KVM의 효율적인 힙 메모리 관리를 위한 MCCL 가비지 콜렉션의 설계 및 구현

최인범°, 이재규, 조문행, 남상훈†, 이철훈
충남대학교 컴퓨터공학과, †삼성 탈레스㈜ R&D 팀
(ibchoi°, jklee, mhcho, chlee)@ce.cnu.ac.kr, †sanghoon.nam@samsung.com

The Design and Implementation of MCCL(Mark-Compact-Compress-Lazy Allocate) Garbage Collection for effective Heap Memory Management in KVM

In-Bum Choi°, Jae-Kyu Lee, Moon-Haeng Cho, Sang-Hoon Nam†, Cheol-Hoon Lee
Dept. of Computer Engineering, Chungnam National Univ.
† Dept. of R&D Team, SAMSUNG THALES CO.,LTD

요 약

IT 산업이 발전하고, 제한된 리소스를 탑재한 소형 기기들의 사용이 증가함에 따라, 이러한 소형 기기들의 성능을 극대화하고 안정된 서비스를 제공하기 위한 다양한 핵심 소프트웨어 플랫폼들이 제안되고 있다. 자바는 플랫폼 독립성(Platform Independency), 보안성(Security), 네트워크 이동성(Network Mobility) 등의 장점을 가지고 있어, 많은 분야의 소형 기기들에서 핵심 소프트웨어 플랫폼으로 채택되고 있다. 임베디드 장치나 모바일 같은 제한된 리소스를 사용하는 기기들은 자바의 소프트웨어 플랫폼중의 하나인 K 가상 머신(K Virtual Machine: KVM)을 탑재하여 사용하고 있다. 본 논문에서는 제한된 리소스를 사용하는 소형 기기의 KVM 에서 좀 더 효율적으로 힙 메모리 관리를 하기 위한 MCCL(Mark-Compact-Compress-Lazy Allocate) 가비지 콜렉션 기법을 설계하고 구현한 내용을 설명한다.

1. 서론

자바의 플랫폼 독립성, 보안성, 네트워크 이동성, 실행 코드의 재사용성, 작은 실행 파일 크기, 동적 적응성, 이식성, 개발의 용이성 등과 같은 장점들은 소형 기기에 적용 시 개발 비용 감소 등 여러 가지 이득을 발생 시키기 때문에, 자바를 임베디드 장치나, 제한된 리소스를 사용하는 소형 기기에 적용하기 위한 연구들이 다방면에서 진행되어 왔다.

임베디드 장치나, 제한된 리소스를 사용하는 소형 기기에서는 장치 크기의 한계로 인해 메모리의 탑재가 제한될 수 밖에 없으며, 이러한 제한된 크기의 메모리를 효과적으로 사용하여 어플리케이션 프로그램의 수행속도를 높이기 위해, 장치에 탑재된 메모리를 효과적으로 관리하기 위한 여러 가지 기법들이 제안되고 있다.

본 논문은 여러가지 메모리 관리 기법들 중 힙 메모리 관리 측면에서 KVM의 힙 메모리를 효율적으로 관리하기 위한 MCCL (Mark-Compact-Compress-Lazy Allocate) 가비지 콜렉션 기법에 대하여 기술한다.

본 논문에서는 2 장에서 관련 연구로 표준 KVM 의 메모리 관리 체계를, 3 장에서 효율적인 KVM 힙 메모리 관리를 위한 MCCL (Mark-Compact-Compre ss-Lazy Allocate) 가비지 콜렉션 구현을, 4 장에서 테스트 환경 및 결과를, 5 장에서 결론 및 향후 연구 과제를 기술한다.

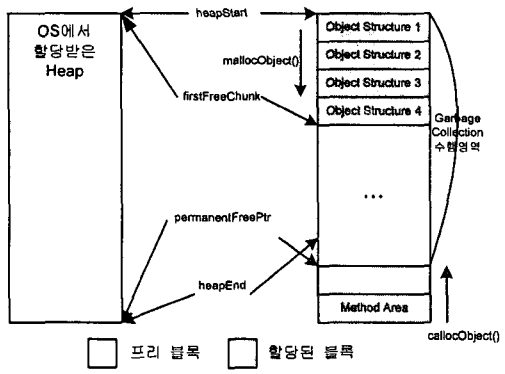
2. 관련 연구

2.1 KVM 메모리 관리 체계

[그림 1]은 KVM 의 힙 메모리 관리를 보여준다. 실행중인 자바 어플리케이션은 객체나 배열을 생성할 때 필요한 메모리 공간을 할당 받기 위해 mallocObject() 함수를 이용하는데, 이를 이용하여 힙 메모리를 할당할 경우 firstFreeChunk 위치부터 시작하는 프리 리스트(Free List)에서 퍼스트 핏(First Fit) 알고리즘에 따라 힙 메모리를 할당하고, 할당되지 않은 공간은 다시 프리 리스트에 추가한다. 프리 공간이 남아 있지 않다면 CHUNK 구조체를 프리 리스트에서 삭제한다.

callocObject() 함수는 컨스턴트 풀 (Constant Pool)이나 필드 또는 메소드 정보 등 변하지 않는 클래스 정보를 저장하기 위한 공간 할당 시 사용한다. 이 함수를 이용하여 힙 메모리를 할당 할 경우, 요구하는 메모리 크기가 현재 사용 가능한 메모리 공간 (permanentFreePtr - heapEnd)보다 작을 경우에는 메모리 공간을 할당하고, 그렇지 않을 경우에는 heapEnd 값을 재 설정한 후 메모리 공간을 할당한다.

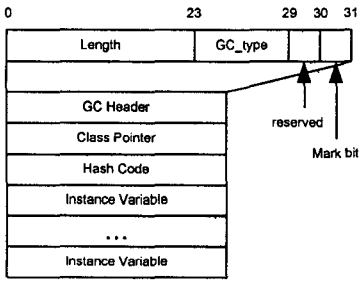
• 본 연구는 정보통신부의 선도기반기술개발사업의 지원으로 수행되었습니다.



[그림 1] KVM 메모리 관리

2.2 오브젝트 구조(Object Structure)

오브젝트 구조는 GC(Garbage Collection) 헤더를 포함하여 클래스에 대한 포인터, 모니터와 해쉬코드에 대한 포인터, 클래스 객체에 필요한 데이터를 저장하기 위한 공간을 포함한다.



[그림 2] Object Structure

[그림 2]에서 Length 는 GC 헤더 (1 byte)를 제외한 unsigned long(4 byte)을 하나의 크기로 가지는 길이를 나타내며, GC_type 은 가비지 콜렉션 타입을 나타낸다. 또한 Mark bit는 가비지 콜렉션 수행 시 오브젝트의 참조여부를 마크하기 위한 비트이다.

2.3 기본 가비지 콜렉션

KVM 에는 마크-회수 방법을 이용하여 가비지 콜렉션이 구현되어 있다. 새로운 객체나 배열을 위한 메모리 할당 시 더 이상 힙에 메모리를 할당하지 못할 경우 마크-회수 방법을 이용하여 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.

3. KVM 힙 메모리 관리를 위한 MCCL(Mark-Compact-Compress-Lazy Allocate) 가비지 콜렉션 구현

KVM 에서는 기본적으로 마크-회수 방법을 이용하여 가비지 콜렉션이 수행되는데, 새로운 객체나 배열을 위한 메모리 할당을 위해 반복적으로 수행하게 되면, 힙 메모리에 단편화 현상이 발생할 수 있다. 그 결과 전체적으로는 메모리를 할당 할 수 있는 공간적인 여유가 있으나, 연속된 할당 가능한 공간의 크기가 작아 더 이상 메모리를 할당 할 수 없는 현상이 발생하게 된다. 그래서 본 논문에서는 이러한 현상을 방지하고, KVM 의 힙 메모리 영역을 최대한 활용하기 위한 MCCL (Mark-Compact-Compress-Lazy Allocate) 가비지 콜렉션 기법을 구현하였다.

MCCL (Mark-Compact-Compress-Lazy Allocate) 가비지 콜렉션의 구현을 위해서는 MCC (Mark-Compact-Compress) 가비지 콜렉터와 Lazy Allocate 힙 메모리 정책이 필요하다.

3.1 MCC (Mark-Compact-Compress) 가비지 콜렉터

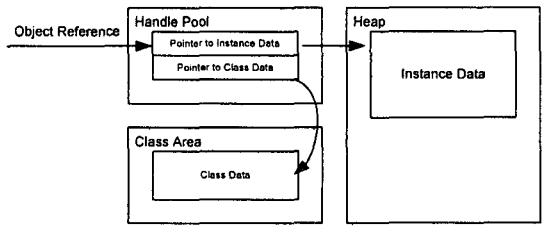
MCC (Mark-Compact-Compress) 가비지 콜렉터는 필요 없는 메모리 공간을 회수하고, 할당 가능한 힙 공간을 최대화 하기 위하여 2 단계의 정책을 사용하고 있다.

3.1.1 마크 단계 (Mark phase)

이 단계에서는 현재 사용되고 있는 객체들의 Mark bit 를 셋팅함과 동시에 객체들의 크기를 파악해 저장한다. 이를 이용하면 계산된 객체들의 총 크기를 통하여 전체 힙에서 할당 가능한 힙 공간의 영역 크기를 계산 할 수 있게 된다.

3.1.2 압축 단계 (Compact-Compress phase)

이 단계에서는 새로 할당 받아야 할 객체의 크기가 마크 단계에서 계산된 할당 가능한 힙 공간보다 작을 경우 Compact 만을 수행하여 새로운 객체를 할당한다. [그림 3]에서와 같이 Permanent 영역에 Handle Pool 구조체 영역을 추가하면 Compact 를 용이하게 할 수 있다.

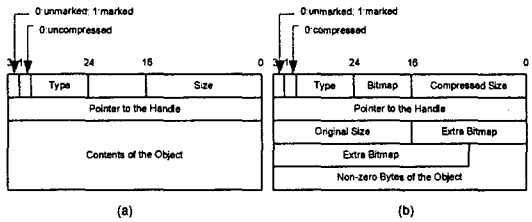


[그림 3] Handle Pool 을 사용한 객체 참조

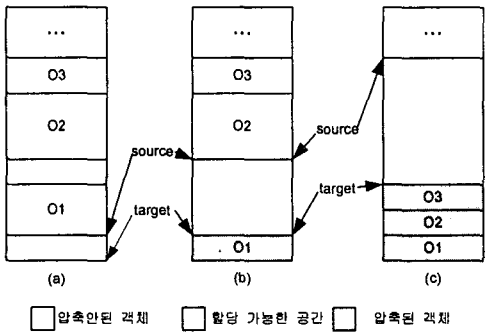
그러나 새로 할당 받아야 할 객체의 크기가 계산된 힙 공간의 크기보다 클 경우에는 Compress 를 수행하여 사용중인 객체들을 압축하여 힙 공간을 늘려 새로운 객체를 할당하게 된다.

Compress 알고리즘으로 " zero removal" 을 사용하여 객체 내의 zero byte 를 삭제, 압축하였다. Compress 알고리즘을 적용하기 위하여 [그림 4] (a)와 같이 일반적인 객체의 포맷과는 다른 Compress 알고리즘을 위한 새로운 객체 포맷 (b)를 정의하고 압축된 객체는 이 포맷에 맞춰 힙 메모리의 target 영역

부터 재 저장된다. [그림 5]에서와 같이 Compress 를 수행하게 되면 수행 전에 비하여 객체의 크기가 훨씬 작아진 것을 확인할 수 있다.



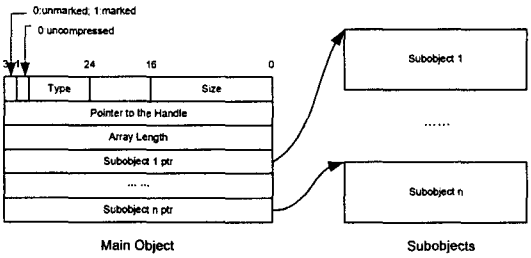
[그림 4] 객체 포맷



[그림 5] Compressing Heap

3.2 Lazy Allocate 힙 메모리 정책

동적으로 사용하기 위한 배열이나 객체 등을 위한 메모리 공간을 컴파일 시에 모두 최대 크기만큼 할당하게 되면 그 만큼 메모리 공간의 낭비를 초래하게 된다. 동적으로 사용하게 될 메모리 공간은 실행 시에 공간의 크기가 결정되므로 미리 메모리 공간을 할당 하지 않고 [그림 6]에서와 같이 전체 크기를 일정한 단위로 분할하여 필요한 만큼의 메모리 만을 사용하는 방법이 더 효율적이다.



[그림 6] 예제 : 동적 배열 분할

4. 테스트 환경 및 결과

본 논문의 테스트 환경은 Redhat LINUX release 9 운영체제 하에서 자바 컴파일러로 J2SDK_1.4.2_05 를 사용하였으며, 자바 가상머신은 SUN사의 CLDC1.0.4 를 기반으로 하여 가비지 콜렉션을 구현하였다. 또한 테스트용 프로그램은 웹을 통하여 총 6 개의 자바 어플리케이션 프로그램을 수집하였다. 테스트용 프로그램은 [표 1]과 같이 구성되어 있다.

[표 1] 테스트용 어플리케이션 프로그램 구성

프로그램명	Total Number of Allocated Object (num)	Object Size Average (Max, KB)	Execution Time (sec)	GC Time (sec)
Calculator	11250	26(1036)	65.94	2.19
JBrowser	16160	56(12012)	338.37	21.85
JpegView	10199	44(8972)	417.35	50.03
PhotoAlbum	3864	83(4260)	66.71	4.30
Scheduler	10042	66(1036)	253.09	5.77
Snake	1776	29(1036)	68.96	1.53

※ 시간은 KVM의 마크-회수 가비지 콜렉션 기준으로 측정

[표 2]의 테스트 결과에서 보여주듯이 마크-회수 가비지 콜렉션 보다는 본 논문에서 구현한 MCCL 가비지 콜렉션이 힙 메모리 관리 측면에서 좀 더 효율적임을 알 수 있다.

[표 2] 프로그램 수행에 필요한 평균 힙 크기

프로그램명	평균 힙 크기 (KB)			백분율 환산 (MC기준)		
	MS	MC	MCCL	MS	MC	MCCL
Calculator	55	40	34	138.5	100	85.0
JBrowser	260	226	164	115.0	100	72.6
JpegView	127	85	77	149.4	100	90.6
PhotoAlbum	96	55	50	174.5	100	90.9
Scheduler	56	37	32	151.4	100	86.5
Snake	72	42	35	171.4	100	83.3

※ MS : Mark-Sweep / MC : Mark-Compact

5. 결론 및 향후 연구과제

본 논문에서는 제한된 리소스를 가지는 소형기기의 핵심 소프트웨어 플랫폼이 된 자바환경들 중 KVM에서 제한된 크기의 메모리를 최대한 효율적으로 사용하기 위해 MCCL 가비지 콜렉션 기법을 설계 및 구현하였다.

KVM 은 소형기기 뿐만 아니라 실시간 운영체제 상에서도 동작이 가능하다. KVM 의 확장 측면에서 생각한다면 힙 메모리의 효율적인 사용 뿐만 아니라 실시간성까지 만족시키기 위한 좀 더 발전된 가비지 콜렉션 기법이 필요하다.

6. 참고문헌

[1] G. Chen, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, B. Mathiske, M. Wolczko; " *Heap Compression for Memory-Constrained Java Environment*" ; ACM SIGPLAN Conference on Object-oriented programming, systems, languages, and applications, 2003, pp.282-301

[2] Sun Microsystems, " *JSR-30 J2ME Connected, Limited Device Configuration*"

[3] Bill Venners, " *Inside the Java Virtual Machine*" ,1999

[4] Tim Lindholm, " *The Java™ Virtual Machine Specification Second Edition*" , 1999