Fortune Teller: Improving Garbage Collection Performance in Server Environment using Live Objects Prediction *

Feng Xian , Witawas Srisa-an and Hong Jiang
Deptartment of Computer Science and Engineering
University of Nebraska-Lincoln
Lincoln, NE 68588-0115
{fxian, witty, jiang}@cse.unl.edu

ABSTRACT

Currently, the most adopted criterion to invoke garbage collection is heap space exhaustion. In other words, garbage collection is invoked when the heap space (either the entire space or generational space) runs out. A possible alternative but much more difficult approach is to invoke garbage collection when good garbage collection efficiencies can be obtained. To do so, we need to know number of garbage objects that can be collected before the collection actually takes place.

In this short abstract, we introduce a predictive approach that can provide an accurate estimation of the number of dead objects at any specific point of execution. The proposed estimation model relies on the information obtained from partial reference counting. Our plan is to use this information as a criterion to invoke garbage collection. We have conducted a preliminary study to determine the feasibility of this idea and found that the model is sufficiently accurate in three SPECjvm98 benchmark applications.

Categories and Subject Descriptors

D.34 [**Programming Languages**]: Memory management

General Terms

Languages, Measurement, Performance

Keywords

Predictive model of garbage collection

1. INTRODUCTION

The goal of this research is to improve the performance and efficiency of garbage collection in Java and .NET server applications. Our recent study of SPECjAppServer2002 and SPECjbb2000 finds that garbage collection is a major performance bottleneck that limits the scalability of server applications. A recent study by [1] found that an application that utilizes garbage collection would need on average 5 times more heap size than the same application with explicit memory management. Moreover, the reduction in the additional space (or "headroom") would degrade the garbage collection performance by as much as 70%. Currently, large application servers are tuned to provide a large amount of headroom to yield good average case performance. However, a significant and unexpected increase in the workload would cause these servers to slow down to a crawl or even crash. This is because the desired amount of headroom to maintain decent garbage collection performance is no longer available.

Our recent experiment also found that half of objects in the server applications are not short-lived [3]. In such scenario, the really short-lived objects would have died and festered for a long time before they are collected. We are currently investigating two solutions to address this problem. First, we further segregate objects by lifespan and manage them in two young generations of different sizes. Second, we develop a predictive model called Fortune Teller that at any time, can estimate the amount of objects that can be collected. This information is then used to invoke garbage collection at instances that would yield high efficiency so that the headroom is reduced as unused memory are recycled more efficiently. The second approach is the focus of this short abstract.

2. GOAL STATEMENT

The basic principle behinds Fortune Teller is very straight forward but yet unexplored. We want to use partial reference counting to dynamically predict the amount of space that can be reclaimed if garbage collection is invoked right now. This is a predictive model because typically, the garbage collection efficiency (a ratio of the amount of space collected over the amount of space used prior to a collection) is not known until the collection is done. If this value can be accurately predicted, it can be a powerful criterion to improve the overall garbage collection performance and predictability and reduce memory usage. We propose to use partial reference counting as a way to achieve this goal. In partial reference counting, when an object is freed, we do not scan the object to identify other references that may be in the object to recursively reduce their reference count. Recursive freeing is known to be expensive and nondeterministic. For estimation purpose, we have selected to only perform partial reference counting to reduce overhead and make the operation simple.

Our initial research goal is to study the feasibility of using partial reference counting as the predictor. The first chal-

*This work is supported by NSF ITR grant CNS-0411043. Any opinions, findings and conclusions expressed herein are the authors and do not necessarily reflect those of the sponsors.

OOPSLA'05, October 16–20, 2005, San Diego, California, USA.
ACM 1-59593-193-7/05/0010.

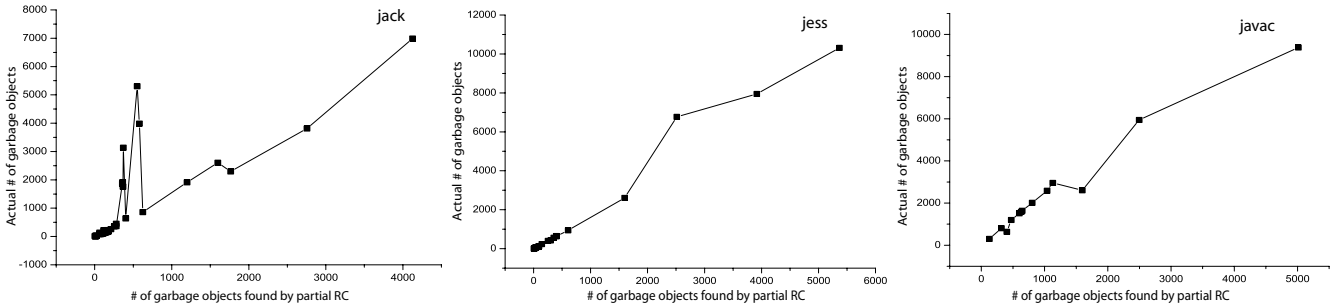

Figure 1: Comparison between partial RC and full RC

lenge is to evaluate the accuracy of the predictor by comparing it to the oracle, which is the result of full reference counting. If we find that partial reference counting is a good predictor, the second challenge is to estimate the its cost and explore other alternatives to further reduce cost. The next section reports the preliminary results of our study.

3. APPROACH

We adopt Merlin algorithm [2] as a way to generate the exact object lifespan. Merlin algorithm was introduced as an alternative to the brute force approach[1]. It relies on the last access time as a way to determine lifespan. As a preliminary study, we created lifespan traces using Merlin algorithm for three benchmark programs in SPECjvm98.

We then conduct simulations of partial reference counting and compare its ability to identify garbage with the actual results provided by Merlin algorithm. We fully anticipate that the number of garbage objects identified by partial reference counting would be smaller than the actual number of garbage objects. However, our goal is to investigate whether the differences can be approximated by mathematical equations. The result of our studies preliminarily indicated that the differences in each application can be described using a linear equation.

For example, the differences tend to be a factor of 1.5 for jess, 2.5 for javac, 1.25 for jack (shown in Figure 1). Figure 2 present the ratios of this three benchmarks as box plot. The middle horizonal line is the average value. The upper and lower edges of the box define the standard deviation and the whiskers represent the maximum and minimum values. We can easily see that the ranges of the standard deviation are very narrow. The average ratios for jess, javac and jack are 1.5, 2.5 and 1.25, respectively.

So far, the preliminary simulation results are quite promising. Our next steps to further investigate this idea include the following research activities:

1. Generate lifespan traces for server benchmarks such as SPECjAppServer2004 and SPECjbb2005 and again, compare the accuracy of partial reference with the actual results. A predictive model can then be integrated into the prototype system for further investigation.

2. It has been shown that the slope in our predictive model varies from one application to the next. Thus, we need to be able to determine this value dynamically. Also, we then need to perform empirical study to evaluate the overhead of the proposed predictive model and its effect on the overall execution time.

3. Validate the efficiency of different predictive models. Also, We need to investigate the ramification when the predictive model does not work. Can it severely degrade the performance? Can we detect and prevent such scenarios from taking place?

[1]Brute force approach invokes garbage collection after every object allocation to determine the exact lifespan of objects [2].

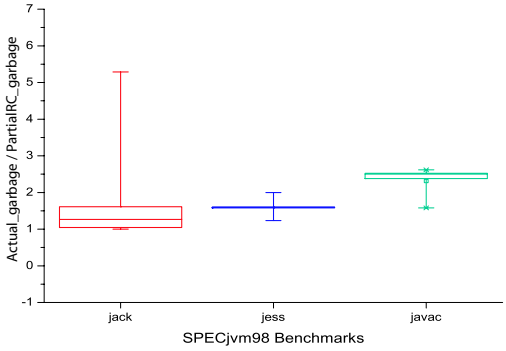

Figure 2: Discrepancy ratio of full/partial RC

4. REFERENCES

[1] M. Hertz and E. Berger. Quantifying the performance of garbage collection vs. explicit memory management. In *OOPSLA '05: 20th annual ACM SIGPLAN conference on Object-oriented Programming Systems, Languages, and Applications*, 2005.

[2] M. Hertz, S. M. Blackburn, J. E. B. Moss, K. S. McKinley, and D. Stefanovi. Error-free garbage collection traces: how to cheat and not get caught. In *SIGMETRICS '02: Proceedings of the 2002 ACM SIGMETRICS conference*, pages 140–151, New York, NY, USA, 2002. ACM Press.

[3] W. Srisa-an and M. Oey. Remote Objects: The Next Garbage Collection Challenge. *Journal of Object Technology*, 4:155–172, May 2005.