NVM 기반 시스템에서의 다중 타입 객체 지원 가비지 수집 기법

이도근[O] 손성배 이성진 원유집
한양대학교
matureelf@hanyang.ac.kr, seongbae.son@gmail.com, insight@hanyang.ac.kr, youjip.won@gmail.com

Garbage Collection Technique for Multi-type Object in NVM-based Systems

Dokeun Lee[O] Seongbae Son Seongjin Lee Youjip Won
Hanyang University

요 약

NVM에 존재하는 객체는 전원이 없이도 데이터 영구 보존이 가능하다. 이 객체들이 가비지화 될 경우 시스템에 치명적인 공간 오버헤드를 유발한다. 이 가비지들을 찾아내기 위해서는 객체가 어떤 형태로 구성되어 있는지 아는 것이 필수적이다. 본 논문에서는 사용자 자료구조의 형태를 메타데이터화 하여 다중 타입의 자료구조에 대해서도 가비지 수집이 가능하게 하였으며 가비지 수집을 빠르게 하기 위한 탐색 기법과 오버헤드 분산을 위한 가비지 수집 정책을 제안하였다. 또한, 이를 영속 힙 기반의 HEAPO에 구현하여 실험 및 검증하였다.

1. 서 론

Non-Volatile Memory(NVM) 에 존재하는 객체는 사용자가 명시적으로 삭제하지 않는 이상 영구적으로 메모리에 존재한다. 이 객체들은 자신에게 접근할 수 있는 메타데이터(포인터)가 존재하지 않을 경우 가비지가 되어 메모리 공간을 영구적으로 차지한다. 이러한 가비지들이 지속적으로 쌓이게 되면 시스템에 치명적인 공간 오버헤드를 유발하기 때문에 이를 해결하기 위한 가비지 수집 기법이 필수적이다. 기 개발되었던 대다수의 가비지 수집기법은 사용자가 데이터에 접근이 가능한지의 여부를 바탕으로 가비지를 판별하였다. 언어 레벨에서 참조 횟수를 관리하는 자바와는 다르게, NVM 기반의 메모리 할당 기법[1][2][3]들에서는 객체의 참조 횟수를 관리하기가 어려워 기초적인 가비지 수집을 하는데 어려움이 있었다. 시스템 레벨에서 객체가 참조 되고 있는지 아닌지 여부를 판별하려면 객체가 어떻게 구성되어 있는지가 기록된 '타입 정보'와 '포인터 위치'등이 반드시 필요하다. 본 논문에서는 NVM기반 시스템에서 객체들의 타입 정보를 메타데이터화 하여 여러 형태의 사용자 객체에 대하여도 가비지 수집이 가능하게 하는 기법을 제안한다. 제안된 기법은 영속 힙 방식의 HEAPO[4] 플랫폼에 구현하였고 실제 머신에서의 실험을 통해 본 기법의 타당성을 검증하였다.

2. 설 계

2.1 다중 타입 지원 방식

HEAPO의 객체저장소는 하나의 자료구조 단위의 데이터가 저장된다. 본 연구에서는 이를 이용하여 객체저장소를 표현하는 메타데이터(descriptor)에 자료구조 형태를 표시하는 메타데이터를 추가하였다. 이는 사용자가 객체저장소를 만들 때 사용자에 의해 갱신되고, 이후 객체저장소를 프로세스 주소공간에 사상한 후 접근하여 사용할 수 있다. 본 연구에서는 이를 위해 자료구조 표현 메타데이터를 저장하고 참조하는 시스템 콜을 추가하여 사용자에게 제공하였다.

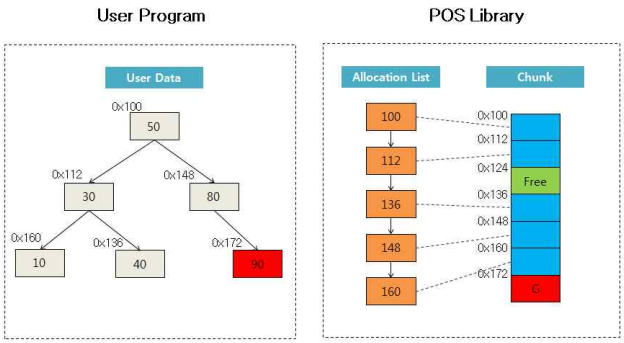

그림 1. 가비지 탐색 기법

2.2 가비지 탐색 기법

HEAPO의 힙 관리 방식은 glibc의 malloc()에서 사용하는 청크 기법[5]과 동일하다. 따라서 본 논문에서는 가비지를 할당은 되었으나 더 이상 접근이 불가능한 메모리 청크로 정의하였다. 어떠한 메모리 청크가 할당되어 있는지의 여부를 알기 위해서는 청크들의 할당 정보가 필요하다. 하지만 malloc() 알고리즘에서는 자유 청크 리스트만 관리할 뿐 할당된 청크들의 정보는 관리하지 않는다. 본 논문에서는 이를 위하여 청크 순회 알고리즘을 고안하였다. 첫 번째 청크의 시작 주소는 객체저장소의 첫 번째 NVM페이지 주소에서 메타데이터(struct malloc_state)가 차지하는 공간을 더한 주소이다. 이 주

소에 청크의 크기를 더하면 다음 청크의 주소를 알 수 있다. 이 방법을 통해 하나의 세그먼트(NVM 페이지의 집합)내의 청크들은 모두 순회가 가능하다. 청크를 전부 순회하여 자유 청크가 아닌 청크(할당 청크)가 프라임 청크(루트 노드)부터 자료구조 검색 알고리즘을 통해 검색하였을 때 검색이 되지 않는다면 가비지로 판단한다. 객체저장소 메타데이터에 어떤 자료구조로 객체저장소가 구성되어 있는지 알 수 있으므로 검색 알고리즘을 선택하여 검색이 가능하다. 본 논문에서는 검색 속도를 빠르게 하기 위해 미리 사용자 자료구조를 순회하여 주소로 정렬된 리스트(Allocation List)를 관리한다. 청크 구조 또한 주소로 정렬되어 있으므로 각 노드의 1:1 비교를 통해 여러 번의 검색 없이 한 번의 순회만으로 가비지 검출이 가능하다.

2.3. 가비지 수집 정책

객체저장소 단위의 GC는 Full GC의 오버헤드를 분산시키는 것을 목적으로 한다. 본 연구에서는 빠른 공간 창출을 위한 Local GC와 전체 객체저장소를 점증적으로 GC하는 Separated Full GC로 구분하였다.

2.3.1 Local Garbage Collection

Local GC(LGC)는 사용자가 객체저장소에 NVM 청크 할당을 요청했는데, 자유 청크가 없을 경우 수행되는 Foreground GC이다. 하나의 객체저장소에 대해서만 동작하며 현재 사용하는 NVM 세그먼트에서 모든 자유 청크를 다 뒤지고, 청크들을 합병시켰는데도 요청한 공간 확보가 안될 경우 LGC가 수행되어 가비지를 수집한다. 이후 GC를 통해 확보된 공간으로 청크 할당이 가능하면 청크의 시작 주소를 사용자에게 제공하고, 청크 할당이 불가능하면 커널에 새 NVM 페이지 할당을 요청한다.

2.3.2 Separated Full Garbage Collection

Separated Full GC(SFGC)는 전체 객체저장소를 대상으로 하는 GC(Full GC)이다. 하지만 모든 객체저장소에 대해 GC를 수행하는 것은 오버헤드가 크고, 객체저장소가 여러 프로세스에서 동시에 매핑되서 사용되는 경우 락을 획득하기 위한 경쟁이 일어나기 때문에 이 오버헤드를 완화할 수 있는 GC 기법이 필요하다. 본 논문에서는 이를 위하여 Full GC를 분할하여 백그라운드로 수행할 수 있는 기법을 설계하였다. SFGC는 전체 객체저장소를 몇 개의 객체저장소 단위로 쪼개어 GC를 수행하는 기법으로서 5초 주기의 백그라운드로 동작하는 유저레벨의 데몬으로 동작하게 된다. SFGC의 대상이 되는 객체저장소는 어떠한 프로세스에서도 사상되어 사용되지 않는 것으로만 한정하였다. 하나의 객체저장소가 여러 프로세스에 사상되어 사용될 수 있는 환경이기 때문에 SFGC데몬이 다른 프로세스에 사상되어 있는 객체저장소에 대해 가비지 수집을 시도할 경우 락 오버헤드가 생기게 되고, 다른 프로세스에 사상되어 있는 객체 저장소는 LGC에 의해 가비지 수집이 될 확률이 높기 때문이다. 본 연구에서는 이를 위해 사상되지 않은 객체들의 리스트를 유지하고, 사상해제시 이 리스트에 해제된 객체저장소를 삽입함으로서 SFGC를 지원할 수 있도록 하였다.

3. 실 험

표 1. 실험 환경

processor	Intel(R) i7-3770 (Freq. 3.40GHz)
Main memory	DDR3-SDRAM 6GB
Storage	HDD 1TB
Linux kernel version	2.6.32

성능 측정은 표 1의 사양을 가진 PC에 가비지 수집 기법을 적용한 HEAPO 커널과 라이브러리를 설치하여 수행하였다.

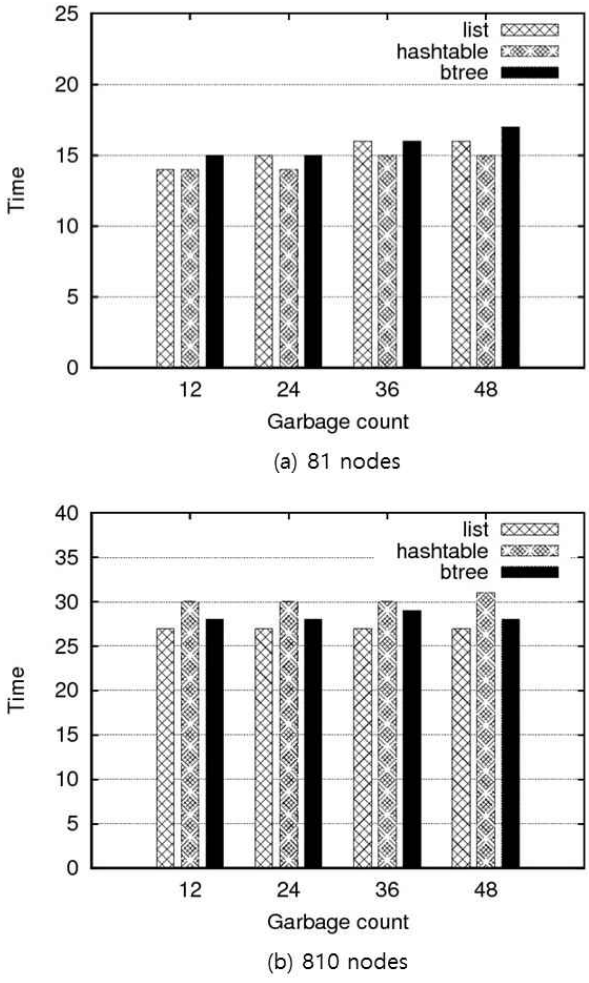

그림 2 . 자료구조 별 Local GC 수집 시간

그림 2는 각각 81개 노드가 입력된 자료구조(a)와 810개의 노드가 입력된 자료구조(b)에서 가비지의 개수를 12개부터 48개까지 늘려가면서 가비지 수집 시간을 측정한 그래프이다. 그림 2.(b)가 그림2.(a)에 비해 입력된 노드가 10배 늘어났지만 가비지 검출 시간은 약 2배 정도 증가했음을 볼 수 있다.

그림 3은 810개 노드가 입력된 자료구조에서 가비지의 개수를 12개부터 480개까지 12개 단위로 늘려가면서 가비지 수집 시간을 측정한 그래프이다. 가비지의 수가 계속 증가하여도 가비지 수집 시간은 일정함을 볼 수 있다. 따라서 본 가비지 검색 기법은 가비지의 수에 크게 영향을 받지 않고 생성된 청크의 수(노드의 수)에 비례하여 증가함을 볼 수 있다. 이는 본 가비지 탐색 기법의 특성상 전체 청크를 순회하여야하기 때문에 적절한 결과가 나왔다고 할 수 있다.

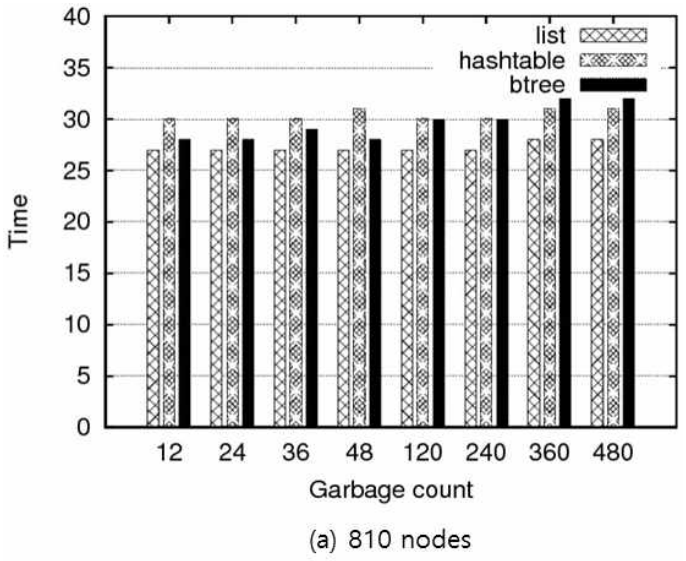

그림 3. 가비지 수의 변화에 따른 Local GC 수행 시간

그림 4는 SFGC의 1회 수행 시간을 측정한 그래프이다. Linked List 자료구조를 구성하는 20개의 객체저장소를 생성하고, 각각 100개의 노드를 입력한 후 가비지의 수를 변화시키면서 가비지 수집 시간을 측정하였다. SFGC역시 가비지 수에 따른 수행 시간이 별 차이나지 않는 것을 볼 수 있다.

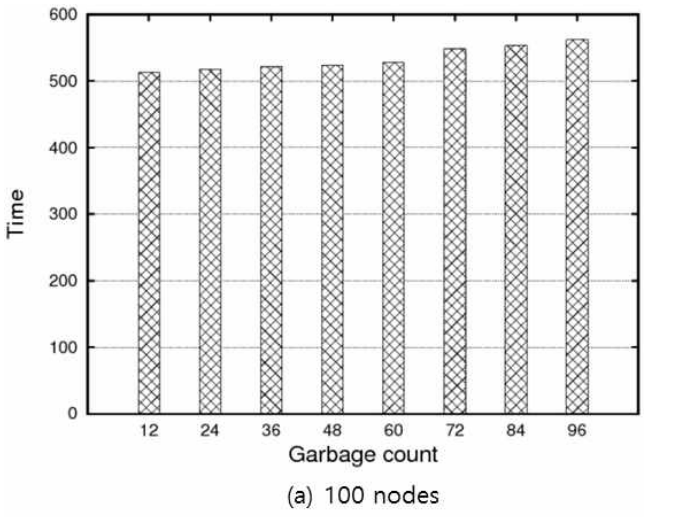

그림 4. SFGC 수행 시간

4. 결 론

본 논문에서는 NVM기반 환경에서 다중 타입의 자료구조를 위한 가비지 수집 기법을 개발하였고 이를 영속 힙 기반의 HEAPO 플랫폼에 구현하여 검증하였다. 객체 저장소에 존재하는 여러 청크들의 가비지 여부를 판단하기 위해 각각의 청크들을 사용자 자료구조 순회 알고리즘으로 검색하는 대신, 1번의 순회를 통해 모든 가비지를 검출 할 수 있는 탐색 기법을 개발하였고, 가비지 수집의 오버헤드를 분산하기 위한 2가지 가비지 수집 정책을 제안하였다. 본 연구진은 이를 실제 리눅스 PC에 구현하여 검증하였으며 실험 결과를 통해 적절하게 설계되었음을 증명하였다.

5. 사 사

이 논문은 2016년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No. R7117-16-0232, 32Gbps 데이터 서비스를 위한 익스트림 스토리지 입출력 기술 개발)

참 고 문 헌

[1] J. Coburn, A. M. Caulfield, A. Akel, L. M. Grupp, R. K. Gupta, R. Jhala, and S. Swanson. 2011. “Nv-heaps: Making persistent objects fast and safewith next-generation, non-volatile memories,“ Proc. of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS’11), 2011.

[2] H. Volos, A. J. Tack, and M. M. Swift. “Mnemo syne: Lightweight persistent memory,“ Proc. of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS’11), 2011.

[3] J. Guerra, L. Marmol, D. Campello, C. Crespo, R. Rangaswami, and J. Wei. 2012. Software persistent memory. Proc. of USENIX Annual Technical Conference (ATC) , 2012.

[4] Taeho Hwang, Jaemin Jung, and Youjip Won, "HEAPO: Heap-based Persistent Object Store", ACM Transactions on Storage, Vol. 11, Issue 1, Dec. 2014.

[5] Lea, Doug, and Wolfram Gloger. "A memory allo cator." (1996).