플래시 메모리 환경을 위한 컨테이너 기반 레코드 관리 방법

배덕호○* 김상욱* 장지웅**
한양대학교 전자컴퓨터통신공학부*, 한국산업기술대학교 게임공학과**
smith@zion.hanyang.ac.kr○, wook@hanyang.ac.kr, jwchang@kpu.ac.kr

Container-Based Record Management for Flash Memory Environment

Duck-Ho Bae○* Sang-Wook Kim* Ji-Woong Chang**
Dept. of Electronics and Computer Engineering, Hanyang University*
Dept. of Game & Multimedia Engineering, Korea Polytechnic University**

I. 서론 플래시 메모리는 기존의 저장 매체들과는 다른 고유한 특성을 가진다. 첫째, 플래시 메모리는 덮어쓰기 연산(overwrite operation)이 제한적이다[3]. 둘째, 플래시 메모리에는 오버헤드가 매우 큰 소거 연산이 존재한다. 덮어쓰기 연산을 수행하지 못해 생성된 무효화된 페이지(이하 무효 페이지라고 함)를 다시 사용하기 위해서는 해당 페이지가 포함된 블록에 대한 소거 연산을 먼저 수행하여야 한다[2]. 따라서 플래시 메모리에서는 덮어쓰기 연산을 최대한 수행하여 소거 연산의 횟수를 감소시키는 것이 매우 중요하다. 본 논문에서는 플래시 메모리 환경에서 기존의 레코드 관리 방법의 문제점을 분석하며, 이를 바탕으로 플래시 메모리 환경을 위한 컨테이너 기반 레코드 관리 방법을 제안한다.

II. 플래시 메모리 환경에서 기존의 레코드 관리 방법의 고찰 기존의 레코드 관리 방법은 페이지를 동일한 크기의 논리적 단위인 슬롯으로 나누어 하나의 슬롯에 하나의 레코드를 저장한다[1]. 하나의 슬롯은 레코드가 저장될 공간과 해당 슬롯의 사용 여부를 나타내는 상태 비트로 구성된다. 슬롯에 레코드가 저장되어 있다면, 상태 비트는 0으로 설정되며, 슬롯이 비어있다면, 상태 비트는 1로 설정된다.

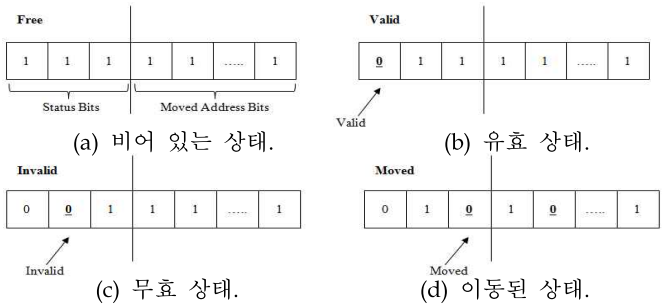
기존의 레코드 관리 방법은 레코드가 저장되었던 슬롯에 다시 레코드를 저장하는 문제점이 존재한다. 첫째, 새로운 레코드 삽입 시, 레코드가 삭제되었던 슬롯에 다시 레코드를 저장하는 경우가 발생할 수 있다. 레코드가 삭제되었던 슬롯은 기존에 저장된 레코드가 그대로 기록되어 있으며, 슬롯의 상태 비트만 1로 설정되어 있다. 따라서 해당 슬롯에 레코드를 다시 저장할 경우, 플래시 메모리 소자를 1에서 0으로만 변경하여 새로운 레코드를 저장할 수 있는 경우는 매우 드물다. 둘째, 레코드 수정 시, 기존의 슬롯에 갱신된 내용을 반영한다. 위와 마찬가지로, 기존의 레코드를 1에서 0으로만 변경하여 수정된 레코드를 저장할 수 있는 경우는 매우 드물다. 이와 같이, 기존의 레코드 관리 방법은 덮어쓰기 연산을 거의 활용하지 못하고, 새로운 페이지를 할당받는 경우가 대부분이다.

III. 컨테이너 기반 레코드 관리 방법 본 논문에서 제안하는 새로운 레코드 관리 방법의 기본 전략은 다음과 같다. 첫째, 레코드가 저장되었던 슬롯에는 다시 레코드를 저장하지 않는다. 이 경우, 플래시 메모리의 특성 상 덮어쓰기 연산을 거의 활용할 수 없다. 따라서 제안하는 레코드 관리 방법은 비어있는 새로운 슬롯에 레코드를 저장한다. 둘째, 이를 위해 슬롯의 사용 여부가 아닌 레코드의 상태에 따라 슬롯의 상태를 설정한다. 제안하는 레코드 관리 방법은 레코드가 기록되었던 슬롯을 무효화하여, 해당 슬롯에는 더 이상 새로운 레코드를 저장하지 않는 전략을 사용한다. 더 나아가, 레코드 수정 시, 기존의 레코드를 무효화하고, 비어있는 슬롯에 수정된 레코드를 저장한 후, 기존의 레코드가 수정된 레코드를 가리키는 방법을 사용한다. 이를 통해 덮어쓰기 연산을 효율적으로 수행할 수 있으며, 생성되는 무효 페이지의 수를 크게 줄일 수 있다.

본 논문에서 제안하는 컨테이너의 구조는 다음과 같다. 컨테이너는 하나의 페이지를 동일한 크기로 나눈 논리적인 구조이며, 하나의 컨테이너에는 하나의 레코드가 저장된다. 컨테이너는 레코드를 저장하는 데이터 부분과 컨테이너에 저장된 레코드를 관리하기 위한 컨테이너 메타데이터 부분으로 구성된다. 컨테이너 메타데이터는 해당 컨테이너 안에 저장된 레코드의 상태를 나타내는 상태 비트(status bits)와 수정이 발생하였을 경우 수정된 레코드가 저장된 컨테이너의 위치를 표시하기 위한 이동된 주소 비트(moved address bits)로 구성된다.

컨테이너의 상태 비트는 총 3비트로 구성되며, 컨테이너의 상태는 레코드가 저장되지 않은 비어있는 상태(free status), 유효한 레코드가 저장된 유효 상태(valid status), 삭제에 의해 무효한 레코드가 저장된 무효 상태(invalid status), 그리고 수정에 의해 해당 페이지의 다른 컨테이너로 이동된 상태(moved status) 등 총 4가지의 상태가 존재한다. 제안하는 컨테이너 기반 레코드 관리 방법은 플래시 메모리 소자를 1에서 0으로의 변경만으로 모든 컨테이너의 상태를 표시할 수 있다.

그림 1은 컨테이너의 상태 변화에 따른 메타데이터 부분의 변화를 나타낸다. 그림 1(a)에서 모든 비트가 1로 설정되어 있으면, 비어 있는 상태를 나타낸다. 그림 1(b)에서 해당 컨테이너에 새로운 레코드가 삽입되어 유효 상태가 되면, 첫 번째 비트를 0으로 설정한다. 그림 1(c)에서 레코드가 삭제되어 무효 상태가 되면, 두 번째 비트를 0으로 설정한다. 그림 1(d)에서 유효한 레코드가 수정되면, 세 번째 비트를 0으로 설정하고 이동된 주소 비트에 수정된 레코드가 저장된 컨테이너의 위치를 기록한다.



(a) 비어 있는 상태. (b) 유효 상태. (c) 무효 상태. (d) 이동된 상태.

그림 1. 컨테이너의 상태 변화에 따른 메타데이터의 변화.

이동된 주소 비트는 수정 전 레코드가 저장된 컨테이너에 수정된 레코드가 저장된 컨테이너의 위치를 표시하기 위해 사용된다. 이동된 주소 비트는 하나의 페이지 안에 존재하는 컨테이너를 표시할 수 있을 만큼의 비트로 구성된다. 식 1은 이동된 주소 비트의 개수를 구하는 식이다. 예를 들어, 하나의 페이지가 8개의 컨테이너로 구성되어 있을 경우, 이동된 주소 비트는 3비트가 된다. $numofmovedaddressbits = \lceil \log_2^{numofcontainer} \rceil$ (식 1)

제안하는 컨테이너 기반 레코드 관리 방법의 레코드 삽입, 삭제, 수정 절차는 다음과 같다. 새로운 레코드 삽입 시, 비어있는 컨테이너에 레코드를 저장한 후, 해당 컨테이너의 상태를 유효 상태로 설정한다. 레코드 삭제 시, 삭제할 레코드가 저장된 컨테이너의 상태를 무효 상태로 설정한다. 레코드 수정 시, 레코드를 수정할 페이지의 비어있는 컨테이너 C_2에 수정된 레코드를 저장하고, 컨테이너 C_2의 상태를 유효 상태로 설정한다. 그리고 수정 전 레코드가 저장된 컨테이너 C_1의 상태를 이동된 상태로 설정하고, 컨테이너 C_1의 이동된 주소에 컨테이너 C_2의 위치를 표시한다.

제안하는 컨테이너 기반 레코드 관리 방법은 해당 페이지에 빈 공간이 존재함에도 불구하고, 무효 상태의 컨테이너들이 많이 존재하여 새로운 레코드나 수정된 레코드를 삽입하지 못하는 경우가 발생한다. 이를 해결하기 위해 레코드를 삽입하지 못할 경우, 해당 페이지의 모든 무효 컨테이너들을 비어있는 상태로 설정하고, 해당 컨테이너들의 데이터 부분의 비트를 모두 1로 설정한다. 또한, 이동된 컨테이너와 기존의 컨테이너의 병합을 수행한다. 컨테이너의 병합은 수정된 레코드를 기존의 컨테이너에 복사하여 기존의 컨테이너를 유효 컨테이너로 변경한 후, 수정된 레코드가 저장되어 있던 컨테이너는 비어있는 상태로 설정하고, 해당 컨테이너의 데이터 부분의 비트를 모두 1로 설정한다. 본 논문에서는 이러한 작업을 페이지 압축(page compaction)이라고 부르며, 이를 통해 레코드가 삽입될 컨테이너를 확보할 수 있다.

IV. 성능 비교 실험 본 실험은 크게 두 가지로 구성된다. 실험 1에서는 기존의 레코드 관리 방법과 제안하는 방법의 각 연산의 성능을 측정한다. 실험 2에서는 기존의 레코드 관리 방법과 제안하는 방법의 전반적인 성능을 측정한다.

실험 1. 삽입, 삭제, 수정 연산의 성능 비교 실험 1에서는 기존의 레코드 관리 방법과 제안하는 방법의 각 연산의 성능을 측정한다. 이를 위하여 삽입, 삭제, 수정 연산을 각각 수행하며 성능을 측정하였다. 실험 1의 결과는 그림 2를 통해 알 수 있다. 그래프 x축은 삽입, 삭제, 수정 연산을 나타내고, y축은 측정한 전체 연산의 비용을 나타낸다.

실험 결과, 제안하는 방법의 삭제 연산과 수정 연산의 성능은 기존의 레코드 관리 방법에 비해 우수하였다. 반면에, 삽입 연산의 경우, 기존의 레코드 관리 방법의 성능이 제안하는 방법에 비해 우수하였다. 레코드 삽입 시, 기존의 레코드 관리 방법과 제안하는 방법 모두 덮어쓰기 연산이 수행 가능하다. 그러나 제안하는 방법의 경우, 컨테이너를 관리하기 위한 메타데이터 부분이 추가적으로 필요하여 하나의 페이지에 삽입 가능한 레코드의 개수가 기존의 레코드 관리 방법에 비해 적다. 이로 인한 쓰기 연산의 횟수가 많아져 기존의 레코드 관리 방법에 비해 성능이 낮아졌다.

그림 2. 각 연산의 성능.

실험 2. 전반적인 성능 비교 실험 2에서는 기존의 레코드 관리 방법과 제안하는 방법의 전반적인 성능을 측정한다. 이를 위하여 전체 연산 중 삭제 연산의 비율을 20%로 고정한 후, 나머지 80%의 삽입, 수정 연산 중 삽입 연산의 비율을 20%, 40%, 60%, 80%로 증가시켜가며 성능을 측정하였다. 실험 2의 결과는 그림 3을 통해 알 수 있다. 그림 3은 삽입 연산의 비율을 증가시켜가며 측정한 전체 연산의 비용 변화를 나타낸다. 실험 결과, 제안하는 방법의 성능이 기존의 레코드 관리 방법에 비해 우수하였다. 이는 제안하는 방법이 덮어쓰기 연산을 많이 수행하여, 이로 인해 소거 연산의 수가 크게 줄었기 때문이다.

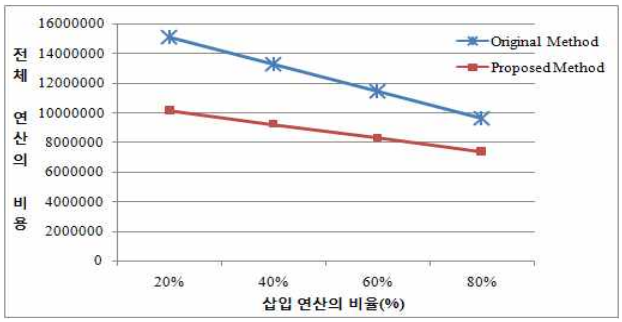


그림 3 삽입 연산의 비율 변화에 대한 실험 결과.

V. 결론 본 논문에서는 플래시 메모리 환경에 적합한 새로운 레코드 관리 방법을 제안하였다. 첫째, 플래시 메모리의 물리적인 특성이 기존의 레코드 관리 방법에 미치는 영향을 분석하고, 플래시 메모리 환경에서 나타나는 기존의 레코드 관리 방법의 문제점을 지적하였다. 둘째, 이를 기반으로 플래시 메모리 환경에 적합한 레코드 저장 구조인 컨테이너 구조를 제안하고, 이를 이용한 새로운 레코드 관리 방법을 제안하였다. 셋째, 실험을 통해 제안하는 방법의 우수성을 규명하였다.

감사의 글 본 연구는 지식경제부 및 정보통신연구진흥원의 대학 IT 연구센터 지원사업(IITA-2008-C1090-0801-0040) 및 2007년도 정부(과학기술부)의 재원으로 한국과학재단(R01-2007-000-11773-0)의 연구비 지원을 받았습니다.

참고 문헌

[1] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann, 1995.

[2] J. Jeong , S. Noh, S. Min and Y. Cho, "A Design and Implementation of Flash Memory Simulator, *Journal of Korean Information Science: C*, Vol. 8, No. 1, pp. 36－45, 2002.

[3] S. Lee and B. Moon, "Design of Flash-Based DBMS: An In-Page Logging Approach," In *Proc. ACM Int'l. Conf. on Management of Data*, ACM SIGMOD, pp. 55-66, 2007.