A Study of Garbage Collection API adapted to Real-Time Environments

Toshinori Yasoshima

School of Information Science,
Japan Advanced Institute of Science and Technology

February 15, 2000

Keywords: garbage collection, real time, Java, API, Kaffe OpenVM.

1 Background

In recent years, programming language Java has been coming into use widely, because Java has many software engineering benefits such as security features in distributed environments, reusability of code, and portability to other platforms. So, even for real-time systems or embedded ones, programmers hope that they can write code in Java.

But Java execution environment and language specification are not sufficient to support real-time requirements. One reason is Java's garbage collector (GC for short). Java's GC relieves programmers from manual memory management. Programmers do not have to worry about memory management, since GC automatically manages heap memory. Therefore GCs greatly improve the maintainability of Java code.
On the other hand, GCs become cumbersome when using Java for real-time systems, since GCs make it difficult to guarantee the maximum time of response time and the remaining heap size, and to estimate or predict the whole processing time of GC and so on.
We must consider the following requirements when applying Java to real-time systems.

- To satisfy real-time requirements.
- To provide sufficient memory.
- To preserve software engineering benefits such as portability, reusability, and maintainability.

Many garbage collection algorithms for real-time systems have already been proposed, but they are not good enough. This is because the performance of GCs heavily depends on memory usage pattern or life time of objects, but almost all algorithms do not care about this problem. We observed this is because they provide no way for users to customize themselves. Our idea is to provide GC, which is customizable, for example by allowing users to change some parameters of GC. However, improving performance by changing GC parameters have not been studied well. Therefore we expect it may achieve great improvements to adapt GC to underlying environments or applications by changing GC parameters.

2 Purpose

In this paper, we propose and implement garbage collection API (GC API for short) on Java extended to real-time environments. The GC API allows users to change GC parameters or behaviors so that it is possible to adapt to real-time environments. The GC API consists of the following three functions:

- Specifying execution interval and priority of GC

  The GC, which is invoked at the same interval (called Timed-GC), has been already proposed. We use a GC based on Timed-GC and implement this API to change the interval. The shorter execution interval is, The more amount of memory are reclaimed and The more overhead are, and vise versa. Thus, GC can achieve improvement in performance by specifying suitable interval.

- Allowing or disallowing to invoke GC

  There are time critical sections where even the overhead of write-barrier is not permitted. In such cases, it is important to guarantee that GC is not invoked. This API provides this feature.

- Reserving memory which GC does not manage

  In the following situations, programmers may want to manage heap memory explicitly:

  * In the case that a real-time constraint requires fast allocation/deallocation, since it takes extra cost to allocate/deallocate heap memory via GC.
  * In the case that memory which GC does not manage is required because of disallowing to invoke GC.
  * In the case that some amount of memory should be guaranteed.

  This API satisfies the above requirements, although the programmer is burdened again with manual memory management and must take care not to violate several conditions. For example, objects on manually allocated memory cannot include any references of objects on GC memory.

Using the GC API, programmers can change parameters of GC depending on each applications. Setting suitable parameters, we expect to reduce blocking time of execution of applications by interrupting GC.

3 Implementation Outline

In this research, we use Kaffe-1.0.b1-rt which is a Java VM customized for real-time environment. However the GC in Kaffe use non-incremental mark-sweep algorithm, this algorithm is not appropriate for real-time environments, since we can not estimate a blocking time by interrupting GC. Therefore we implement a GC based on Timed-GC which can be triggered by time and suspended any time by programmer.
To implement the GC, we made as follows changes:

- Changing the GC on Kaffe to concurrent GC
- Implementing write-barrier to use memory protection mechanism supported on on Operating System RT-Mach
- Adding timer thread to invoke GC every interval

And we implement three functions of GC API as follows:

- Specifying execution interval and priority of GC

  We use the timer thread to change execution time of the GC.

  We implement to change priority of GC using obtained thread ID of the GC. The thread ID is obtained using `kaffe_thread_interface` function.

- Allowing or disallowing to invoke GC

  We implement disallowing to invoke GC by suspending GC thread. Inversely allowing to invoke GC by resuming GC.

- Reserving memory which GC does not manage

  This function is not implemented yet, designed only.

4 Conclusion

We proposed GC API which allows users to customize GC parameters or behavior to satisfy real-time requirements. We showed the feasibility of GC API, although experimental implementation is unfortunately not completed yet. To show the usefulness of GC API, we need to complete our implementation and to show many examples where the performance can be improved using GC API.