메모리 분할을 이용한 효과적인 가비지 컬렉션에 관한 연구

허서경, 이승룡
경희대학교 컴퓨터공학과
e-mail:{skheo, sylee}@oslab.khu.ac.kr

A Study on Effective Garbage Collection using Memory Partitioning

Seo-Kyung Heo, Sungyoung Lee
Dept of Computer Engineering, Kyung-Hee University

요 약

자바는 플랫폼 독립성, 이식성, 보안, 멀티 쓰레드 지원, 동적 적재, 자동화된 메모리 관리(Garbage Collection) 등 많은 장점을 갖는 언어이다. 특히, 가비지 컬렉터(Garbage Collector)는 메모리 누수(memory leak), 동강난 포인터(dangling pointer) 등과 같은 메모리의 잘못된 사용으로 인한 버그로부터 프로그래머를 자유롭게 하며, 디버깅의 용이함, 개발비용의 절감, 프로그램의 일관성 및 견고성의 향상 등의 이점을 얻을 수 있다. 그러나, 자바 가상머신(Java Virtual Machine)에서 가비지 컬렉터가 객체를 추적(tracing)하고 수집(collecting)하는 작업은 프로그램의 수행 성능을 저하시키는 요인이 된다. 따라서, 본 논문에서는 가비지 컬렉터의 성능을 향상시키기 위하여, 힙(heap)에 할당하는 객체들의 특성을 고려하여 메모리를 분할한 후, 효율적으로 컬렉션 작업을 수행 할 수 있는 기법을 소개한다.

1. 서론

자바는 플랫폼 독립성, 이식성, 보안, 멀티 쓰레드 지원, 동적 적재, 가비지 컬렉션(Garbage Collection) 등 많은 장점을 갖는 언어이다. 특히, 가비지 컬렉션을 통해 자바 응용 프로그램 개발자들은 동적 메모리를 사용하기 위해 명시적으로 메모리 할당을 요청하지만, 그에 상응하는 메모리 해제 작업을 수행하지 않는다. 이러한 자동화된 메모리 관리는 기존의 정적 메모리 할당이나 사용자의 명시적인 할당/반환 정책에 비해 많은 장점을 지닌다[1]. 이를 통해, 프로그래머는 C나 C++로 작성된 프로그램에 비해 메모리 사용과 관련된 버그로부터 자유로울 수 있으며, 디버깅의 용이, 개발비용의 절감, 프로그램의 일관성 및 견고성의 향상 등의 이점을 얻을 수 있다. 그러나, 가비지 컬렉터(Garbage Collector, GC)가 객체를 추적(tracing)하고 수집(collecting)하는 작업은 오버헤드를 유발한다. 따라서, 최근의 가비지 컬렉터들은 이러한 오버헤드를 줄이기 위하여, 힙(heap) 객체들을 특정 기준에 따라, 각각 독립적으로 분할된 영역에 구분하여 할당하는 방법을 사용한다. 예를 들면, 세대별 가비지 컬렉터는 객체의 나이(age)를 기준으로 별도의 영역으로 분할하여, 나이가 어린 객체들이 할당된 메모리 영역에 대하여 집중적으로 컬렉션 작업을 수행한다. 그러나, 세대별 가비지 컬렉션은 일반적으로 수명이 긴 객체들에서는 그 효율이 떨어진다.

본 논문에서는 객체를 할당할 메모리를 분할함에 있어서 객체의 나이에 국한하지 않고, 좀 더 다양한 객체 특성 정보를 이용하여 메모리 영역을 좀 더 세분화하여 분할함으로써 가비지 컬렉션의 대상이 되는 영역을 최소화하여, 프로그램의 수행 성능을 향상시킬 수 있는 메모리를 분할을 통한 효과적인 가비지 컬렉션 기법을 소개한다.

이후의 본 논문의 구성은 다음과 같다. 2장에서는 메모리 분할(partitioning)을 통한 메모리 관리 기법과 객체의 특성에 관한 연구들을 살펴보고, 3장에서는 객체의 특성과 메모리 분할 기법을 이용한 효과적인 가비지 컬렉션를 제안한다. 마지막으로, 4장에

서 결론 및 향후 과제를 설명한다.

2. 관련 연구

자바 가상머신(Java Virtual Machine, JVM)이 사용하는 모든 메모리 영역을 대상으로 가비지 컬렉션을 수행하는 것은 시간이나 공간적인 비용이 너무나 크기 때문에, 최근에는 이러한 오버헤드를 최소화하기 위한 방법으로 힙 메모리를 여러 영역으로 분할하여 특정 영역에 집중적인 작업을 수행하거나, 할당할 객체를 분석하여 처음부터 힙에 할당하지 않고 지역 변수 저장 영역인 스택에 할당하여 가비지 컬렉터의 추적 대상에서 제외시키는 방법들이 연구되고 있다.

2.1. 나이 기반 분할(Age-based Partitioning)

나이 기반(age-based) 메모리 분할 기법은 '객체의 생존 비율이 그것의 나이와 깊게 관련된다'는 가설을 근거로 하여 힙 메모리를 Young 세대와 Old 세대로 분리하여 사용하는 방법이다. 즉, 최근 생성된 객체가 빨리 소멸될 가능성이 크기 때문에, 이들을 다른 객체들과 분리된 메모리 영역인 Young 세대에 할당하여, 그 영역에 더 많은 가비지 컬렉션 작업을 수행함으로써, 프로그램의 수행 성능 향상을 기대할 수 있다. 대부분의 세대별(generational) 가비지 컬렉터들이 각 세대를 위한 영역의 크기를 고정시킨 후, 위와 같은 방법을 사용한다[2]. 반면에, Appel[3]은 메모리 사용률을 최대화하기 위해 Young 세대와 Old 세대들간의 메모리 경계를 동적으로 변경하는 방법을 사용하였다. 최근 몇몇 연구들에서 최근 생성된 객체들이 빨리 소멸되지 않을 수 있는 것으로 밝혀졌으며, 따라서, 장수형(long-live) 객체들을 Young 세대에서 Old 세대로 복사(tenuring)하는데 드는 오버헤드를 피하기 위하여 곧바로 Old 세대에 할당하는 방법이 사용되기도 한다(pretenuring)[4].

2.2. 스택 기반 분할(Stack-based Partitioning)

스택 기반(stack-based) 분할 기법들은 그 객체들이 할당되는 스택 프레임(stack frame)들을 기준으로 객체들을 분할하는 방법이다. 이 기법은 '각 개체들의 수명이 자신이 속해있는 스택 프레임이 Pop되는 시간과 관련된다'는 가정을 토대로 하고 있다. 스택 기반 분할 기법으로는 정적인(static) 정보를 사용하는 것과 동적인 검사(dynamic check)를 통한 방법이 있다.

정적인 정보를 이용하는 기법은 '벗어남 분석(escape analysis)'을 통하여 객체들을 스택에 할당하는 방법이다[5]. 각 영역들은 정적으로 미리 정의된 프로그램 지점에서 스택 원리로 할당되고 회수된다. 각각의 객체들은 자신에 적합한 영역에 할당된다. 이후, 각각의 객체들은 개별적으로 회수되지 않으며, 자신이 속한 영역 내의 모든 객체들이 더 이상 사용되지 않는 것으로 판명되었을 때, 그 영역을 회수함으로써 소멸된다[6].

또 다른 스택 기반 분할 기법은 동적인 검사(dynamic check)를 통한 방법이다. Contaminated 가비지 컬렉션은 도달 가능한, 살아 있는 객체들의 최하위 스택 프레임을 정보를 저장하고 있다가 그 스택 프레임이 Pop 될 때에만 객체를 회수한다[7].

Qian과 Hendren의 할당기(allocator)[8]는 자바와 같은 언어에서 동작하며, 프로그래머에게 투명성을 제공한다. 이 기법은 영역 기반 관리 기법을 근간으로 하여, 페이지 단위로 구성된 힙을 사용하는 JVM에서 실행시간(runtime) 분석을 통해 특정 기준을 만족하는 객체들을 스택에 할당한다. 실행시간 분석은 벗어남 분석을 이용한다. 즉, 처음 할당된 후에 해당 영역을 벗어나지 않는 것으로 판명된 객체들을 지역 변수 영역에 할당한 후, 이와 연관된 스택 프레임들을 그 객체와 함께 할당하고 회수한다. 각각의 스택 프레임과 영역을 연결시키고, 객체가 스택 프레임을 벗어나는지를 기록하여, 만약 벗어난다면 그 영역을 전역 영역에 합병시켜 conventional GC로 처리하고, 그렇지 않은 경우에는 스택 프레임이 Pop 될 때 객체를 회수한다. 이러한 방법은 컬렉션의 빈도를 감소시키기 때문에, 가비지 컬렉터의 부담을 줄이는 효과를 가져온다.

스택 기반 기법은 함수형 프로그램 언어에서 잘 동작하지만, 프로그래머가 손수 튜닝해야 하는 단점을 가지고 있다. 이는 자동화된 메모리가 가져다 주는 이점인 프로그래머의 메모리에 관련된 작업으로부터의 주의 경감이라는 본래의 취지를 해치기도 하지만, 스택 원리는 크기가 큰 객체 지향 프로그램들에는 적합하지 않다.

2.3. 영역 기반 메모리 관리 기법

영역 기반(region-based) 메모리 관리 기법[11]은 관련된 객체들을 동일한 영역 내에 그룹 지어서 관리하는 기법으로, 각각의 객체들이 프로그램에서 지

정된 영역에 할당되며, 영역내의 모든 객체들이 해제된 영역을 시스템에 반환함으로써 메모리를 회수한다. 이러한 영역 기반 메모리 관리는 프로그램의 명확성 및 데이터의 지역성(locality)을 향상시키며, 객체 할당 비용을 한정된 범위로 쉽게 제한 할 수 있기 때문에 프로그램의 수행 성능을 보다 쉽게 예측할 수 있다. 하지만, 프로그래머가 영역에 할당된 객체들의 수명을 예측하기 어려운 경우에는 이 기법을 적용하기 힘들다.

2.4. 객체의 특성(Object behavior)

Blackburn[9]은 객체들의 수명과 소멸시간에 의거하여 단명형(short-lived), 장수형(long-lived), 불사형(immortal) 객체 등으로 분류하였다.

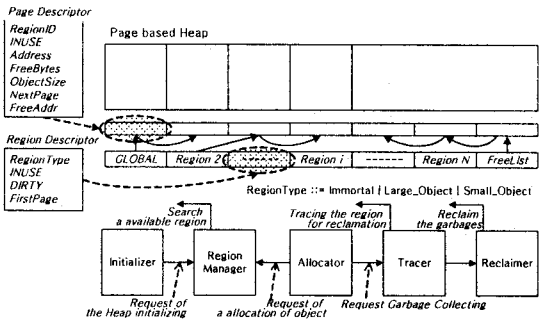
M. Hirzel[10]은 이를 확장하여 프로그램이 종료될 때 소멸되는 객체를 불사형(truly immortal), 객체의 소멸에서 프로그램 종료까지의 시간이 객체의 수명보다 짧은 객체를 준불사형(quasi immortal), 수명이 Ta × high_watermark (Ta = 0.2, high_watermark : 도달 가능한 힙 객체들의 최대 바이트 개수) 보다 짧은 객체를 단명형(short-lived), 이외의 객체들을 장수형(long-lived) 등으로 구분하였다.

결과적으로, 프로그램 스택을 통해서만 도달 가능한 객체들은 짧은 수명(short-lived)을 갖으며, 전역변수로부터 도달 가능한 객체들은 대체로 준 불사형이거나 불사형 객체였으며, 포인터들을 통해 직접 또는 추이적(transitively)으로 연결된 객체들은 대체로 동일한 시간에 소멸된다. 따라서, 원칙적으로 추적형(tracing) GC는 준 불사형(quasi immortal)이나 불사형(immortal) 객체들은 탐색하지 않으며, 주로 짧은 수명을 갖는(short-lived) 객체들에 탐색을 집중적으로 수행하며, 장수형(long-lived) 객체들에는 보다 적은 빈도로 컬렉션 작업을 수행한다.

3. 제안하는 가비지 컬렉션 기법

본 논문에서 제안하는 자바 가상머신(Java Virtual Machine, JVM)에서 동작하는 가비지 컬렉터는 [그림 1]과 같이 힙 메모리를 페이지 기반으로 구성하며, 메모리 초기자(initializer), 영역 관리자(region manager), 할당기(allocator), 추적기(tracer), 회수기(reclaimer) 등으로 구성된다.

본 논문에서는 효과적인 가비지 컬렉션을 위해서 객체의 크기, 객체의 소멸시간, 객체간의 연결성 등을 고려하였다. 포인터들을 통해 직접 또는 간접적(추이적)으로 연결된 객체들은 대체로 동일한 시간에 소멸된다는 객체간의 연결성을 이용하여, 이러한 객체들을 최대한 동일한 메모리 영역에 위치시킴으로써, 차후에 해당 영역의 모든 객체들이 더 이상 사용되지 않을 것으로 판명되었을 때 영역 전체를 한번에 회수하는 정책을 사용한다. 본 논문에서 제안하는 가비지 컬렉터를 구성하고 있는 각각의 모듈의 기능은 다음과 같다.



[그림 1] 제안하는 가비지 컬렉터와 메모리 구조

먼저, 메모리 초기자(initializer)는 JVM이 시작될 때, 최초 메모리를 초기화하는 부분으로 시스템으로 JVM이 사용하게 될 힙 메모리를 할당받아, 이를 시스템에 정의된 페이지 크기를 기준으로 자유 연결 리스트에 저장한다. 또한, 차후의 할당 요청을 처리할 때 가용한 메모리 블록을 빠르게 찾을 수 있도록 객체의 크기를 매핑 할 수 있는 크기 테이블을 초기화한 후, 영역 관리자를 호출하여 영역에 관한 정보를 초기화한다.

메모리 할당기(allocator)는 객체를 크기와 소멸시간에 따라 구분하여 미리 정의된 메모리 영역에 할당한다. 먼저, JVM을 구동하기 위해 내부적으로 사용되는 객체들은 객체의 수명이 가상머신과 동일하거나, 가상머신을 구현할 때 이러한 객체들에 할당된 메모리는 명시적으로 해제시킬 수도 있으므로, 분할된 메모리 영역, 즉, Immortal 메모리 영역에 할당하여 가비지 컬렉터의 추적 대상에서 제외시킨다. 다음으로, 페이지 기반의 힙에서 크기가 작은 객체들은 심각한 내부 단편화(internal fragmentation)를 발생시킬 수 있기 때문에, 객체 크기 테이블을 이용하여 정해진 크기 이하의 객체들은 정의된 크기별로 동일한 메모리 블록 내에 할당하는 정책을 취한다. 또한, 사전에 정의된 크기를 초과하는 객체들은 객체에 대한 헤더 정보만을 포함시켜 단일 메모리 블

록으로 할당한다.

영역 관리자(region manager)는 할당기와 회수기에 의해서 호출되며, Immortal 객체, Small 객체, Large 객체를 위한 영역, 가용 메모리 풀 등으로 나누어 관리한다. 할당기의 요청을 받으면 이미 할당된 영역에서 가용한 메모리의 주소를 반환하거나, 가용한 메모리 풀에서 새로운 메모리 블록을 찾아 해당 블록의 주소를 반환한다. 더 이상 가용한 메모리 블록이 없으면 '할당 실패'를 반환하여 할당기로 하여금 컬렉터를 호출하도록 한다.

추적기(tracer)는 삼색 추상화 기법[12]을 이용하여 메모리에 존재하는 객체들을 탐색하여 가비지를 찾아낸다. 삼색 추상화(Tricolor Marking)란 자신이 활성 상태이면서 직접 참조하는 객체까지 탐색된 객체는 Black, 자신은 활성 상태임이 검증되었으나 자신이 직접 참조하는 객체들은 탐색되지 않은 객체는 grey, 미검증 상태의 객체를 white 상태로 구분하여, 부분 검증 상태(grey)의 객체가 더 이상 존재하지 않을 때까지 추적 작업을 수행하는 기법으로, 탐색 작업이 종료된 후에도 미검증(white)상태로 남아 있는 객체들이 사용하는 메모리 영역을 회수하여 가용 메모리 풀로 반환하게 된다. 따라서, 추적기는 먼저 루트 셋(root set)을 결정한 후, 이러한 루트 셋들로부터 도달 가능 여부를 검사하여 위에서 언급한 기법을 이용하여 가비지들을 찾아낸다.

회수기는 추적기가 가비지로 판별한 객체들이 차지하고 있는 메모리 영역을 회수하여 가용 메모리 풀에 반환하다.

4. 결론

본 논문에서는 효과적인 가비지 컬렉션을 지원하기 위해 다양한 객체의 특성을 이용한 메모리를 분할 방법을 살펴보았다. 객체의 크기를 이용하여 내부 단편화를 최소화하고, 객체의 소멸시간을 이용하여 가상머신과 동일하거나 비슷한 시간에 소멸되는 객체들을 가비지 컬렉션의 대상에서 아예 제외시킴으로써 가비지 컬렉션에 기인한 프로그램의 성능 저하를 최대한 줄일 수 있으며, 또한 객체간의 연결성을 이용하여 서로 연결된 객체들을 최대한 동일한 메모리 영역에 위치시킴으로써 데이터의 지역성을 향상시키고 영역 기반 메모리 관리 기법을 적용시킬 수 있도록 하고 있다.

향후 과제로는 분할된 영역들간의 종속성을 분석하여 보다 효과적으로 각각의 영역을 회수할 수 있는 방안과 최대의 효율을 얻을 수 있는 영역을 선택하여 회수하기 위한 방안에 대한 연구가 필요하다.

참고문헌

[0)] R. Jones and R. Lins. "Garbage Collection : Algorithms for Automatic Dynamic Memory Management." John Wiley and Sons, 1996

[0)] David Ungar. "Generation scavenging : A non-disruptive high performance storage reclamation algorithm." In Practical Software Development Environments, 1984

[0)] M. Arnold, S. Fink, D. Grove, M. Hind, and P. Sweeney. "Adaptive optimization in the Japapeno JVM." OOPSLA, 2000

[0)] Perry Cheng, Robert Harper, and Peter Lee. "Generational stack collection and profile-driven pretenuring." PLDI, 1998

[0)] Young Gil Park and Benjamin Goldberg. "Escape analysis on lists." PLDI, 1992

[0)] M. Tofte and J. Talpin. "Region-based memory management." Information and Computation, 1997

[0)] D. Cannarozzi, M. Plezbert, and R. Cytron. "Contaminated garbage collection." PLDI, 2000

[0)] F. Qian and L. Hendren. "An adaptive, region-based allocator for Java." ISMM '02, 2002

[0)] S. Blackburn, S. Singhai, M. Hertz, K. S. McKinley, and J. E. B. Moss. "Pretenuring for Java." OOPSLA, 2001

[0)] M. Hirzel, J. Henkel, A. diwan, M. Hind. "Understanding the Connectivity of Heap Objects." ISMM '02, 2002

[11] D. Gay and A. Aiken. "Memory management with Explicit Regions." 1998

[12] E. W. Dijkstra, L. Lamport, C. S. Scholten and E. F. M. Steffens. "On-the-fly garbage collection : An exercise in cooperation." 1978