Java 기반 임베디드 시스템을 위한 쓰레기 수집기 설계

°
배 유 석, 김 태 근
ETRI 컴퓨터 & 소프트웨어 기술 연구소 실시간컴퓨팅연구부 분산컴퓨팅연구팀

Design of Garbage Collector for Java-based Embedded Systems

Yuseok Bae, Taegeun Kim
Distributed Computing Team, Real-Time Computing Dept., Computer & Software Technology Lab., ETRI

요 약

가전 제품이나 정보 가전 기기를 포함하는 실시간 임베디드 응용 분야에서 Java를 소프트웨어 개발 플랫폼으로 선택하는 범위가 확대되고 있다. 현재 Sun에서 제공하는 PersonalJava 기술은 동적인 메모리 할당에 대하여 마크-회수(mark-sweep) 방식의 자동 쓰레기 수집(garbage collection) 기능을 지원하는데, 이 방법은 쓰레기 회수 수행전 응용프로그램의 수행을 중지시키는 방식으로 실시간 임베디드 환경에 적합하지 않다. 본 논문에서는 Java 기반 인터넷 셋톱박스나 디지털 가전기기에 탑재할 수 있는 Java 플랫폼을 대상으로 기존에 적용된 비실시간 마크-회수 메모리 관리 기법을 대체하는 실시간 자동 메모리 관리 기법을 제시한다.

1. 서론

Java는 가전기기의 상호 운용성을 높이기 위해서 처음 개발되었지만, 인터넷의 보급과 더불어 동적인 인터넷을 구성하는 네트워크 언어로 각광을 받았으며, 현재 본래의 목적인 네트워크 장치와 실시간 임베디드 시스템을 위한 새로운 프로그래밍 언어로 각광 받고 있다[4, 6]. 이러한 분야를 위한 Java 기술로 PersonalJava와 EmbeddedJava가 있다. 한편 임베디드 시스템은 RAM이나 ROM에서 메모리 제약조건을 가지며, 가상 메모리를 제공하지 않음으로 이에 맞는 메모리 관리 기법을 필요로 한다[5].

현재 Java에서의 자동적인 메모리 관리는 마크-회수 쓰레기 수집기를 통하여 동적으로 할당된 메모리를 회수하고 있지만, 불필요한 데이터를 정리하는 동안 응용프로그램의 수행이 중지되는 스탑더월드(stop-the-world) 방식으로 실시간 특성이 고려되어 있지 않다. 또한 Java의 수행 성능을 향상시키는 HotSpot 성능 엔진에서 적용된 메모리 관리 기법은 성능을 향상시키기 위해 추가적인 메모리를 많이 요구하기 때문에 임베디드 환경에 적용하는 것은 불가능하다[7].

따라서, 본 논문에서는 네트워킹과 애플릿 수행을 지원하는 PersonalJava 플랫폼을 대상으로 가상 메모리를 지원하지 않는 실 메모리 환경에서 메모리 이용효율을 향상시키고, 점진적인 메모리 회수 기능을 제공하여 사용자 반응 시간을 단축 시키는 자동 메모리 관리 기법을 제시하고자 한다.

본 논문은 제 2장에서 관련 연구 및 연구 방향을 제시하고, 제 3장에서는 점진적 비복사 쓰레기 수집기의 동작 원리를 설명하며, 제 4장에서는 점진적 비복사 쓰레기 수집기에서의 성능 개선 사항을 논하며, 제 5장에서 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 쓰레기 수집기 방법으로 참조 계수, 마크-회수, 복사, 세대별 수집기로 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[2, 5, 8].

마크-회수(mark-sweep) 방법은 전체 메모리를 탐색하여 live 객체와 dead (또는 garbage) 객체를 구분하여 마크하는 단계와 dead 객체의 공간을 회수하는 회수 단계로 구성되는데, 현재 PersonalJava에서 사용되고 있는 방법으로 마크 비트를 이용함으로 공간 오버헤드는 작지만, 모

든 메모리 공간을 두 번 탐색하는 시간 오버헤드를 지니고 있으며, 기본적으로 스탑더월드 방식의 쓰레기 수집 방법으로 실시간 시스템에 적용하기 어렵다. 한편 단편화된 메모리를 압축하기 위해 회수단계에 별도의 압축 기능을 추가할 수 있는데, 추가적인 메모리와 수행 시간을 필요로 한다[2, 4, 5, 8].

실시간 시스템에 가장 많이 사용되는 쓰레기 수집기로 복사(copying) 수집기를 들 수 있다. 이 수집기는 메모리 공간을 균등하게 두 부분으로 나누고, 이 중 하나의 메모리에 계속적인 할당을 수행하다가 더 이상 할당할 수 없는 경우에 live 객체를 다른 쪽 메모리로 이동하여 메모리 할당 과정을 수행하는 메모리 관리 방법으로 단편화 문제가 자동적으로 해결되지만, 다른 방법에 비해 두 배의 메모리를 필요로 하는 단점이 있다. 특히, 가상 메모리가 있는 경우에 효율이 뛰어나다. 하지만 가상 메모리를 지원하지 않고 메모리 제약조건을 갖는 임베디드 시스템에는 적합하지 않다[1, 2, 5, 8, 9].

세대별(generational) 수집기는 복사 쓰레기 수집 방법을 개선하여 객체에 나이라는 개념을 도입하여 나이에 따라 세대별로 객체를 관리함으로써 복사 방식의 단점인 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 수집기를 사용하여 메모리 관리 효율을 높일 수 있다. 이러한 방법이 Java 의 최신 메모리 관리 기법이 HotSpot 에 응용되고 있다. 하지만, 이 방법도 세대별 관리 오버헤드가 증가하며, 세대별 객체 저장 공간을 별도로 필요로 하기에 메모리 요구사항이 높음으로 임베디드 시스템에 부적합하다[2, 8].

또한 비실시간 스톱더월드 방식에서 벗어나 실시간 기능을 지원하기 위해 응용프로그램과 쓰레기 수집기의 수행을 병행하여 쓰레기 수집 수행 시간에 대한 예측 가능성 및 사용자의 반응 시간을 보장하는 점진적(incremental) 쓰레기 수집 기능이 지원되고 있다[2, 8].

본 논문에서는 임베디드 시스템에 탑재할 수 있도록 실 메모리 환경에서 가용 메모리를 효율적으로 사용할 수 있는 방법으로 물리적인 복사 대신에 메모리 영역을 링크드 리스트로 구성하여 논리적 복사의 역할을 수행하는 비복사 쓰레기 수집기를 제안하며, 또한 응용프로그램과 쓰레기 수집기의 병행 수행을 보장하기 위해 점진적인 쓰레기 수집 기능을 결합한 점진적 비복사 쓰레기 수집기를 제시한다.

3. 점진적 비복사 쓰레기 수집기의 동작원리

점진적 비복사 쓰레기 수집기는 별도의 메모리 공간을 이용한 객체의 복사를 수행하지 않음으로 임베디드 시스템 환경에서 실 메모리 공간을 효율적으로 이용할 수 있으며, 점진적인 쓰레기 수집 특성을 통하여 사용자의 반응 시간을 줄여주는 특성을 제공한다.

3.1 비복사를 위한 Treadmill 구조

본 연구에서는 Henry G. Baker 의 Treadmill 을 이용한 비복사 쓰레기 수집기를 모델로 하고 있다[1, 2, 3, 8, 9]

Treadmill 은 고정 크기를 갖는 객체에 대하여 일정 시간 할당 및 메모리 회수 기능을 제공하는 실시간 비복사 수집기로, 모든 객체는 원형 더블 링크드 리스트 형태로 유지되며, 4 가지 칼라 모델을 사용한다. Treadmill 의 구조는 그림 1 과 같다.

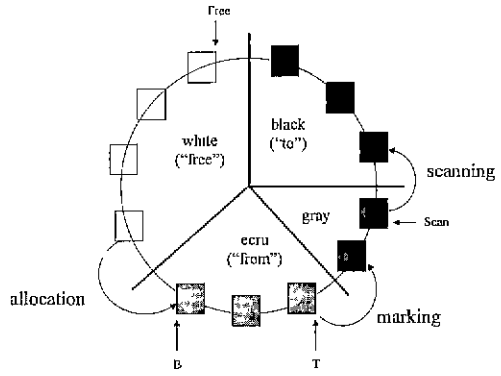

그림 1 Treadmill 수집기의 구조

Treadmill 은 4 가지 칼라에 기반한 수집 방법을 사용하는데, white 는 free 공간이며, gray 는 live 하지만 아직 하위 노드가 스캔되지 않은 객체를 의미하며, black 은 live 하면서 scan 이 완료된 객체이다. 한편 ecru 는 free 공간에서 할당된 객체로 아직 live 의 판정과 scan 이 되지 않은 객체이다. 각 칼라 영역의 구분은 Free, B(bottom), T(top), Scan 포인터로 구분된다.

3.2 Treadmill 의 동작원리 및 효과

Treadmill 에서의 객체의 할당은 쓰레기 수집이 시작하기 전에는 white 리스트에서 ecru 리스트로 이동하게 되지만, 쓰레기 수집 동안에는 white 에서 black 으로 할당하여 다음 사이클에 쓰레기 수집 과정을 수행하도록 한다. 수집 과정은 ecru 리스트가 dead 객체만을 포함할 때까지 ecru 리스트에서 live 객체를 제거함으로써 진행되며 남은 ecru 셀을 white 셀로 바꾸어 white 리스트에 추가하여 메모리를 회수한다.

쓰레기 수집 사이클은 scan 포인터가 T 포인터를 만나서 더 이상 gray 객체가 없을 때 완료된다. 또한 Free 포인터가 B 포인터를 만나면 두 가지 칼라인 ecru 와 black 만 남게 되며 dead 객체만을 갖는 ecru 를 white 로 바꾸고, B 와 T 포인터를 바꾸는 flip 과정을 수행하며, 이러한 수집 과정을 반복한다.

Treadmill 은 모든 객체가 같은 크기를 갖는 경우, 빈 공간을 탐색할 필요가 없으므로 할당 시간은 일정하지만, 다양한 가변 크기의 객체를 갖는 경우 단편화 현상으로 인해 메모리 이용 효율은 급격히 떨어진다[3, 9].

3.3 점진적 특성을 위한 배리어(barrier) 활용

Treadmill 에서 실시간 쓰레기 수집을 위해 응용프로그램과 쓰레기 수집기를 병행해서 수행하는데 이로 인해 수집 과정 중 응용프로그램의 ecru 셀의 접근으로 인해 발생하는 객체 도달 그래프의 변경을 막기 위해 배리어(barrier)를 사용한다. 배리어는 읽기(read) 배리어와 쓰기(write) 배리어로 구분되며, Baker 가 활용한 읽기 배리어는 응용프로그램이 ecru 객체에 접근할 때, 수집기가 즉시 그 객체를 방문하여 gray 리스트에 추가하여 scan 과정을 거치도록 하는 것으로 수집기의 작업이 방해를 받는 문제를 유발함으로 일반적으로 쓰기 배리어에 비해 성능이 떨어진다[2]. 쓰기 배리어는 실행프로그램의 ecru 객체에 대한 포인터 쓰기 발생시 그 정보를 기록하여 나중에 수집기가 그 노드를 재방문하여 탐색을 완료할 수 있도록 하는 방법이다[1, 2, 3, 8, 9].

4. 점진적 비복사 쓰레기 수집기의 성능 개선

4.1 비복사를 위한 메모리 모델과 단편화 처리

비복사를 위한 메모리 모델로 응용프로그램 로딩 전 Java 가상 머신에서 사용하는 객체에 대해서는 영구 객체(permanent object)로 간주하여 별도의 메모리에 탑재하여 쓰레기 수집 대상에서 제외한다.

또한 객체 핸들과 객체 리스트를 위한 Treadmill 을 별도로 구성하였다. 만일 핸들과 객체 영역을 포함하는 하나의 Treadmill 로 구성할 경우, 메모리 영역의 분할이나 병합시 핸들 정보가 포함되어 계산이 복잡하여 할당 시간의 지연을 초래함으로 객체 영역을 별도의 Treadmill 로 구성하여 할당의 편리를 도모할 수 있다. 또한 객체 영역을 동일한 크기를 갖는 Treadmill 로 구성할 경우, 심각한 단편화 문제를 초래할 수 있다[3]. 따라서 객체 크기별 분리된(segregated) free 리스트를 갖는 Treadmill 을 구성한다[3, 9]. 2^n의 방식으로 객체 데이터 크기별로 Treadmill 을 구성하고, 별도의 여유 메모리를 두어 특정 크기 영역이 부족할 경우의 메모리 할당에 대비한다.

4.2 메모리 할당 및 회수

메모리 할당은 우선 객체 요구 크기에 대하여 해당 Treadmill 을 탐색한 후, round-up 된 크기의 객체를 갖는 Treadmill 에 할당한다. 이 과정에서 특정 영역에 free 메모리가 부족한 경우, 여유 메모리에서 메모리를 할당 받아 분할하여 할당에 사용하며, 여유 메모리도 부족할 경우, 큰 메모리를 분할하거나 작은 메모리를 병합하여 메모리 할당에 활용한다. 메모리의 회수는 Treadmill 의 동작 원리에 의하여 처리한다.

4.3 쓰레드 수행과 배리어 적용

점진적 특성을 지원하기 위해 객체 할당 크기에 비례하여 쓰레기 수집기 수행 시간 설정하는데, 일반적으로 Free 가 B 를 만나기 전 scan 이 T 를 만나 스캔을 완료하는 것이 사용자의 수집 작업으로 인해 기다리는 시간을 줄일 수 있으며, 바로 flip 이 일어날 수 있기 때문에 효과적이다. 객체 할당 요구 크기에 비례하여 쓰레기 수집 factor 를 충분히 크게 주어 scan 작업을 완료하는 형태로 쓰레기 수집 수행 시간을 결정한다.

현재 시간 관리를 위한 쓰레드를 별도로 구현하여 응용프로그램 쓰레드와 수집기 쓰레드를 제어하고자 하며, 이 과정에서 응용프로그램의 객체 그래프의 변경으로 인한 탐색 실패 문제를 해결하기 위해서 점진적 쓰기 배리어를 적용하여 문제를 해결하고자 한다.

5. 결론

본 논문은 Java 기반 임베디드 시스템을 위한 쓰레기 수집기의 설계 및 운용 방안을 제시하고, 비실시간 환경을 가정한 Java 의 마크-회수 쓰레기 수집기를 대체하여, 임베디드 환경에서 효과적으로 메모리 활용도를 높일 수 있는 점진적 비복사 쓰레기 수집기의 설계에 관해 언급하였다.

현재 설계가 완료된 단계이며, 메모리 할당 및 쓰레드 제어 과정에 대한 개발이 진행되고 있는 상태이며, 향후 성능 평가와 회수 과정에서 추가적인 단편화 성능 개선에 관한 연구가 필요하다.

참고 문헌

[1] H. G. Baker, "The Treadmill: Real-Time Garbage Collection Without Motion Sickness," ACM SIGPLAN Notices, Vol. 27, No. 3, pp 66-70, March 1992.

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

[3] T. Lim, P. Pardyak, and B. Bershad, "A Memory-Efficient Real-Time Non-Copying Garbage Collector," 1998 International Symposium on Memory Management, pp.118-129, October 1998.

[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] Sun Microsystems Inc., *The Java Language Environment: A White Paper*, May 1996.

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

[8] 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.

[9] P. Wilson, and M. Johnstone, "Real-Time Non-Copying Garbage Collection, " Position paper for ACM OOPSLA 1993 Workshop on Memory Management and Garbage Collection, September 1993.