자바 플랫폼에서 효율적인 쓰레기 수집기

이은화*, 윤성대**
*부경대학교 전산정보학과
**부경대학교 전자계산학과
e-mail:lehsgi@hanmail.net

An Efficient Garbage Collector on Java Platform

Eun-Hwa Lee*, Sung-Dae Youn**
*Dept. of Computer and Information, Pukyong National University
**Dept. of Computer Science, Pukyong National University

요 약

세대별 쓰레기 수집기의 알고리즘을 사용하는 자바 플랫폼에서 객체의 생명 주기가 짧은 응용프로그램과 객체의 생명 주기가 긴 응용프로그램에 각 각 힙의 크기를 조정하여 가비지 콜렉션 성능 측정과 동일한 힙의 크기일 때 young generation크기 조정을 하여 가비지 콜렉션의 회수와 실행시간의 성능을 향상시키도록 한다.

1. 서론

하드웨어 플랫폼에 독립적이고, 이식성을 보장하는 자바의 기술은 서버 급에서 정보가전기기에 확산되어 자바의 런타임 환경은 J2SE, J2ME, J2EE의 3가지로 나누어져있다.

자바는 특히 메모리 관리를 가비지 콜렉터로 자동으로 관리해 준다. 가비지 콜렉터는 프로그램에 의해 더 이상 참조되지 않는 객체들을 찾아서 그 객체들이 차지하고 있던 힙(heap) 공간을 재 사용할 수 있게 함으로써 프로그래머로 하여금 메모리에 대한 부담을 전혀 가지지 않으므로 소프트웨어 생산성을 향상시킬 수 있는 장점을 가지고 있다.[1][2]

자동적인 가비지 콜렉터의 종류는 참조 계수, 마크-회수, 복사, 세대별 방법이 있다.

본 논문에서는 세대별 방법을 선택하고 있는 자바 플랫폼에서의 가비지 콜렉션을 특징을 고찰해서 힙의 크기에 따른 성능 측정과 동일한 힙의 크기에서 Young generation의 크기 조정에 따른 성능 측정을 살펴서 효율적인 방법을 선택할 수 있는 방법을 제시한다.

본 논문의 구성은 2장은 본 논문의 관련 연구에 대해서 소개하고, 3장은 세대별 가비지 콜렉션에 대해서 기술했으며, 4장은 성능 측정을 하고, 마지막 5장에 결론을 맺는다.

2. 관련연구

자바는 프로그램의 메모리를 관리한다. 자바가상머신은 메모리를 free시키고, 메모리를 필요한 양만큼 할당시키고 하는 등의 일을 한다. 자바 가상머신은 불필요한 객체 수거를 가비지 콜렉터에 의해서 이루어진다. 자동적인 가비지 콜렉터의 종류는 참조 계수, 마크-회수, 복사, 세대별 방법이 있다.

첫 번째로 참조 계수(reference counting)방법은 객체 참조 계수를 비교하여서 메모리를 회수하는 방법으로 참조 계수 값이 0인 객체를 수거하는 방법이다. 구현하기는 간단하지만 참조가 생길 때마다 참조 계수를 변경해야 하는 카운팅 오버헤드가 발생하고, 메모리 단편화 문제가 발생한다[2][3]

두 번째로 마크-회수(mark-sweep)방법은 root로부터 시작하여 참조 트리를 순회하며 살아있는 객체

와 가비지에 대해서 마크하는 마크단계와 다음 단계로 마크되지 않은 객체를 반환하여 그 객체가 차지하고 있던 힙 영역을 프로그램으로 다시 사용하는 회수단계로 이루어진다. 알고리즘 포인터 조작이 필요 없고, 작은 메모리 공간을 차지하지만 메모리 단편화가 발생한다. Personal Java와 Kaffe VM, KVM에서 사용된다.[1]

세 번째로 복사(copying)방법은 메모리를 두 개로 나누어 한쪽에만 할당을 계속하고, 할당할 수 없을 때 나머지 한쪽에 살아있는 객체를 이동하고, 할당을 하는 방법이다. 메모리 단편화는 해결되지만 동적 메모리의 효율이 50% 이하로 떨어지는 단점이 있다.[1][2]

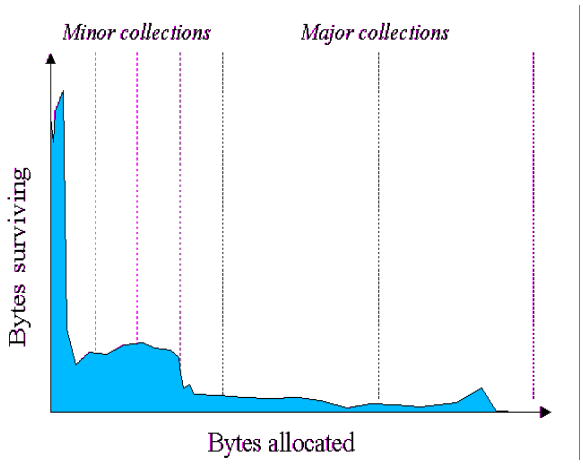
네 번째로 세대별(generation)방법은 자바 프로그램에서 만들어지는 대부분의 객체들은 매우 짧은 시간 동안만 참조되어진다는 것과 오랫동안 유효했던 객체는 계속적으로 유효할 것이라는 전제를 바탕으로 한다. 객체에 생성된 시기를 기준으로 나이라는 개념을 도입하여 young 영역과 old영역으로 구분하여 관리된다. 객체는 young영역내의 eden영역에 할당되는데, 더 이상 객체를 할당할 수 없는 경우 살아있는 객체는 복사 알고리즘에 의해 young 영역의 survivor 영역으로 복사된다. young 객체가 old 객체보다 많이 가비지 콜렉터 한다. Hotspot, CVM에 적용되고, 세대별 객체 저장 공간을 별도로 필요함으로 메모리 요구 사항이 높아진다.[2][3]

3. 세대별(generation) 가비지 콜렉션

가비지(garbage)의 대상은 heap 영역에 할당되어 있는 객체들 중에서 더 이상 참조되지 않는 것을 수거한다.

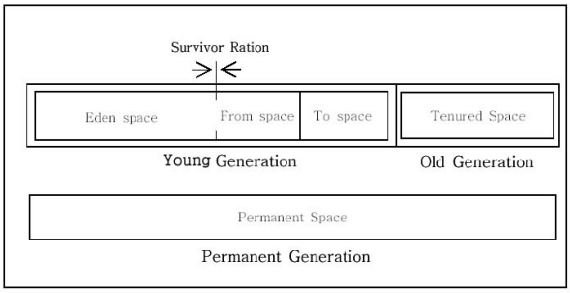
가비지 콜렉터의 시기는 해당 객체를 가리키는 참조변수가 더 이상 존재하지 않는다고 판단될 경우 가비지 콜렉터에 의해 finalize() 메소드가 호출된다. 자바에서는 시간 알고리즘에 의하여 수행되고, 메모리 사용에 대한 요구가 현재 남아있는 메모리의 크기보다 큰 경우이다.[4]

그림1에서 칠해진 영역은 객체들의 생명주기의 전형적인 분포이다. 왼쪽의 뾰족한 부분은 할당되어진 후에 간단하게 다시 선언되어진 객체들을 나타낸다 일부 객체들은 더 오랫동안 살아 있다. 그래서 분포는 오른쪽 밖으로 뻗어 있다. 물론 일반적으로 프로세스가 존재하는 동안에 살아서 초기화 된 객체들이 있다. 오른쪽으로 산등성이처럼 보이는 몇 가지 중간 계산 동안에 살아 있는 객체들이 존재한다. 일부 애플리케이션들은 매우 다른 관점의 분포들을 가진다. 그러나 놀랍게도 많은 수가 위의 일반적인 분포를 가진다. 객체의 대부분이 초기에 없어지므로 세대별 가비지 콜렉션이 만들어 져야 한다.[5]



<그림1> 객체 생명 주기 분포

세대별 쓰레기 수집기를 사용하는 JVM의 힙 영역은 그림2와 같다. 힙 영역은 세 부분으로 나누어지는데 JVM 클래스와 메소드 객체를 저장하는 Permanent 영역, 만들어진지 좀 지난 객체들을 저장하는 Old generation 영역, 모든 새로 생성된 객체를 저장하는 Young generation 영역으로 구성된다. Young generation 영역은 new에 의해서 새로 생성된 객체를 저장하는 Eden 영역과 Eden에 있던 객체가 Old 영역에 가기 전에 객체를 저장하는 Survivor 영역으로 구성된다. [7]



<그림2> 힙(heap) 영역 레이아웃

세대별 가비지 콜렉터에는 두 가지의 collection이 있다. minor collection과 major collection 이다.

minor collection은 young generation에서 Old generation으로 살아있는 객체들을 이동시키고, 불필

요한 객체들을 수거하는 것으로 pause time이 짧다. major collection은 young과 old generation이 가득 차서 메모리 할당을 할 수 없을 때 수행하는 것으로 pause time이 길다.[2][3]

4. 성능 측정

측정환경은 펜티엄Ⅲ,windows98, 자바가상머신으로는 JDK1.4.2를 사용했고, 응용프로그램으로는 10000개의 소수(prime)를 구하여 객체로 생성하는 프로그램과 벤치마크 프로그램인 java-olden 중 그래프의 최소 스패닝 트리(spanning tree)구하는 프로그램을 vertex를 500개를 인자로 주어 측정하고, 힙의 크기를 조정하여 측정결과를 가비지 콜렉션 투닝하는 프로그램을 통하여 디스플레이 하였다. 동일한 힙 크기 사용 시 young generation의 크기비율을 조정하여 결과측정을 하였다.

힙의 크기를 변화시켜서 가비지 콜렉션의 회수와 실행시간을 측정하였다. 표1은 객체의 생명주기가 짧은 객체를 10000개 생성하였을 때 나온 결과이다. 힙의 크기가 커질수록 가비지의 콜렉션의 회수는 줄었지만, 실행시간은 늦어짐을 볼 수 있었다.

<표1> 힙의 크기 변화에 따른 측정 결과(소수 구하기)

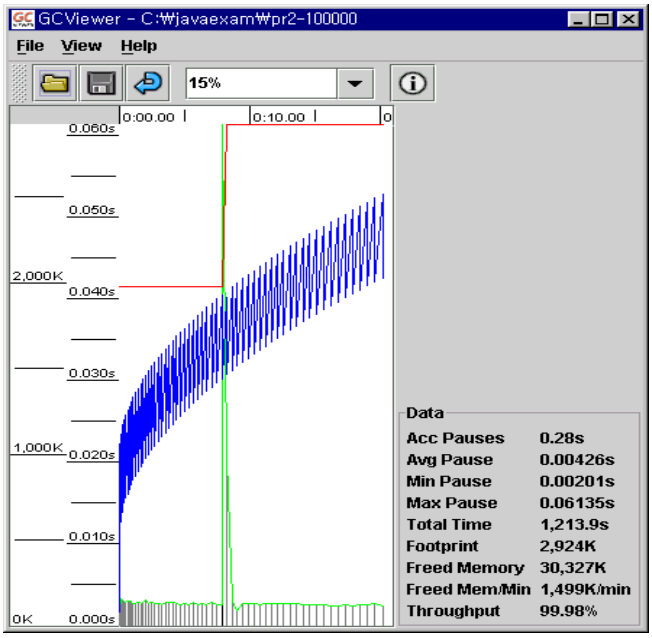
힙 크기	가비지 콜렉션 회수	실행시간(ms)
1M	5	8950
2M	5	9000
4M	5	9170
8M	5	9330
10M	4	9720
16M	2	9780
32M	1	9940
64M	0	12300

표2는 vertex 500개로 최소 스패닝(spanning)트리를 구하는 프로그램에서 힙의 크기 변화에 따른 측정 결과이다. 객체의 생명주기가 앞의 예제보다는 긴 객체 생성에 대한 예제이다. 힙의 크기가 클수록 가비지 콜렉션 회수가 감소했으며, 실행시간도 감소함을 볼 수 있다. 작은 크기의 힙에서는 살아있는 객체의 수가 증가함으로써 pause time을 많이 차지하는 major collection(괄호속 숫자)가 많이 발생함으로써 실행 시간이 증가하였다.

그림3은 객체 100000개를 생성하고, 힙의 크기를 2M 사용시 가비지 콜렉션 튜닝 프로그램으로 결과를 디스플레이한 결과이다. 1열은 힙의 소비량,2열은 garbage collection time, 3열은 응용프로그램 실행시간, 하단의 수직선은 minor collection을 나타내고, 중간에 가장 긴 수직선은 major collection을 나타내고, 리플은 사용되는 힙을 나타낸다. 수십 번의 minor collection 발생하였고, 한 번의 major collection이 한 번 발생한 이후로 사용 가능한 힙의 크기가 증가됨을 볼 수 있다.

<표2> 힙의 크기변화에 따른 측정 결과(최소 스패닝트리구하기)

힙 크기	가비지 콜렉션 회수 (major collection)	실행시간(ms)
1M	23(4)	2360
2M	20(3)	1650
4M	19(2)	1600
8M	19(1)	1370
10M	15(0)	880
16M	9(0)	820
32M	4(0)	800
64M	2(0)	770



<그림3> 가비지 콜렉션 튜닝

힙의 크기를 고정시키고,young generation 영역의 비율을 다르게 하여서 측정한 결과이다. 표3,표4에서 힙의 크기를 1M,2M,4M,8M로 고정하고, 각 young generation 영역을 힙에서의 차지하는 비율로 조정한 결과이다. 표3은 객체의 생명주기가 짧은 객체를 10000개 생성한 결과이다. 힙의 크기가 1M일 때에는 가비지 콜렉션의 회수는 young generation 영역 조정하지 않을 때와 결과는 동일하나, 실행시간이 더 걸렸다. 힙의 크기가 2-8M일 때 young

generation 크기가 클 때에는 가비지 콜렉션 회수를 줄이고, 실행시간이 많이 걸렸다. 전체적으로 young generation을 조정하지 않을 때 보다 훨씬 가비지 콜렉션 회수는 감소하였으나 실행 시간은 많이 걸렸다.

<표3> 동일 힙 크기에 young generation 비율 조정(소수구하기)

힙 크기	Young /heap	가비지 콜렉션 회수 (major collection)	실행시간(ms)
1M	1 / 2	5	9450
	1 / 3	5	9390
	1 / 4	5	9230
	1 / 5	5	9120
2M	1 / 2	3	9830
	1 / 3	5	9440
	1 / 4	5	9340
	1 / 5	5	9180
4M	1 / 2	1	12240
	1 / 3	2	10550
	1 / 4	3	10440
	1 / 5	4	10320
8M	1 / 2	0	15660
	1 / 3	1	12350
	1 / 4	1	11260
	1 / 5	2	10380

<표4> 동일 힙 크기에 young generation 비율 조정 (최소 스패닝 트리 구하기)

힙 크기	Young /heap	가비지 콜렉션 회수 (major collection)	실행시간(ms)
1M	1 / 2	7(3)	1210
	1 / 3	10(3)	1320
	1 / 4	12(4)	1600
	1 / 5	13(5)	1970
2M	1 / 2	7(3)	1590
	1 / 3	10(3)	1380
	1 / 4	11(3)	1370
	1 / 5	13(3)	1480
4M	1 / 2	4(2)	1320
	1 / 3	7(2)	1380
	1 / 4	9(2)	1480
	1 / 5	11(2)	1540
8M	1 / 2	3(1)	1040
	1 / 3	4(1)	1090
	1 / 4	6(1)	1160
	1 / 5	7(1)	1210

표4는 vertex 500개로 최소 스패닝(spanning)트리를 구하는 프로그램으로, 동일한 힙 크기일 때, young generation 크기가 클수록 가비지 콜렉션의 회수가 감소하고, 실행 시간도 감소함을 보였다. 전체적으로 young generation을 조정하지 않을 때 보다 훨씬 가비지 콜렉션 회수와 실행 시간면에서 성능이 향상되었다.

5. 결론

대부분의 객체가 초기에 소멸되므로 능률적인 가비지 콜렉션 방법으로 세대별 가비지 콜렉션을 사용하는 것이다. 특히 객체의 생명 주기가 짧은 응용프로그램에서는 힙의 크기가 커질수록 가비지의 콜렉션의 회수는 줄지만 실행시간이 증가되었다. 동일한 힙에서 young generation을 조정하지 않을 때, 보다 훨씬 가비지 콜렉션 회수는 감소하였으나 실행 시간은 많이 걸렸다.

메모리의 최대 사용량이 높고, 객체의 생명이 긴 응용프로그램에서는 힙의 크기에 따라 가비지의 콜렉션의 회수도 줄고, 실행시간도 줄었다. 동일한 힙에서 young generation을 조정하지 않을 때 보다 조정할 때가 훨씬 가비지 콜렉션 회수와 실행 시간 면에서 성능이 향상되었다.

향후 연구로는 자동화된 메모리 관리 하에 명시적으로 객체를 수거하여 가비지 콜렉션 회수와 실행 시간 면에서 성능 향상을 고찰하고자 한다.

참 고 문 헌

[1] Bill Venners, "Inside the JAVA2 Virtual Machine Second edition",pp.356-384,1999

[2] Richards Jones,"Garbage Colletion Algorithm for Automatic Dynamic Memory Management" pp.75-182,1999

[3] Sun Microsystems,The Hotspot Virtual MachineTehnical White Paper, Sun Microsystems,2001

[4] http://sookmyung.ac.kr/~u0011616/hw_s_0924.html

[5] Java HotSpot VM Options http://www.monosun.com/doc/gc.html

[6] http://www.javaperformancetuning.com/ java Performance tunning 2002

[7] http://www.borlandforum.com/