m\}$	A set of m target references

And user-defined probabilities:

M_l	The probability of any message being lost
M_{ns}	The probability of a <i>update reference</i> message not being sent.
T	The threshold probability, ie. when an actors probability of being garbage is greater than this, that actor is collected.

And the following protocol:

- Once per period, each active object A sends an *update reference* message to each known target reference $t_j \in T$, with probability $1 - M_{ns}$. This message contains a bit set to 1 if A is root reachable, 0 otherwise.
- If A receives a new reference, add the target of that message to T and send a *new reference* message to the target of that reference, with a bit set to 1 if A is root reachable, 0 otherwise.
- If A receives a *new reference* message, do not collect it at the end of this period, and add the source of that reference to S as s_i set to 1 if the source of the *new message* was root reachable, 0 otherwise.
- A reference t_i or s_j is removed from T or K , respectively, if the source of t_i or the target of s_i is collected.
- If at the end of a period which no *new reference* messages have been received, $Pr(A \in G) > T$, collect A .

We can define the probability that actor A is garbage in terms of the probability of not receiving a *new reference* message if a reference was propagated (NM) and the probability that the sources of all s_i are RR .

$$Pr(A \in G) = Pr(\text{no } S \in RR) - Pr(NM)$$

$Pr(NM)$ is the probability of at least one reference being propagated and A not receiving a *new message* from the active object that reference was propagated to. In any period, any reference can be propagated to any number of other active objects. If $O = \{o_1, o_2, \dots, o_n\}$ is the set of all objects and $NM(o_i)$ is the probability that there was a reference propagated to o_i and no *new message* was received by A,

$$Pr(NM) \leq NM(o_1) \times NM(o_2) \times \dots \times NM(o_n)$$

Since the probability of a reference being propagated to another object and no message being received is simply M_l , $Pr(RP)$ can be bounded as follows:

$$Pr(NM) \leq M_l$$

We can define $Pr(\text{no } S \in RR)$ as the probability that no source of s_i is in RR ,

$$Pr(\text{no } S \in RR) = Pr(s_1 \notin RR) \times Pr(s_2 \notin RR) \times \dots \times Pr(s_n \notin RR)$$

with the probability that $s_i \notin RR$ if A received a message from the source of s_i in the last period as:

$$Pr(s_i \notin RR) \leq 1 - |s_i - M_l|$$

and otherwise, where $Pr(P_{ns,i})$ is the number of periods that A has not received an *update message* from the source of s_i ,

$$Pr(s_i \notin RR) \leq 1 - (1 - M_l - M_{ns})^{P_{ns,i}}$$

4.2 Protocol Analysis

To measure the performance of RRC, two analyses are provided. Firstly, we measure the impact RRC will cause on a distributed system, in terms of message sending. Secondly, we measure the expected number of periods for an object to be collected once it has become garbage, and bound this value.

The RRC protocol sends $M_{ns} * R$ messages each period, where R is the number of references in the system. Additionally, each time a reference is propagated, another message send is incurred, so the upper bound on message sending per period, where O is the number of active objects, is:

$$O(\ln(M_{ns} * R + R * O)) \text{ or } O(\ln(R * O))$$

The expected time for an object to be collected once it's in the set of garbage objects can be found as follows, using the fact that an active object is collected only if the probability that it is garbage is greater than the threshold.

$$T \leq \prod_{i=0}^n E(Pr(\text{no } S \in RR)) - M_l$$

$$E(Pr(s_i \notin RR)) = (1 - |s_i - M_l|) * (1 - M_l + M_{ns}) + (M_l + M_{ns}) * (1 - (1 - M_l - M_{ns})^{P_{ns,i}})$$

If this object $\in G$, then $s_i = 0$. With $M = M_l + M_{ns}$,

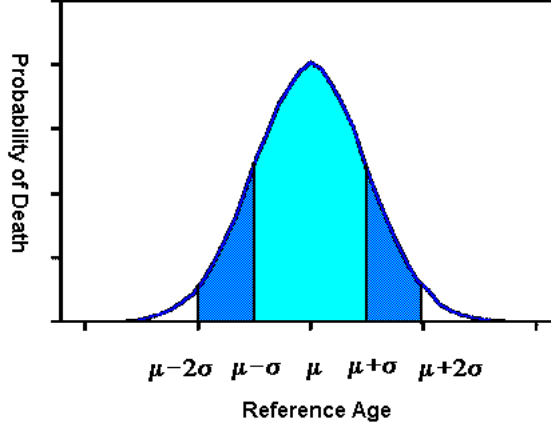


Figure 1: Modeling the probability of lifetime of references using the Normal Distribution. The figure shows a normal distribution with mean μ and variance σ

$$E(\text{Pr}(s_i \notin RR)) = (1 - M_l) * (1 - M) + M * (1 - (1 - M)^{P_{ns,i}})$$

$P_{ns,i}$ can be seen as the number of periods that have elapsed since the object has become garbage. Therefore $P_{ns,i}$ is the same for every i and we label it as P_{ns} . Solving for P_{ns} gives:

$$T \leq \prod_{i=0}^n ((1 - M) * (1 - M_l) + M * (1 - (1 - M)^{P_{ns}})) - M_l$$

$$P_{ns} = \ln(1 - \frac{e^{(T - M_l)/N} + (1 - M) * (1 - M_l)}{M}) / \ln(1 - M)$$

5 Lifetime Based Randomized Garbage Collection

5.1 The Algorithm

We have chosen the normal distribution to model mathematically the lifetime probability of references. The normal probability distribution describes the life expectancy of every reference. As Figure 1 shows, the youngest and oldest references have a very short lifetime probability. In other words, these generations are expected to have a very high percentage of dead references.

Let μ be the mean of the distribution and σ be the variance. If a reference i has age x_i , then the probability that this reference will be alive is:

$$\Pr[x_i] = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (1)$$

Garbage collection will be triggered periodically once the available storage reaches a certain threshold. Let S be the set of all references that still exist in memory before a certain garbage collection round. The list S will be sorted increasingly according to the age of the references. In other words, the lowest indexes will contain the youngest references and the highest indexes will contain the oldest references. The mean μ and the variance σ of the set S will be calculated for each round. Let S_1 be the set of references whose age belongs to the interval $[\mu - \sigma, \mu[$ and S_2 be the set of references whose age belongs to the interval $]\mu, \mu + \sigma]$ (see Figure 2).

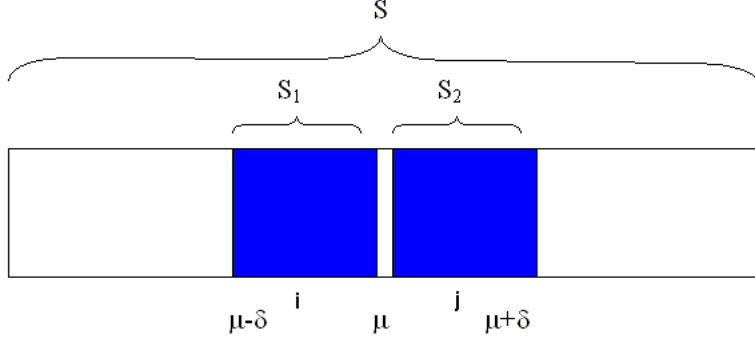


Figure 2: Structure of the sorted set of references S

The algorithm proceeds as follows:

For each round, do

1. Sort S based on the age of the references.
2. Calculate the mean time μ and the variance σ of S
3. Pick two random values i and j from the sets S_1 and S_2 respectively.
4. If coin = HEADS then garbage collect using a deterministic distributed garbage collection algorithm (DBGC) all references whose indexes are less than i else garbage collect all references whose indexes are greater than j using a DBGC algorithm

5.2 Algorithm Analysis

In this section we will try to bound the probability that the references in the subset chosen are alive or not garbage. Let p be the probability of choosing the references i and j from the subsets S_1 and S_2 . Let N be the size of S . The probability P that an element chosen for collection is not garbage is at most:

$$P = \frac{1}{2} \times p \times \sum_{k=j}^N \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x_k - \mu)^2}{2\sigma^2}} + \frac{1}{2} \times p \times \sum_{k=1}^i \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x_k - \mu)^2}{2\sigma^2}}$$

$$P = \frac{1}{2\sigma\sqrt{2\pi}} p \left[\sum_{k=j}^N e^{-\frac{(x_k - \mu)^2}{2\sigma^2}} + \sum_{k=1}^i e^{-\frac{(x_k - \mu)^2}{2\sigma^2}} \right]$$

If we assume that $\sigma > 1$, we have $\frac{p}{\sigma} < 1$. We also have $\sum_{k=j}^N e^{-\frac{(x_k - \mu)^2}{2\sigma^2}} + \sum_{k=1}^i e^{-\frac{(x_k - \mu)^2}{2\sigma^2}} < 1$. We conclude that $P < \frac{1}{2\sqrt{2\pi}}$.

6 Dormancy Based Randomized Garbage Collection

6.1 The Algorithm

Let S be the list of actors present before a given collection round. Let N be the size of S .

1. Sort the list S based on the inactivity time of actors. The list should be sorted in increasing time of inactivity.
2. Randomly pick an index j from $1 \dots N$.
3. Apply a deterministic actor garbage collection algorithm to the subset S_j that consists of all actors whose index is greater than j .

6.2 Algorithm Analysis

In this section, we will try to bound the expectation of the number of active actors that are still alive in the subset chosen for garbage collection. To model a system of active actors, we choose the radioactive delay model. This model has been proved to be useful only for long-lived objects [6]. In this model, a probability distribution function describes the life expectancy of every object. The model uses a parameter h called the half life cycle. For every object that is alive at time t_0 , the probability that this object will be alive at time $t + t_0$ is $2^{-t/h}$. The probability that this object will be dead is $1 - 2^{-t/h}$. Using this model in an actor system, if an actor has a mailbox inactivity of a duration d , then the probability that it is a live actor is $2^{-d/h}$. Note that as the inactivity duration d increases the probability that the actor is alive decreases. So the radioactive decay model gives a good representation of a system of actors.

Let X_{ij} be a random variable of a given actor i in the sorted list that belongs to the randomly chosen subset S_j . This random variable is defined as follows:

$$X_{ij} = \begin{cases} 1 & \text{,if actor } i \text{ is alive} \\ 0 & \text{,otherwise} \end{cases}$$

The expected number of live actors in the randomly selected subset is $\sum_{j=1}^k \sum_{i>j}^k E[X_{ij}]$ since all these random variables are independent.

$$\begin{aligned} \mathbf{E}[X_{ij}] &= p_{ij} + (1 - p_{ij}) \times 0 \\ &= p_{ij} \\ &= \frac{1}{N} \times 2^{-i/h} \end{aligned}$$

Let us try to bound this expectation:

$$\begin{aligned} \sum_{j=1}^N \sum_{i>j}^N E[X_{ij}] &= \sum_{j=1}^N \sum_{i>j}^N \frac{1}{N} \times 2^{-i/h} \\ &= \sum_{j=1}^N \sum_{k=1}^{N-j} \frac{1}{N} \times 2^{-j-k/h} \\ &= \frac{1}{N} \sum_{j=1}^N 2^{-j/h} \left(\sum_{k=0}^{N-j} 2^{-k/h} \right) \\ &= \frac{1}{N} \sum_{j=1}^N 2^{-j/h} \left[\frac{1 - 2^{-\frac{N-j+1}{h}}}{1 - 2^{-1/h}} \right] \\ &= \frac{1}{N(1 - 2^{-1/h})} \left[\sum_{j=1}^N (2^{-j/h} - 2^{-\frac{k+1}{h}}) \right] \end{aligned}$$

$$\begin{aligned}
&= \frac{1}{N(1 - 2^{-1/h})} \left[\frac{1 - 2^{-\frac{N+1}{h}}}{1 - 2^{-1/h}} - 1 - N \times 2^{-\frac{N+1}{h}} \right] \\
&= \frac{1}{N(1 - 2^{-1/h})} \times 2^{-1/h} \left[\frac{1 - 2^{-N/h}}{1 - 2^{-1/h}} - N(2^{-N/h}) \right] \\
&\leq \frac{1}{N(2^{1/h} - 1)}
\end{aligned}$$

We conclude that the expected number of live objects in the randomized subset is very small.

7 Conclusions and Future Work

We have provided mathematical analysis for the different proposed approaches. Our analysis has proved that randomization can be used to speed up currently existing garbage collection algorithms, when soundness of the algorithm is not necessary.

RRC shows that much of the complexity of the reference counting algorithm can be removed when the more complicated issues are bounded by certain probabilities. Additionally, the RRC protocol can be tuned by a user to collect garbage as efficiently as possible. By increasing the period in which the protocol operates by, the number of additional messages incurred by the protocol can be decreased. Also, the user can determine how accurate the garbage collector is, at the cost of collecting garbage more slowly. Due to message loss being built into the protocol, RRC can operate under less than perfect conditions in which messages are lost frequently, as well as use faster protocols such as UDP. RRC as it is currently developed uses some loose bounds, which could be tightened to increase its accuracy in garbage collection and further improve the speed of the algorithm.

Both the LBG and DBG approaches show that the reduced space has a very large percentage of garbage. Very few dead objects are expected to remain in the other objects not sampled. If they are garbage, their age or inactivity time will be increased and they will be collected later on in future rounds. This approach is very useful in improving the performance of garbage collection as well as improving paging and reducing memory fragmentation. Parameters such as the frequency of the rounds of garbage collection could be tuned depending on the underlying systems, the frequency of communication, and the available resources.

As future work, We plan to empirically test the results of the proposed protocol and algorithms RRC using the SALSA programming language and WWC middleware; to see how the algorithms could be improved even more.

References

- [1] S. E. Abdullahi and G. A. Ringwood, "Garbage collecting the Internet," Tech. Rep., 1996. [Online]. Available: citeseer.nj.nec.com/182790.html
- [2] G. Agha, I. A. Mason, S. F. Smith, and C. L. Talcott, "A foundation for actor computation," *Journal of Functional Programming*, vol. 7, no. 1, pp. 1–72, 1997. [Online]. Available: citeseer.nj.nec.com/agma98foundation.html
- [3] N. AK, "A storage reclamation scheme for applicative multiprocessor systems," 1979.
- [4] D. I. Bevan, "Distributed garbage collection using reference counting," 1987.
- [5] A. Birrell, D. Evers, G. Nelson, S. Owicki, and E. Wobber, "Distributed garbage collection for network objects," 130 Lytton Avenue, Palo Alto, CA 94301, Tech. Rep. 116, 1993. [Online]. Available: citeseer.nj.nec.com/birrell93distributed.html

- [6] W. D. Clinger and L. T. Hansen, “Generational garbage collection and the radioactive decay model,” in *SIGPLAN Conference on Programming Language Design and Implementation*, 1997, pp. 97–108. [Online]. Available: citeseer.nj.nec.com/clinger97generational.html
- [7] B. Goldberg, “Generational reference counting: A reduced communication distributed storage reclamation scheme.” in *Proceedings of Programming Languages Design and Implementation, ACM SIGPLAN Not.24, 313-321.*, 1989.
- [8] D. Kafura, D. Washabaugh, and J. Nelson, “Garbage collection of actors,” 1990.
- [9] C.-W. Lermen and D. Maurere, “A protocol for distributed reference counting,” 1986.
- [10] K. A. Mohammed-Ali, “Object oriented storage management and garbage collection in distributed processing systems,” 1984.
- [11] I. Puaut, “A distributed garbage collector for active objects,” in *Proc. Ninth Annual Conf. on Object-Oriented Programming Systems, Languages, and Applications*, vol. 29, no. 10, Portland OR (USA), 1994, pp. 113–128. [Online]. Available: citeseer.nj.nec.com/article/puaut94distributed.html
- [12] P. M. Sansom and S. L. P. Jones, “Generational garbage collection for haskell,” in *Functional Programming Languages and Computer Architecture*, 1993, pp. 106–116. [Online]. Available: citeseer.nj.nec.com/sansom93generational.html
- [13] I. Watson, P. and Watson, “An efficient garbage collection scheme for parallel computer architectures,” in *Proceedings of the PARLE '87- Parallel Architecture and Languages Europe, LNCS 259, Springer, 432-443*, 1987.