임베디드 환경에서 KVM의 성능 향상을 위한 가비지 컬랙션의 설계 및 구현

손필창[0], 강희성, 이원용, 이철훈
충남대학교 컴퓨터공학과
(pcson[0], hskang, nissikr, clee)@cnu.ac.kr

A Design and Implementation of The Garbage Collection for the Performance Improvement of KVM for Embeded Environments.

Pil-Chang Son[0], Hui-Sung Kang, Won-Yong Lee and Cheol-Hoon Lee
Dept. of Computer Engineering, Chungnam National Univ.

요 약

급속도로 IT 산업이 발전하면서, 리소스가 제한된 소형 기기들의 사용이 비약적으로 증가하는 추세이다. 자바는 플랫폼 독립성(Platform Independency), 보안성(Security), 이동성(Mobility) 등의 장점을 가지고 있기 때문에 성능을 극대화하고 안정된 서비스를 제공해야 하는 소형기기들에게 핵심 소프트웨어 플랫폼으로 많이 사용되고 있다. 임베디드 장치나 모바일 시스템과 같은 제한된 리소스를 사용하는 기기들은 자바 어플리케이션 수행을 위해 자바의 소프트웨어 플랫폼중의 하나인 K 가상 머신(K Virtual Machine: KVM)을 탑재하여 사용한다. 본 논문에서는 KVM 가비지 컬랙션의 지연 시간(Pause-Time)과 Tracing 의 수행 빈도수를 줄여 임베디드 환경에서 성능 향상을 위한 가비지 컬랙션을 설계하고 구현한 내용을 기술한다.

1. 서 론

최근 실시간 시스템(Real-time System)을 사용하는 소형 기기들이 많이 등장하고 있다. 이와 같은 시스템에서는 응용 태스크의 빠른 응답시간이나 높은 수준의 예측가능성이 보장되어야 한다.

자바는 플랫폼 독립성, 보안성, 이동성 등의 장점을 가지고 있기 때문에 성능을 극대화하고 안정된 서비스를 제공해야 하는 소형기기들에게 핵심 소프트웨어 플랫폼으로 많이 사용되고 있다. 그러나 인터프리터 방식의 느린 수행 엔진, 이식성을 고려한 하드웨어 직접접근 제한, 시간 제약성의 고려가 없는 병행 수행 모델, 동적 메모리 관리 등의 실시간성을 위협하는 문제점 때문에 실시간 시스템에 적용하기가 쉽지 않다.

자바의 장점은 소형 기기에 적용 시 개발 비용 감소 등 여러 가지 이득이 예상되기 때문에, 앞에서 언급한 문제점들을 극복하여 소형 기기에 탑재 한다면 실시간 시스템에서도 적용 가능한 플랫폼으로 사용될 수 있을 것이다.

본 논문에서는 위의 문제점들 중 동적 메모리 관리 측면에서 프로그램 수행 중간에 자바 가상 머신의 가비지 컬랙션 지연 시간 및 잦은 Tracing 수행에 의한 응용 태스크의 Pause-Time을 줄여 임베디드 환경에서 빠른 응답시간을 갖는 가비지 컬랙션 기법에 대해 설계 및 구현한 내용을 기술한다.

본 논문의 2 장 관련연구에서는 RC(Reference-Counting) 컬랙터 기법과 GC(Generational-Copying) 컬랙터 기법의 소개를, 3 장에서는 임베디드 환경에서 성능 향상을 위한 가비지 컬랙션 기법의 구현을, 4 장에서는 테스트 환경 및 결과를, 5 장에서는 결론 및 향후 연구 과제를 기술한다.

2. 관련 연구

2.1 RC (Refernce-Counting)

RC(Reference Counting)는 가비지 컬랙션 기법과 함께 대표적인 동적 메모리 관리 기법 중 하나이다[1]. 각각의 Object 는 처음 할당 되었을 때 0 으로 초기화 되는 Reference Count Field 를 가진다. 그 Count 의 값은 다른 Object 로부터 참조가 되면 1 증가 하고, 참조하던 Object 가 제거되면 1 감소 한다. Count 값이 계속 감소 하게 되어 0 이 된 Object 는 가비지가 되어 RC 컬랙터에 의해 시스템으로 반환되고, Count 값이 계속 증가하여 미리 정해놓은 Maximum Value 에 도달한 Object 는 Popular Object 로 간주되어 더 이상 1 감소 동작을 하지 않게 된다[3].

이와 같은 RC 컬랙터는 프로그램 수행 중간에 Object 의 Reference Count Field 값의 증가, 감소 및 0 이나 Maximum Value 와의 비교 동작만을 하기 때문에 Root Object 로부터 Tracing 을 하는 가비지 컬랙션 기법에 비교해서 지연되는 시간이 매우 짧다는

* 본 연구는 정보통신부의 선도기반기술개발사업의 지원으로 수행되었습니다.

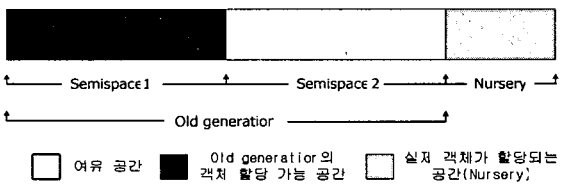
장점이 있다. 이러한 RC 컬렉터의 특성으로 RC 컬렉터가 Tracing 을 해야 하는 가비지 컬렉션 기법보다 실시간 환경에서 더 적합하다는 것을 알 수 있다[1].

그러나 RC 컬렉터는 Cycle 구조들을 찾아낼 수 없다는 단점이 있다. Cycle 구조의 한 예로 부모 Object 가 자식 Object 의 reference 를 가지고 있으면서, 자식 Object 도 부모 Object 의 reference 를 가지고 있는 경우를 들 수 있다. 이러한 경우 Reference Count Field 는 절대 0 이 될 수 없기 때문에 다른 외부 Object 로부터 접근이 불가능 할 지라도 가비지가 되지 않아 시스템으로 반환이 이루어 지지 않는다[3].

2.2 GC (Generational-Copyng)

KVM 에는 마크-회수 방법을 이용하여 가비지 컬렉션이 구현되어 있다. 마크-회수 방법은 참조되지 않는 오브젝트의 Mark bit 을 셋팅한 후 이를 수거하여 프리 리스트에 추가하게 된다. 이 방법은 간단하고 오버헤드가 적은 장점이 있으나, 시간이 지남에 따라 힙에 단편화 현상을 발생시켜 메모리의 효율성을 점점 감소시키는 단점이 있다[2].

힙 메모리 영역의 단편화와 가비지 컬렉션의 수행 횟수를 고려하여 좀 더 효율적인 방법들이 많이 나오고 있는데 그 대표적인 기법으로 Generational-Copying 가비지 컬렉션 기법을 들 수 있다.



[그림 1] Generational-Copying 가비지 컬렉션

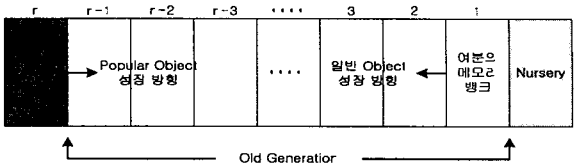
[그림 1]는 Generational-Copying 가비지 컬렉션을 위한 힙 메모리 영역의 구조를 나타내고 있다. 이 가비지 컬렉션은 힙 영역을 New Generation 영역(Nursery)과 Old Gerneration 영역으로 나누는 기법인 Generational 컬렉터에 Old Generation 부분만 Copying 컬렉터를 적용시킨 기법이다. 메모리 단편화 현상과 수행의 효율성을 동시에 높일 수 있는 효과를 얻을 수 있지만 Copying 단계를 수행하기 위해 전체 Old generation 영역의 1/2 만 객체를 할당할 수 밖에 없다는 단점과 풀 컬렉션을 할 경우 지연시간이 오래 걸린다는 단점이 있다[4].

3. 임베디드 환경에서 성능 향상을 위한 가비지 컬렉션 설계 및 구현

본 논문에서는 앞 장에서 살펴본 RC(Reference-Counting)기법과 GC(Generational-Copying)가비지 컬렉션 기법을 이용하여 임베디드 환경에서 빠른 응답성을 가진 가비지 컬렉션 기법을 설계 및 구현하였다.

3.1 힙 메모리 영역 설계

본 논문에서 제시하는 RC(Reference-Counting)기법과 GC(Generational-Copying)가비지 컬렉션을 혼합한 가비지 컬렉션을 적용하기 위해선 KVM 의 힙 메모리 영역과는 다른 구조가 필요하게 된다.



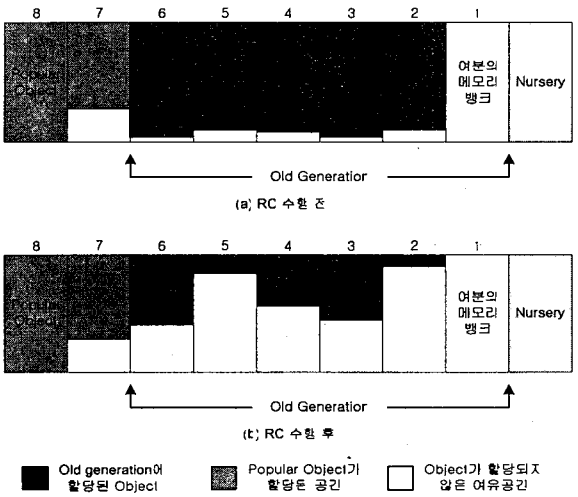
[그림 2] 힙 메모리 영역의 구조

[그림 2]는 본 논문에서 제시하는 힙 메모리 영역의 구조를 나타내고 있다. Object 의 할당은 Nursery 영역에서만 이루어지고, Nursery 영역에서 살아남은 Object 들은 Old Generation 영역에 저장 된다. Old Generation 영역은 Nursery 영역의 크기로 여러 개의 메모리 뱅크로 나누어 지게 되고, 각각의 뱅크는 오른쪽부터 차례로 고유번호를 가지고 있다. 또한, Old Generation 영역의 메모리 뱅크들 중 하나는 여분으로 두어 GC(Generational-Copying)의 Copying 단계에서 사용하게 된다. 처음 GC 가 이루어지기 전에는 1 번 메모리 뱅크를 여분의 메모리 뱅크로 설정해서 사용한다. 두 번째 GC 부터 여분의 메모리 뱅크 설정 과정은 3.3 절에서 자세히 설명한다.

일반 Object 는 오른쪽부터 쌓이게 되는데, Object 들 중 여러 다른 Object 로부터 reference 를 받는 Object 즉, Popular Object 는 왼쪽부터 쌓이게 된다. Popular Object 영역은 Popular Object 가 발생하기 전에는 왼쪽 마지막 뱅크를 시스템에서 예약 함으로서 일반 Object 들이 할당 되지 않도록 한다.

3.2 RC 단계(Reference-Counting Phase)

[그림 3]은 힙이 하나의 Nursery와 8 개의 메모리 뱅크로 이루어졌을 때 RC 수행 전과 수행 후의 힙의 모습을 나타낸 것이다.



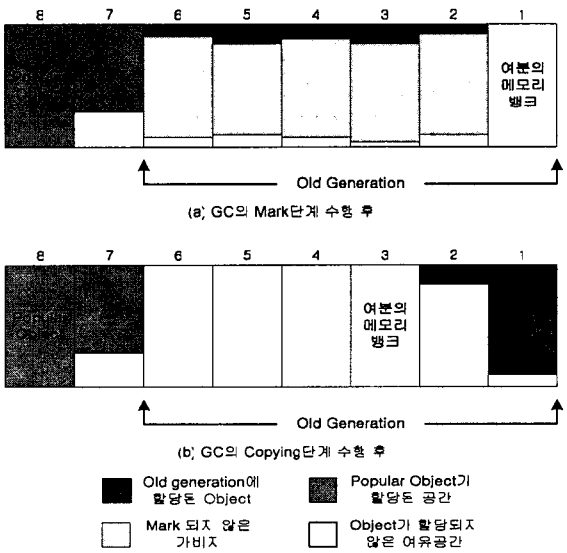
[그림 3] RC 단계(Reference Counting Phase)

RC는 Old Generation에 더 이상 Object를 할당할 공간이 없을 경우 시작하게 되며, Popular Object영역의 Object들은 가비지가 되지 않기 때문에 수행하지 않는다. Popular Object는 프로그램 수행 중 아주 드물게 발생하기 때문에 힙 메모리 영역을 매우 적게 차지 한다.

RC 단계는 Popular Object영역의 존재와 Compact 되지 않은 힙 영역 때문에 자주 수행되기는 하지만 수행시간이 기존의 tracing 가비지 컬랙션에 비해 적게 걸리기 때문에 프로그램 지연 시간을 크게 단축 시킬 수 있다.

3.3 GC 단계(Generational-Copying Phase)

RC 단계에선 Cycle 구조로 이루어진 Object들은 찾아낼 수 없기 때문에 RC를 계속 수행하게 되면 Old Generation에 Object할당 가능 공간이 점점 줄어들게 되어 RC를 수행해도 Object를 할당할 공간이 생기지 않는 경우가 발생하게 된다. 이럴 경우 여분의 메모리 뱅크를 이용하여 GC를 수행한다.



[그림 4] GC 단계(Reference Counting Phase)

[그림 4]는 GC의 Mark단계와 Copying단계 수행 후의 힙의 모습을 나타내고 있다. Copying 기법은 Mark-Copy을 사용하여 메모리 효율성도 높였다[4]. GC 수행 후 Object가 할당 되어있는 뱅크 중 번호가 가장 큰 뱅크 바로 옆 또는 만약 그 번호가 6 번이라면 1 번 뱅크를 여분의 메모리 뱅크로 설정한다.

GC는 RC의 Cycle Object 구조를 회수 하지 못한다는 단점과 메모리 Compaction을 하지 못한다는 단점을 해결하기 위해 본 논문에서 사용한 가비지 컬랙션 기법이다. 이 기법은 많은 양의 Object를 Copying을 하기 때문에 프로그램 지연시간을 길게 하는 단점이 있으나, GC를 수행하는 시기의 힙 메모리는 대부분이 RC에서 찾지 못한 Cycle Object 구조이기 때문에 Copying 해야 하는 Object수가 많지 않아 Copying 단계에 의한 지연시간이 많이 걸리지 않는다. 또한 대부분의 가비지 컬랙션은 RC에 의해 수행되기 때문에 GC만을 사용했을 경우에 비해 Tracing 수행 횟수를 크게 줄일 수 있다.

4. 테스트 환경 및 결과

본 논문의 테스트 환경은 레드햇(Red Hat) 리눅스 9.0 운영체제 위에 자바 컴파일러로 J2SDK 1.4.2_08를 사용하였고, 자바 가상 머신으로는 CLDC 1.0.4 를 사용하여 KVM의 가비지 컬랙션 부분을 구현하였다.

[표 1] 프로그램 테스트 결과

프로그램명	Total Number of Allocated Object (num)	Object Size Average (KB)	Execution Time (sec)	MS Pause Time (sec)	RC-GC Pause Time (sec)
JBrowser	16160	56	338.37	21.85	13.11
PhotoAlbum	3864	83	66.71	4.30	2.07
Scheduler	10042	66	253.09	5.77	2.31
JpegView	10199	44	417.35	50.03	25.02

※ MS : Mark-Sweep / RC-GC : Reference-Counting & Generational-Copying

[표 1]은 전체 1MB 크기의 힙을 100KB의 메모리 뱅크 10 개로 나누어 KVM의 기본 가비지 컬랙션인 마크-회수 기법과 본 논문에서 제시한 가비지 컬랙션 기법의 지연시간을 4 개의 자바 테스트용 프로그램을 이용하여 비교하였다. 그 결과 본 논문에서 제시한 가비지 컬랙션 기법이 마크-회수 기법보다 메모리 관리로 인한 지연시간 40%~60%정도 더 짧기 때문에 본 논문에서 제시한 가비지 컬랙션 기법이 응용 태스크의 빠른 응답성을 가진다는 것을 확인 할 수 있었다.

5. 결론 및 향후 연구 과제

본 논문에서는 임베디드 환경에서 KVM의 가비지 컬랙션의 성능향상을 위해 RC(Reference-Copying) 기법과 GC(Generational-Copying) 가비지 컬랙션 기법을 이용해서 태스크 프로그램의 응답성을 좀 더 빠르게 하는 가비지 컬랙션 기법을 설계 및 구현하였다.

본 논문에서 제시한 기법은 GC(Generational-Copying) 가비지 컬랙션 기법으로 인해 발생하는 지연시간 때문에 실시간성을 떨어뜨리고 있다. 효과적인 Cycle Object 구조 탐색 기법 및 프로그램 지연시간이 적은 Compaction 기법을 적용한다면 좀 더 빠른 응답성을 가진 가비지 컬랙션을 구현할 수 있을 것이다.

6. 참고문헌

[1] Bill Venners, " *Inside the Java2 Virtual Machine*" .

[2] Sun Microsystems, Inc., Java™ CLDC1.04 source code.

[3] L. P. Deutsch and D. G. Bobrow., " *An efficient incremental automatic garbage collector.*", *Communications of the* ACM, 19(9):522-526, September 1976.

[4] Narendran Sachindran and J. Eliot B. Moss., " *Mark-Copy: Fast copying GC with less space over head.*", In OOPSLA' 03 ACM SIGPLAN Notices, Anaheim, CA, November 2003.