# 소규모 메모리 자바 프로그램을 위한 가비지 콜렉션

권혜은[O] 김상훈
세명대학교 일반대학원 전산정보학과[O], 세명대학교 소프트웨어학과
{smallpig[O], ksh}@plac.semyung.ac.kr

## Garbage collection for small memory java application

Hye-Eun Kwon[0] Sang-Hoon Kim
Dept. of Computer & Information, Semyung University[0], Dept. of Software, Semyung University

**요 약**

제한된 환경에서 자바 실행환경을 제공하는 KVM은 마크-회수 방식의 가비지 콜렉터를 사용하고 있다. 그러나 마크-회수 방식은 시간이 지남에 따라 심각한 메모리 단편화 문제로 충분한 메모리 공간이 있음에도 불구하고 프로그램의 실행이 중단되는 문제점을 가지고 있다. 본 논문에서는 단편화 문제의 개선을 위해 단일 링크드 리스트 형태로 구성되어 있던 프리 리스트를 이중 링크드 리스트로 구성하고 객체의 크기에 따라 할당 위치를 달리하는 방법을 제안한다. 이를 통하여 단편화 문제가 최소화되어 메모리 낭비를 줄일 수 있음을 실험을 통해 확인하였다.

## 1. 서론

J2ME (Java 2 Platform, Micro Edition)의 CLDC (Connected Limited Device Configuration)는 16비트 혹은 32비트 프로세서에 160KB ~ 512KB의 제한된 메모리를 가지고 네트워크에 연결될 수 있는 장치들을 위한 최소한의 자바 플랫폼을 정의한다.[1, 2] CLDC에서의 메모리 제한과 더불어 제한된 자바가상머신 명세서를 따르는 SUN의 자바가상머신이 KVM(K Virtual Machine)이다. KVM의 메모리는 JVM(Java Virtual Machine)과 마찬가지로 가비지 콜렉터에 의해 관리된다. 가비지 콜렉터는 더 이상 사용되지 않는 객체를 찾아서 그 공간이 실행 프로그램에 의해 재 사용될 수 있도록 한다. 이는 프로그래머가 메모리를 직접 관리해야 하는 부담을 줄여 프로그램 생산성 향상에 도움을 준다.[3]

본 논문에서는 제한된 메모리를 가지는 KVM에서 사용되는 가비지 콜렉터의 메모리 단편화 문제를 최소화하기 위한 방법으로 기존의 링크드 리스트를 이용한 단방향 할당 대신 이중 링크드 리스트를 이용하는 방법을 제안한다. 이는 객체의 크기에 따라 할당 시작 위치를 조절함으로써 단편화 문제가 개선되어 메모리 낭비를 줄일 수 있음이 실험을 통해 확인되었다.

본 논문의 구성은 2장에서는 기존 자바 환경의 가비지 콜렉션에 관해 살펴보고, 3장에서는 개선된 할당 방법에 관해 설명한다. 4장에서는 기존 알고리즘과 개선된 방법을 실험하여 성능을 비교하고, 5장은 결론 및 향후 연구과제로 이루어져 있다.

이 논문은 한국과학재단의 특정기초연구 (과제번호:1999-1-30300-3) 지원에 의한 것임.

## 2. 관련연구

프로그램의 루트 셋으로부터 도달 가능한 객체를 live 객체, 도달 불가능한 객체를 가비지라 한다. 가비지 콜렉터는 프로그램에서 더 이상 필요로 하지 않는 객체인 가비지가 차지한 공간을 회수하여 실행 프로그램에서 재 사용할 수 있도록 한다.[3]

### 2.1. KVM의 가비지 콜렉션

KVM에서 사용되는 마크-회수(mark-sweep) 방식은 두 단계로 구성된다. 첫 번째 마크 단계는 루트 셋으로부터 도달 가능한 모든 객체에 표시를 하고, 두 번째 회수 단계는 메모리 시작 번지부터 마지막 번지까지 live 객체의 표시를 지우면서 표시가 없는 가비지는 인접한 공간을 병합하면서 프리 리스트로 불리는 링크드 리스트에 연결한다. 프리 리스트의 공간은 실행 프로그램의 메모리 요청에 따라 새로운 객체를 위해 할당된다.

마크-회수 방식은 한번 할당된 객체의 위치를 이동시키지 않기 때문에 시간이 경과함에 따라 객체들이 분산되어 심각한 메모리 단편화 문제와 참조의 지역성 문제가 발생한다. 또한 콜렉션 발생 시점에 메모리를 두 번 탐

색해야 하므로, 메모리 크기에 비례하는 콜렉션 비용을 가진다.[3, 4]

2.2. JVM의 가비지 콜렉션

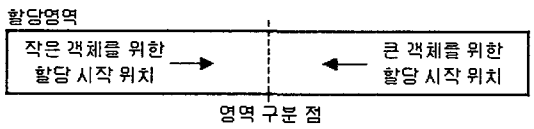
다양한 방식을 혼합적으로 사용하는 JVM은 객체의 나이에 따라 영역을 분리하여 관리하는 세대별(Generation) 방식을 사용한다. 세대별 방식의 Young영역은 복사(Copying) 방식으로 관리되는 Eden과 두 개의 Survivor 영역으로 구성되고, 일정한 나이에 도달한 객체는 Old영역으로 복사된다. Old영역은 가상머신 종류에 따라 다른 알고리즘이 적용되는데, HotSpot ClientVM은 live객체를 모두 인접하게 이동시켜 단편화를 최소화하는 마크-압축(mark-compact)기법을 사용하고, ServerVM은 점진적 회수 방식인 Train 알고리즘을 사용한다.[5, 6] 이러한 다수의 알고리즘은 구현에 따르는 오버헤드가 크기 때문에[2], KVM과 같이 제한된 환경에 적용되기에는 적합하지 않다.

## 3. KVM을 위한 가비지 콜렉터 개선

프로그램 실행 중에 첫 번째 콜렉션이 발생하기 전에 생성된 객체들 중 루트 셋으로부터 도달 가능한 대부분의 객체는 프로그램 종료 시점까지 회수되지 않는 경향이 있다.[7] 그러므로, 본 논문에서는 객체의 위치를 변경하지 않는 마크-회수 방식에서 객체의 할당 방법을 단편화 문제 개선에 있어서 중요한 부분으로 보고 할당기의 알고리즘을 개선하였다.

3.1. 할당 영역의 구조

[그림 1]은 실행 프로그램의 요청에 따라 객체를 할당할 영역에 대한 초기 구조를 나타내고 있다. 할당 영역은 영역 구분 점에 의해 두 개의 영역으로 구분된다. 영역 구분 점의 위치는 실행 프로그램의 할당 요청에 따라 가변적이다.




[그림 1] 할당 영역의 초기 구조

3.2. 할당기 구조

할당기는 메모리 할당 요청을 받아 요청 크기를 확인하고 이에 따른 할당 함수를 호출하는 부분과 실제 할당 부분으로 구성된다. 할당 함수는 이중 링크드 리스트에 대한 자신의 시작 포인터를 이용하여 적합한 위치를 찾아 필요한 정보를 저장하고, 포인터를 조정한 후 해당 위치의 주소를 반환한다. 만약 적합한 위치를 찾지 못했다면 NULL값을 반환하여 가비지 콜렉터가 실행되도록 한다. 가비지 콜렉터 호출 후에도 할당에 실패하면 OutOfMemoryError가 발생하고 프로그램은 종료된다.

3.3. 개선된 할당 알고리즘

회수단계는 회수한 영역을 메모리 주소 순서대로 인접한 공간으로 병합하여 하나의 이중 링크드 리스트를 구성한다. 프로그램 시작 시점에 메모리는 두 개의 영역으로 나뉘어지고 각 영역의 시작 번지를 지시하는 두 개의 포인터가 존재한다.

[표 1]은 할당기의 알고리즘이다. KNIFE에 지정된 크기와 비교하여 서로 다른 할당 함수를 호출한다. 지정된 크기 보다 작은 요청은 첫 번째 노드에서 마지막 노드 방향으로 할당되는 mallocFrontAlloc에서 처리되고, 큰 요청은 마지막 노드에서 첫 번째 노드 방향으로 할당되는 mallocBackAlloc에서 처리되어 할당 영역의 시작위치를 반환한다. 각 노드는 상황에 따라 분리되며 리스트의 왼쪽과 오른쪽 시작 위치를 나타내는 두 개의 포인터는 서로의 값을 조절하여 충돌하지 않는다.

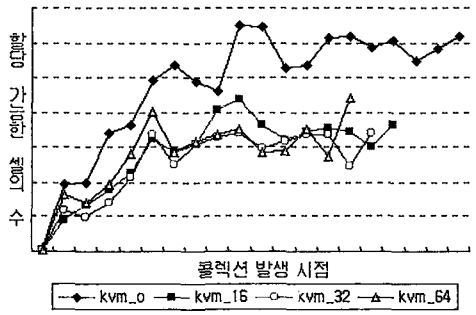
[표 1] 할당 알고리즘

```
cell* allocateFreeChunk(long 요청크기) {
   return (요청크기 < KNIFE) ? mallocFrontAlloc(요청크기) :
                               mallocBackAlloc(요청크기);
}
cell* mallocFrontAlloc(long 요청크기) {
   CHUNK thisChunk = 리스트의 왼쪽 시작 위치;
   while(thisChunk) {
            .....
      if(현재공간의 크기 > 요청크기) {
         할당위치 = 현재공간시작주소
         현재공간시작주소 += 요청크기
      } else if(현재공간의 크기 == 요청크기) { .... }
            .......
      thisChunk = thisChunk->next;
   }
  return NULL;
}
cell* mallocBackAlloc(long 요청크기) {
   CHUNK thisChunk = 리스트의 오른쪽 시작위치;
   while(thisChunk) {
         ....
         할당위치 = 현재공간의 마지막주소 - 요청크기
         ....
      thisChunk = thisChunk->prev;
   }
   return NULL;
}
```

## 4. 실험 및 성능평가

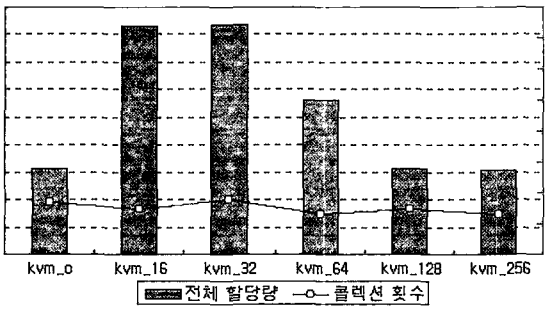
실험환경은 SUN SPARC Ultra-60 하드웨어에 운영체제는 Sun OS 5.7, gcc와 cc를 이용하였다. KVM을 위한 메모리 영역은 150KB로 제한하고 할당은 기존 알고리즘을 그대로 이용하는 경우와 16, 32, 64, 128, 256개의 셀을 기준으로 개선된 할당 알고리즘을 적용한 경우에 대해 실험했다. 반복적인 요청에 대해 메모리 부족 에러가 발생하는 시점까지의 여러 가지 요소에 대해 비교하였다.

다음 결과는 서로 다른 수명 시간을 가지는 1byte ~ 1024byte의 임의의 크기 객체가 평균 136byte로 각 할당기에 메모리를 요구하는 경우이다.




[그림 2] 콜렉션 발생 시점의 할당 가능한 셀의 수

[그림 2]는 기존 알고리즘과 제안하는 방법에서 콜렉션 발생시점에 남아 있는 할당 가능한 셀의 수를 나타내고 있다. 그래프의 끝 지점은 더 이상 할당이 불가능하여 프로그램이 종료된 시점이다. [그림 2]에 나타난 것과 같이 단방향으로 모든 객체를 할당하는 기존 알고리즘에서 단편화 문제가 가장 심각함을 보여주고 있다.




[그림 3] 콜렉션 횟수와 전체 할당량

[그림 3]은 가비지 콜렉션이 발생한 후에도 할당 요청이 거부되어 에러가 발생한 시점까지의 전체 콜렉션 횟수와 할당량이다. 대부분 기존 알고리즘과 유사하거나 높은 성능을 나타낸다. kvm_32의 경우 기존 알고리즘 보다 11,455셀이 더 많이 할당되었고, 콜렉션 발생 횟수는 4회 감소하였다.

## 5. 결론 및 향후 연구 과제

KVM의 가비지 콜렉션 방식으로 채택된 마크회수 방식은 콜렉션이 진행함에 따라 단편화가 발생하여 충분한 메모리 공간이 있음에도 불구하고 프로그램의 실행이 중단되는 문제점을 가지고 있다.

본 논문에서는 객체의 크기에 따라 할당위치를 다르게 결정하는 방법을 제안하고, 실험을 통해 성능을 확인하였다. 마크회수 방식은 콜렉션이 필요한 시점에 사용자의 프로그램이 중단되고 콜렉션이 완료되면 다시 실행되므로, 실시간 시스템에서는 적합하지 않아 이를 지원하기 위한 부분을 향후 연구과제로 하여 보완할 것이다.

## 6. 참고문헌

[1] Sun microsystems, Connected, Limited Device Configuration specification Version 1.0a, 2000

[2] Sun microsystems, Java 2 Platform Micro Edition(J2ME) Technology for Creating Mobile Devices White Paper, 2000

[3] Paul R. Wilson, Uniprocessor Garbage Collection Techniques, University of Texas, 1992

[4] Eric Giguere, Java 2 Micro Edition, WILEY 2000

[5] Sun microsystems, The HotSpot Virtual Machine Technical White Paper, 2001

[6] Jacob Seligmann, Steffen Grarup, Incremental Mature Garbage Collection Using the Train Algorithm, 1995

[7] Sun microsystems, Tuning Garbage Collection with the 1.3.1 Java Virtual Machine