제 18 회 한국정보처리학회 추계학술발표대회 논문집 제 9 권 제 2 호 (2002.11)

실시간 Java 를 위한 점진적 가비지 콜렉터에 관한 연구

배유석*, 원희선*, 문경덕*
* ETRI 컴퓨터소프트웨어연구소 정보가전연구부
e-mail : baeys@etri.re.kr

A Study on Incremental Garbage Collector for Real-Time Java

Yu-Seok Bae*, Hee-Sun Won*, Kyeong-Deok Moon*
* Information Appliance Technology Dept., Computer & Software Lab., ETRI

요 약

디지털 TV, 셋톱박스, 홈 서버 등 정보 가전 분야에서 Java 기술을 적용하는 범위가 확대되고 있지만, 현재 Sun 의 JVM 이나 공개용 Kaffe VM 등의 경우에 스케줄링 측면이나 가비지 콜렉터의 관점에서 실시간 조건을 고려하고 있지 않기 때문에 실시간 응용 분야에 바로 적용하기에는 문제점을 갖고 있다. 특히, 가비지 콜렉터의 측면에서는 가비지 회수 동안 응용 프로그램의 수행을 중지시키는 stop-the-world 방식으로 진행되어 응답 시간 및 실행 시간 예측 불가로 인해 실시간 시스템에 적합하지 않다. 따라서 작은 단위의 가비지 컬렉션 기능과 응용 프로그램의 수행을 병행하는 점진적(incremental) 가비지 콜렉터의 설계가 필요하다. 본 논문에서는 가비지 콜렉터의 관점에서 실시간 특성을 보장하는 점진적 가비지 컬렉션 방법에 관해 살펴본다.

1. 서론

최근 우수한 보안 기능과 동적 응용 프로그램 수행 기능 등을 지원하는 Java 기술을 디지털 TV 나 셋톱 박스, 홈 서버 등의 정보 가전 분야에 적용하려는 연구가 활발히 진행되고 있다[4, 6]. 그 예로 Sun 에서 임베디드 시스템을 지원하기 위해 PersonalJava 와 EmbeddedJava 버전을 제시하였지만, 그 버전은 단순한 Java Standard 버전의 축소판일 뿐이며, 실시간 스케줄링이나 메모리 관리 측면에서 실시간 특성에 대한 고려가 전혀 되어 있지 않다. 따라서 RTJEG(Real-Time Java Expert Group Java)에서 실시간 스케줄링과 메모리 관리, 비동기 이벤트 서비스 등을 추가하여 Java 에서 실시간 지원을 위한 표준 문서 RTSJ(Real-time Specification for Java)를 만들고 있으며, 현재 실시간 Java API 를 포함하여 1.0 버전(JSR-000001)이 나와 있으며, Timesys 는 RTSJ(JSR-000001)를 지원하는 CVM 버전을 개발하여 배포하고 있다[6, 9].

Sun 의 JVM 이나 Kaffe VM 등에서 보편적으로 사용되는 방법으로는 Mark-Sweep, Mark-Compact, Copying, Generational 가비지 콜렉터 등이 사용되고 있지만 이러한 방법들은 가비지 컬렉션 시간 동안 응용 프로그램의 수행을 정지시키고 가비지를 회수하는 stop-the-world 방식으로 가비지 컬렉션 시간 예측이 불가능하고 가비지 콜렉션 동안 응용 프로그램 수행이 정지되어 응답 시간을 지연시키는 문제점을 안고 있다[5, 8].

따라서, 본 논문에서는 가상 메모리를 지원하지 않는 실 메모리 환경에서 메모리 이용 효율성 보다는 실시간 성능의 관점에서 점진적인 메모리 회수 기능을 제공하여 응답 시간과 실행 시간 예측 기능을 제공하는 자동 메모리 관리 기법을 제시하고자 한다.

본 논문은 제 2 장에서 관련 연구 및 연구 방향을 제시하고, 제 3 장에서는 일반적인 복사 가비지 콜렉터의 동작 원리를 설명하며, 제 4 장에서는 점진적 가비지 콜렉터에 관해 언급하며, 제 5 장에서 결론을 맺는다.

2. 관련 연구 및 연구 방향

사용자에 의한 명시적인 메모리 관리 기능의 오류를 막고 사용자의 메모리 관리에 대한 오버헤드를 제거하기 위해 제시된 자동적인 가비지 콜렉터 방법으로 참조 계수, 마크-회수, 복사, 세대별 컬렉터로 나눌 수 있다.

참조 계수(reference counting) 방법은 객체의 참조 계수를 비교하여 메모리를 회수하는 방법으로 구현하기는 간단하지만, 사이클을 갖는 객체 그래프를 탐색하기 어렵고, 참조가 생길 때 마다 참조 계수를 변경해야 하는 카운팅 오버헤드로 인해 최적화된 알고리즘 개발이 어려우며, 메모리 압축(compaction) 방법이 제공되지 않기 때문에 메모리 단편화(fragmentation) 현상이 발생함으로 실제로 많이 이용되지 않고 있다[3, 5, 10].

마크-회수(mark-sweep) 방법은 전체 메모리를 탐색하여 라이브(live) 객체와 가비지(garbage) 객체를 구분하여 마크하는 단계와 가비지 객체의 공간을 회수하는 회수 단계로 구성되는데, 현재 PersonalJava 와 Kaffe VM 에서 사용되고 있는 방법으로 마크 비트를 이용함으로 공간 오버헤드는 작지만, 모든 메모리 공간을 2 번 탐색하는 시간 오버헤드를 지니고 있으며, 기본적으로 stop-the-world 방식의 가비지 컬렉션 방법으로 실시간 시스템에 적용하기 어렵다. 한편 단편화된 메모리를 압축하기 위해 회수단계에 별도의 압축 기능을 추가할 수 있는데, 추가적인 메모리와 수행 시간을 필요로 한다[3, 4, 5, 10].

실시간 시스템에 가장 많이 사용되는 가비지 콜렉터로 복사(copying) 컬렉터를 들 수 있다. 메모리 공간을 균등하게 두 부분으로 나누고, 이 중 하나의 메모리에 계속적인 할당을 수행하다가 더 이상 할당할 수 없는 경우에 라이브 객체를 다른 쪽 메모리로 이동하여 메모리 할당 과정을 수행하는 메모리 관리 방법으로 단편화 문제가 자동적으로 해결되지만, 다른 방법에 비해 2 배의 메모리를 필요로 하는 단점이 있다. 특히, 가상 메모리가 있는 경우에 효율이 뛰어나지만 가상 메모리를 지원하지 않고 메모리 제약조건을 갖는 임베디드 시스템에는 적합하지 않다[3, 5, 10].

세대별(generational) 컬렉터는 복사 가비지 컬렉션 방법을 개선하여 객체에 나이라는 개념을 도입하여 나이에 따라 세대별로 객체를 나누어 관리함으로써 라이브 객체를 복사 빈도를 줄여서 전체적으로 복사에 드는 시간을 줄일 수 있다. 이 방법은 시간의 흐름에 따라 객체가 다른 영역으로 복사되거나 다른 세대로 이동함으로 수명이 긴 객체가 짧은 객체 보다 상대적으로 많을수록 유리하지만 반대의 경우 복사 방식보다 불리하다. 또한 객체의 나이에 따라 구분된 세대별로 적합한 다른 컬렉터를 사용하여 메모리 관리 효율을 높일 수 있다. 이러한 방법이 Sun 의 최신 메모리 관리 기법인 HotSpot 에 적용되고 있다[8]. 하지만, 이 방법도 세대별 관리 오버헤드가 증가하며, 세대별 객체 저장 공간을 별도로 필요로 하기에 메모리 요구사항이 높음으로 임베디드 시스템에 부적합하다[3, 10].

또한 비 실시간 stop-the-world 방식의 가비지 콜렉션에서 벗어나 실시간 기능을 지원하기 위해 응용프로그램과 가비지 콜렉터의 수행을 병행하여 가비지 컬렉션 수행 시간에 대한 예측 가능성 및 사용자의 반응 시간을 단축하는 점진적(incremental) 가비지 콜렉션에 관한 연구가 진행되고 있다[3, 10].

본 논문에서는 정보 가전 분야의 실시간 시스템에 적용할 수 있도록 실 메모리 환경에서 메모리 이용 효율성 보다는 실행 프로그램의 예측 가능성과 사용자의 응답 시간을 향상시킬 수 있는 점진적 복사 방식의 가비지 콜렉터를 제시한다.

3. 복사 가비지 콜렉터의 동작 원리

점진적 가비지 콜렉터는 기본적으로 메모리 영역을 분할하여 관리하는 복사 방식에 의존하며, 메모리 이용 효율성 보다는 실시간 특성을 만족하는데 초점을 맞추어 점진적인 가비지 컬렉션 특성을 통하여 사용자의 응답 시간을 줄여주는 특성을 제공한다.

복사 방식의 가비지 콜렉터에서는 전체 메모리를 From-space 와 To-space 의 2 개의 semi-space 로 분할하여 메모리를 관리한다. Flip 과정에서 From-space 에서 라이브 객체를 To-space 로 복사하며 From-space 에 포워딩(forwarding) 포인터를 저장하여 응용 프로그램에서 참조하는 포인터 참조 관계를 유지한다. 복사 방식의 가비지 콜렉터는 마크 회수 방식의 가비지 콜렉터와 달리 라이브 객체를 분리된 메모리로 이동함으로 압축 과정이 자동적으로 이루어져 메모리 단편화 현상이 발생하지 않으며, free 된 연속적인 메모리 구조를 갖기 때문에 객체 크기에 관계없이 신속한 할당이 가능하며 할당 시간이 일정 시간으로 바운드될 수 있어 실시간 환경에 적합하다[3, 10].

복사 방식의 가비지 콜렉터의 할당 알고리즘은 표 1과 같다[3].

```
New(n) =
    if (free + n > top_of_space)
        flip()
    if (free +n > top_of_space)
        abort "memory exhausted"
    newcell = free
    free = free + n
    return newcell
```

표 2 복사 컬렉터에서의 할당 알고리즘

또한 flip 과정과 라이브 객체를 이동할 때 Cheney 알고리즘을 적용하며, flip 과 copy 에서 사용되는 알고리즘은 표 3와 같다[1, 3].

```
flip() =
    From-space, To-space = To-space, From-space
    scan = free = To-space
    for R in Roots
        R = copy(R)
    while (scan < free)
        for P in Children(scan)
            p = copy(*p)
```

```
        scan = scan + size(scan)
copy(P) =
    if (forwared(p))
        return forwarding_address(p)
    else
        addr = free
        move(p, free)
        free = free + size(p)
        forwarding_address(p) = addr
        return addr
```

표 4 Cheney 알고리즘

Cheney 의 알고리즘에서는 복사 방식의 탐색 알고리즘으로 Dijkstra 에 의해 제안된 On-the-fly 마킹 알고리즘을 모델로 하며, 모든 객체는 3 가지 칼라로 구분되며 칼라가 갖는 객체의 특성은 표 5과 같다[1, 2, 3].

Black: 가비지 컬렉터에서 탐색 과정이 완료된 라이브 객체
Grey: 해당노드는 탐색되었지만, 하위 노드의 객체가 완전히 탐색 되지 않은 객체
White: 탐색되지 않은 객체로 컬렉션 사이클 종료시 회수되는 garbage 객체

표 6 컬렉션 과정에서 칼라별 객체의 특성

가비지 콜렉터는 From-space 에서 도달 가능한 White 객체를 To-space 로 이동한 후, Grey 객체로 설정한다. 객체 그래프 탐색 과정은 Black 객체와 White 객체를 분리하는 Grey 객체의 흐름을 따라 진행된다. Grey 객체는 White 상태에서 가비지 콜렉터의 탐색 과정에서 도달되어 To-space 로 이동되었지만 아직 하위 노드의 객체가 검색되지 않은 객체로 하위 객체의 탐색이 완료되면 Black 으로 변경되고, 하위 객체는 Grey 로 바뀌는 과정을 반복하며, 모든 도달 가능한 객체가 To-space 로 이동되어 Black 이 되면 가비지 회수 작업은 완료된다.

객체 그래프의 탐색 과정은 Cheney 알고리즘의 breadth-first 방식을 따라 수행되며, 그림 1은 복사 방식에서의 탐색 및 복사 과정을 보여준다[3].

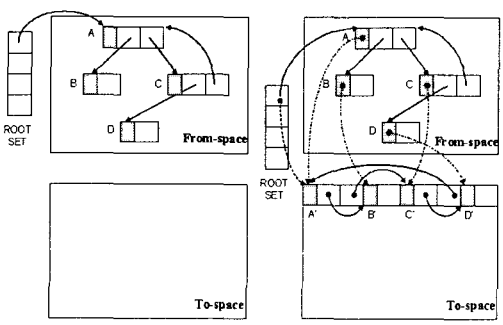

그림 2 객체 탐색 및 복사

4. 점진적 가비지 콜렉터

점진적 복사 가비지 콜렉터는 복사 가비지 콜렉터에 점진적 특성을 가미한 것으로, 응용 프로그램인 뮤테이터(mutator)와 가비지 콜렉터의 실행을 주기적으로 병행하는 방법으로 이 경우 뮤테이터의 객체 그래프 변경으로 인해 가비지 컬렉터가 참조하는 객체 그래프 탐색이 실패하는 경우를 유발할 수 있기 때문에 배리어(barrier)를 이용하여 객체 그래프의 일관성 유지가 필요하다.

모든 탐색 알고리즘에서 기본적인 요구사항이 모든 도달 가능한 객체를 찾는 것으로 이를 위해 컬렉터의 객체 도달 그래프에 대한 뷰(view)가 정확해야 하지만, 응용 프로그램인 뮤테이터가 객체 그래프를 수정하는 경우 이를 보장하지 못한다. 뮤테이터가 White 객체나 Grey 객체를 수정하는 경우는 컬렉터가 향후에 다시 방문하기 때문에 문제가 되지 않는다. 또한, Black 객체나 Grey 객체에 포인터를 추가하거나 제거함으로써 Black 객체를 수정하는 경우도 벌써 컬렉터에 의해 도달 가능한 객체로 판명되었기 때문에 역시 문제가 되지 않는다. 하지만 만일 Black 객체에 White 객체에 대한 포인터를 추가하는 경우, 만일 컬렉터가 그 White 객체에 대한 어떤 다른 경로를 알 수 없는 경우에 그 객체를 방문할 수 있는 방법이 없음으로 라이브 객체를 가비지로 회수하는 문제를 유발할 수 있다. 따라서 컬렉터의 완전한 객체 도달 그래프 탐색을 보장하기 위해 표 7의 트라이칼라 불변 법칙을 만족해야 하며, 강한 법칙은 약한 법칙을 내포한다[3, 10].

구 분	특 성
약한 (weak)	어떤 Black 객체에 의해 지정되는 모든 White 객체는 Grey 노드를 통하여 도달 가능하다
강한 (strong)	어떤 Black 객체에서 어떤 White 객체로의 포인터는 존재하지 않는다

표 8 트라이칼라(Tri-color) 불변 법칙

한편 가비지 컬렉션 과정에서 뮤테이터와 가비지 콜렉터의 작업을 교대로 수행하는 점진적 특성을 지원하는 환경에서는 뮤테이터의 객체 그래프 접근이나 변경으로 인해 컬렉터의 객체 탐색 과정의 오류를 유발시킬 수 있으므로, 트라이칼라 불변 법칙을 보장하기 위해 배리어(barrier)를 설정하여 뮤테이터의 객체에 대한 접근이 완료되기전에 컬렉터가 어떤 행위를 취하여 정상적인 가비지 컬렉션 과정이 수행될 수 있도록 해야 한다. 배리어는 읽기 배리어(read barrier)와 쓰기 배리어(write barrier)로 구분되며, 읽기 배리어가 구현하기 쉽지만 뮤테이터의 객체 그래프 접근으로 인해 컬렉터의 회수 작업이 간섭 받음으로써 쓰기 배리어에 비해 성능이 떨어진다[3, 10].

읽기 배리어는 뮤테이터의 White 객체에 대한 접근을 탐지하는 즉시 컬렉터가 해당 객체를 방문하여 Grey 로 변경하여 컬렉터가 탐색할 수 있도록 하며, 쓰기 배리어는 뮤테이터의 Black 객체에서 White 객체

에 대한 포인터 쓰기가 발생할 경우 컬렉터가 White 객체를 방문할 수 있도록 정보를 기록한다[3, 10].

Wilson 은 쓰기 배리어에서 컬렉터가 객체 도달 그래프 탐색을 실패하는 두 가지 조건을 정의하고 있으며, 표 5 와 같다.

조건 1	뮤테이터가 Black 객체에서 White 객체로의 포인터를 쓰는 경우
조건 2	컬렉터가 알기전 White 객체로의 원래의 경로가 지워지는 경우

표 5 객체 그래프 탐색을 실패하는 두 가지 조건

위의 두 가지 조건에 따라 쓰기 배리어는 점진적 갱신(incremental-update) 배리어와 시작 순간 스냅샷(snapshot-at-the-beginning) 배리어로 구분된다[3, 10].

점진적 갱신 배리어는 <조건 1>이 발생되지 않도록 하며, 시작 순간 스냅샷 배리어는 <조건 2>가 일어나지 않도록 한다. 시작 순간 스냅샷 배리어 알고리즘은 약한 트라이칼라 불변 법칙을 보존하는 방법으로 White 객체에 대한 원래 참조의 손실을 방지하기 위해 포인터가 갱신될 때마다 원래의 참조 객체를 Grey 로 변경하여 컬렉션 과정에 반영한다. 한편, 점진적 갱신 배리어 알고리즘은 강한 트라이칼라 불변 법칙을 보존하여 잠재적으로 중단시키는(disruptive) 포인터 쓰기를 기록하여 Black 객체에 대한 변화에 관해 컬렉터가 반영할 수 있도록 유지한다. 시작 순간 스냅샷 배리어 알고리즘은 하나의 가비지 컬렉션 사이클에서 가비지가 되는 어떠한 객체도 그 사이클에서 회수되지 않고 다음 사이클까지 기다려야 됨으로 매우 보수적인(conservative) 방법이다. 한편, 점진적 갱신 배리어 알고리즘은 점진적으로 뮤테이터에 의해 만들어진 변화를 그래프 모양에 기록하여 처리함으로 시작 순간 스냅샷 배리어 알고리즘에 비해 덜 보수적인 방법으로 구현상에 복잡도는 증가하나 성능에서 이점을 볼 수 있다.

이상으로 볼 때 점진적 가비지 컬렉터의 경우 성능 측면에서 점진적 갱신 배리어를 적용하여 객체 도달 그래프를 유지하는 것이 효과적이다.

5. 결론

본 논문에서는 실시간 Java 를 위한 점진적 가비지 콜렉터의 설계 및 운용 방안을 제시하고, 비실시간 가비지 콜렉터를 대체하여, 실시간 환경에서 효과적으로 메모리를 관리할 수 있는 점진적 복사 가비지 콜렉터에 관해 언급하였다. 점진적 복사 가비지 콜렉터는 메모리 이용 효율보다는 실시간 시스템에 적용할 수 있도록 할당, 복사, 회수 시간의 일정 시간 바운드(constant time bound)에 초점을 맞추고 있으며, 응용프로그램의 수행과 병행하여 주기적으로 작은 단위의 가비지 컬렉션 기능을 수행함으로써 점진적인 실행 특성을 제공하여 실시간 시스템에서 요구되는 응용 프로그램의 응답 시간을 줄이는데 목적이 있다. 따라서 본 논문에서 제시하는 가비지 컬렉터에서는 일반적인 VM 에서 많이 사용되는 동적 메모리 확장 기능은 고려하지 않는다. 그 이유는 할당시 메모리 부족으로 인해 동적인 메모리를 확장할 경우 바운드된 할당 시간을 초과하게 되어 실시간 특성에 적합하지 않으므로 메모리 초기화 시 설정된 메모리 한도내에서 가비지 컬렉션 기능을 지원하도록 한다. 점진적 가비지 콜렉터 기능은 멀티쓰레드 환경하에서 응용 프로그램과 병행해서 수행됨으로 가비지 콜렉터의 성능 향상을 위해 실시간 쓰레드 스케줄링과 가비지 콜렉터의 오버헤드 감소 및 성능 개선에 대한 연구가 추가로 필요하다.

참고문헌

[1] C. J. Cheney, "A nonrecursive list compacting algorithm," Comm. of the ACM, Vol. 13, No. 11, pp. 677-678, November 1970.

[2] E. W. Dijkstra, Leslie Lamport, A.J. Martin, C.S. Scholten, and E.F.M. Steffens, "On-the-fly garbage collection: An exercise in cooperation," Comm. of ACM, Vol. 21, No. 11, pp. 965-975, November 1978.

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

[4] K. Nilsen, "Issues in the Design and Implementation of Real-Time Java," Java Developer's Journal, Vol. 1, No. 1, pp. 44-57, 1996.

[5] A. Petit-Bianco, No Silver Bullet – Garbage Collection for Java in Embedded Systems, Cygnus Solution, August 1998.

[6] Real-Time Specification for Java, http://www.rtsj.org

[8] Sun Microsystems Inc., The Java HotSpot Performance Engine Architecture: A White Paper, April 1999.

[9] Timesys, Real-Time Java, http://www.timesys.com

[10] P. Wilson, "Uniprocessor Garbage Collection Techniques," International Workshop on Memory Management, Number 637 Lecture Notes in Computer Science, pp. 1-42, St. Malo, France, September 1992. Springer-Verlag.