그룹쓰기: 플래시 메모리 환경에서 효율적인 레코드 관리 방법

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

Group Write: An Effective Method for Record Management in Flash Memory Environment

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

요 약

플래시 메모리 환경에서는 디스크 환경과는 달리 논 클러스터링 방법의 성능이 클러스터링 방법에 비해 우수하다. 그러나 논 클러스터링 방법 역시 플래시 메모리의 특성을 고려하여 설계된 것이 아니므로, 성능 저하 요인이 많이 존재한다. 본 논문에서는 이를 바탕으로 플래시 메모리 환경에서 효율적인 레코드 관리 방법을 제안한다. 제안하는 레코드 관리 방법은 빈 공간이 큰 페이지에 레코드들을 최대한 모아 한 번의 쓰기 연산으로 저장함으로써, 쓰기 연산을 크게 줄일 수 있다. 실험 결과에 의하면, 제안하는 방법은 기존 방법의 성능을 최대 1.8배까지 향상시키는 것으로 나타났다.

I. 서론

플래시 메모리는 기존의 저장 매체들과는 다른 고유한 특성들을 가진다. 첫째, 디스크는 읽기 연산과 쓰기 연산 간의 수행 속도의 차이가 없는 반면, 플래시 메모리는 쓰기 연산의 수행 속도가 읽기 연산에 비해 매우 느리다[1,2]. 둘째, 플래시 메모리에는 수행 속도가 매우 느린 소거 연산이 존재한다. 소거 연산은 잦은 쓰기 연산에 의해 발생한다[4,7]. 이로 인해 플래시 메모리에서는 쓰기 연산을 감소시키는 것이 매우 중요하다.

최근, 플래시 메모리의 용량이 증가함에 따라 플래시 메모리에 직접 데이터를 저장, 관리하는 연구가 필요하게 되었다. 이에 따라 데이터베이스 분야의 연구들이 시작되고 있다[4].

특히, 레코드의 검색, 삽입, 삭제 연산을 결정하는 레코드 관리 방법은 DBMS의 성능에 크게 영향을 미치는 중요한 연구 분야이다. 디스크 기반 DBMS의 대표적인 레코드 관리 방법에는 클러스터링 방법(clustering method)과 논 클러스터링 방법(non-clustering method)이 있으며, 범위 질의의 성능이 우수한 클러스터링 방법이 주로 선호되고 있다[10].

그러나 플래시 메모리 환경에서 클러스터링 방법은 레코드 삽입으로 인한 성능 저하가 매우 크다. 반면에 논 클러스터링 방법은 플래시 메모리의 읽기 연산의 빠른 수행 속도로 인해 느린 범위 질의 성능이 보완되고, 버퍼의 효과로 인해 쓰기 연산과 소거 연산의 수가 줄어들게 된다. 이로 인해 플래시 메모리 환경에서는 논 클러스터링 방법의 성능이 클러스터링 방법에 비해 더욱 우수해진다[9].

그러나 논 클러스터링 방법 역시 플래시 메모리의 특성을 고려하여 설계된 것이 아니므로 성능 저하 요인이 존재한다. 특히, 버퍼 교체 전략으로 인해 동일한 페이지에 반복적으로 쓰기 연산을 수행하는 현상이 발생한다. 또한, 반복적인 레코드의 삽입, 삭제로 인한 빈 공간 리스트의 잦은 변경으로 빈번한 쓰기 연산이 발생한다[9]. 따라서 논 클러스터링 방법 역시 플래시 메모리에 적합하지 않다.

본 논문에서는 이러한 분석을 바탕으로 플래시 메모리 환경을 위한 새로운 레코드 관리 방법을 제안한다. 제안하는 레코드 관리 방법은 쓰기 연산을 최소화하기 위해 레코드의 클러스터링을 유지하는 않는다. 그 대신 빈 공간이 큰 페이지에 레코드들을 최대한 모아 한 번의 쓰기 연산으로 저장한다. 본 논문에서는 이러한 레코드 저장 기법을 그룹쓰기(group write)라 부르며, 효과적인 그룹쓰기를 위한 전용 버퍼를 제안한다. 또한, 빈 공간이 큰 페이지들을 효율적으로 관리하기 위해 빈 공간의 크기가 특정 임계값 이상인 페이지들의 일부만을 관리하는 임계값 리스트(threshold list)를 제안한다.

본 논문에서는 다양한 실험을 통하여 제안하는 방법의 우수성을 규명한다. 실험 결과에 의하면, 제안하는 그룹쓰기 방안은 레코드 관리 방법의 성능을 최대 1.8배 개선시킨다.

본 논문의 구성은 다음과 같다. 제 2장에서는 본 논문에서 제안하는 그룹쓰기 방안에 관하여 상세히 서술한다. 제 3장에서는 성능 평가를 통하여 제안하는 방법의 우수성을 규명한다. 제 4장에서는 논문을 요약하고 결론을 맺는다.

II. 제안하는 방법

2.1 기본 전략

플래시 메모리의 고유한 특성에 적합하도록 설계한 새로운 레코드 관리 방법의 기본 전략은 다음과 같다.

첫째, 키 값에 의한 클러스터링을 고수하지 않는다. 플래시 메모리 환경에서는 범위 질의로 인한 성능의 향상보다는 클러스터링을 유지하기 위한 성능의 저하가 더욱 크다[9]. 따라서 제안하는 레코드 관리 방법은 연속적으로 삽입되는 레코드들을 동일한 페이지에 저장한다.

둘째, 레코드들이 삽입되는 페이지는 메모리 내에서 가능한 많은 레코드들을 삽입한 후 쓰기 연산을 수행한다. 이 경우, 여러 번의 레코드 삽입을 한 번의 쓰기 연산으로 저장할 수 있어 쓰기 연산을 크게 줄일 수 있다.

셋째, 빈 공간을 관리하는 오버헤드를 줄인다. 기존의 빈 공간 리스트는 포인터를 페이지 내에 저장하여, 이를 관리하기 위한 오버헤드가 매우 크다. 특히, 리스트의 시작 부분에서만 페이지들이 추가, 삭제되어, 리스트의 시작 부분에는 빈 공간이 작은 페이지들이 대부분이었다. 이로 인한 리스트의 잦은 변경이 발생하였다[9]. 따라서 제안하는 레코드 관리 방법은 빈 공

간이 큰 페이지를 리스트의 시작 부분에서 관리한다.

이와 같이 제안하는 레코드 관리 방법은 삽입되는 레코드들을 빈 공간이 큰 페이지에 최대한 삽입한 후, 쓰기 연산을 수행한다. 플래시 메모리 환경에서의 레코드 삽입은 이와 같은 방법으로 수행되어야 하며, 본 논문에서는 이러한 방법을 그룹쓰기라 부른다.

본 논문에서는 그룹쓰기를 효과적으로 수행하기 위한 방법을 제안한다. 제안하는 방법은 크게 두 가지 요소로 구성된다. 첫째, 그룹쓰기를 효율적으로 지원하기 위한 그룹쓰기 전용 버퍼이다. 둘째, 그룹쓰기 전용 버퍼에 올라갈 페이지를 선택하는 기준이 되는 임계값 리스트이다. 제 2.2절과 제 2.3절에서는 그룹쓰기 전용 버퍼와 임계값 리스트에 대해 상세히 설명한다.

2.2 그룹쓰기 전용 버퍼

그룹쓰기 전용 버퍼는 레코드 삽입을 위한 전용 버퍼로서, 삽입되는 모든 레코드들은 그룹쓰기 전용 버퍼에 올라온 페이지(이하 그룹쓰기 전용 페이지라고 함)에 저장된다. 그룹쓰기 전용 버퍼는 그룹쓰기 전용 페이지가 가득차서 더 이상 레코드를 삽입할 수 없을 때만 쓰기 연산을 수행한다.

그룹쓰기 전용 버퍼는 플래시 메모리의 쓰기 연산의 단위인 한 페이지의 크기의 버퍼를 추가적으로 할당한 것으로 열린(open) 파일마다 하나씩 존재한다. 그림 1은 제안하는 방법의 레코드 삽입 알고리즘을 나타낸다.

단계 1. 그룹쓰기 전용 페이지에 삽입할 레코드를 저장할 공간이 있다면, 해당 페이지에 레코드를 삽입한다.
단계 2. 레코드를 저장할 수 없다면, 그룹쓰기 전용 페이지를 희생자로 선정하고, 임계값 리스트의 첫 번째 페이지를 그룹쓰기 전용 버퍼에 올린 후, 해당 페이지에 레코드를 삽입한다.
단계 3. 만약 임계값 리스트가 비어있다면, 새로운 페이지를 할당받아 그룹쓰기 전용 버퍼에 올린 후, 해당 페이지에 레코드를 삽입한다.

그림 1. 제안하는 방법의 레코드 삽입 절차.

제안하는 그룹쓰기 전용 버퍼를 통해 쓰기 연산의 수를 크게 줄일 수 있으며 이로 인해 소거 연산의 수도 크게 줄어든다. 본 논문에서는 실험 1을 통해 제안하는 레코드 관리 방법의 성능을 측정하고, 이를 분석한다.

2.3 임계값 리스트

임계값 리스트는 그룹쓰기 전용 페이지가 될 수 있는 후보 페이지들을 관리하는 리스트로서, 특정 임계값 이상의 빈 공간을 가진 페이지들을 관리한다. 또한, 리스트 관리 오버헤드를 줄이기 위해 최대 k개의 엔트리만을 연결 리스트로 연결하여 메모리에서 관리한다. 이때, 그룹쓰기의 효과를 극대화하기 위해 빈 공간이 큰 페이지들을 우선적으로 관리한다.

그림 2는 관리하는 최대 엔트리 개수(k)가 4이고, 임계값이 500인 이중 연결 리스트 구조의 임계값 리스트를 나타낸다. 하나의 엔트리에는 이전 엔트리와 다음 엔트리를 가리키는 포인터, 그리고 관리 중인 페이지의 번호와 해당 페이지에 존재하는 빈 공간의 크기가 저장된다. 이 때, 엔트리들은 빈 공간의 크기를 기준으로 내림차순 정렬한다.

그룹쓰기 전용 버퍼에서 새로운 그룹쓰기 전용 페이지의 요청이 발생하면, 임계값 리스트의 시작 부분에 위치한 페이지를 버퍼로 올린 후, 해당 페이지를 임계값 리스트에서 삭제한다. 만약 임계값 리스트가 비어있다면, 새로운 페이지를 그룹쓰기 전용 버퍼로 올린다. 그림 2에서는 가장 큰 빈 공간이 존재하는 페이지인 페이지 p_7이 그룹쓰기 전용 버퍼에 올라간다. 이로 인해 많은 수의 레코드들의 그룹쓰기가 가능해진다.

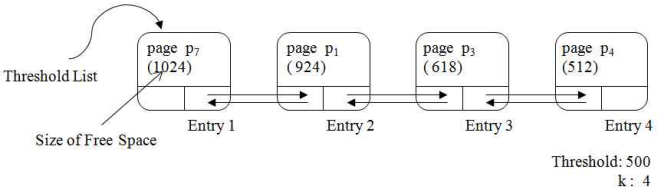


그림 2. 최대 엔트리 개수(k)가 4, 임계값이 500인 임계값 리스트.

그림 3은 제안하는 임계값 리스트에 새로운 엔트리를 추가하는 알고리즘을 나타낸다. 먼저, 페이지 p_i의 빈 공간의 크기가 임계값 이상인 경우, 해당 페이지의 임계값 리스트 추가 여부를 확인한다. 만약, 임계값 리스트에서 현재 관리 중인 엔트리 개수가 최대 엔트리 개수(k)보다 적다면, 페이지 p_i를 임계값 리스트에 추가한다. 이 때, 빈 공간의 크기를 기준으로 내림차순 정렬된 리스트 내의 적절한 위치에 페이지 p_i를 추가한다.

만약, 관리할 수 있는 최대 엔트리 개수(k)만큼의 엔트리를 관리하고 있다면, k번째 엔트리의 빈 공간의 크기와 페이지 p_i의 빈 공간의 크기를 비교하여 빈 공간이 큰 페이지를 임계값 리스트에 추가한다. 이러한 알고리즘을 통해, 빈 공간이 큰 페이지들을 임계값 리스트에서 관리할 수 있다.

Algorithm Insert_Entry (pi, n)
Input:
- p_i: 검색, 삽입, 삭제가 발생한 레코드가 저장된 페이지
- n: 임계값 리스트에서 현재 관리 중인 엔트리 개수

Return:
- *True*: 페이지 pi를 임계값 리스트에 추가
  False: 페이지 pi가 임계값 리스트에 추가되지 않음

```
1:  IF pi의 빈 공간의 크기 > Threshold인 경우 THEN
2:    IF 최대 엔트리 개수(k) > n인 경우 THEN
3:      임계값 리스트에 빈 공간의 크기를 기준으로 내림차순
        정렬된 위치에 페이지 pi를 추가한다.
4:      RETURN True
5:    ELSE
6:      IF k = n인 경우 THEN
7:        IF  k번째 엔트리의 빈 공간의 크기 > pi의 빈 공간의
              크기인 경우 THEN
8:            k번째 엔트리를 삭제한다.
9:            임계값 리스트에 빈 공간의 크기를 기준으로 내림
              차순 정렬된 위치에 페이지 pi를 추가한다.
10:           RETURN True
11:       ELSE RETURN False
12:       END IF
13:     END IF
14:   END IF
15:  ELSE RETURN False
16: END IF
```

Algorithm End

그림 3. 임계값 리스트에 새로운 엔트리를 추가하는 알고리즘.

임계값 리스트는 한정된 메모리의 용량으로 인해 최대 k개의 엔트리만을 관리한다. 이로 인해 페이지 p_i에 임계값 이상의 빈

공간을 존재함에도 불구하고, 더 큰 빈 공간을 가진 페이지들 k개가 이미 리스트에 존재하여, 리스트에 추가되지 못하는 경우가 발생한다.

이러한 문제점을 해결하기 위해 제안하는 임계값 리스트는 빈 공간의 크기의 변동이 일어나지 않는 레코드 검색 시에도 해당 페이지의 리스트 추가 여부를 확인한다[1].

제안하는 레코드 관리 방법은 임계값 리스트의 환경 변수인 임계값과 관리하는 최대 엔트리 개수(k)에 따라 성능이 달라진다. 첫째, 임계값은 쓰기 연산의 횟수와 저장 공간 사용 효율을 조절하는 역할을 한다. 만약 임계값이 낮으면, 임계값 리스트는 빈 공간이 작은 페이지들도 관리한다. 이러한 페이지들은 적은 수의 레코드 삽입만으로도 가득 차게 되어 잦은 쓰기 연산이 발생한다. 이와 반대로 임계값이 높으면, 쓰기 연산의 횟수는 줄어들지만, 리스트가 비어있는 경우가 많아져서 새로운 페이지를 할당받는 경우가 많아진다.

둘째, 임계값 리스트에 유지하는 최대 엔트리 개수(k)는 메모리 사용량과 저장 공간 사용 효율을 조절하는 역할을 한다. 만약 최대 엔트리 개수(k)가 적으면, 임계값 이상의 빈 공간을 가진 페이지가 리스트에 올라가지 못하는 경우가 많이 발생한다. 이로 인해 임계값 리스트가 자주 비게 되어 새로운 페이지를 많이 할당하는 경우가 많아진다.[2]. 이와 반대로 최대 엔트리 개수가 많으면, 메모리 사용량은 늘지만, 엔트리를 많이 유지함으로 인해 새로운 페이지의 할당이 줄어들게 된다. 본 논문에서는 실험 2를 통해 환경 변수의 변화에 따른 제안 방법의 성능 변화를 보인다.

III. 성능 비교 실험

본 장에서는 클러스터링 방법과 논 클러스터링 방법, 그리고 제안하는 방법 간의 성능 비교를 통하여 제안하는 방법의 우수성을 보인다. 아울러 임계값과 최대 엔트리 개수(k) 등 환경 변수의 변화에 따른 제안하는 방법의 성능 변화를 보인다.

3.1 실험 환경

본 실험은 플래시 메모리 개발 프레임 워크의 플래시 메모리 에뮬레이터[3]를 사용한다. 에뮬레이터의 플래시 메모리 용량은 삼성 2G NAND 플래시 메모리[6]를 기준으로 설정한다.

전체적인 실험의 구성은 다음과 같다[3]. 각 실험에서는 20만 개의 레코드를 벌크로드 후, 20만 개의 레코드 검색, 삽입, 삭제 연산을 수행한다. 벌크로드 수행 시, 클러스터링 방법은 각 페이지의 70%만 채우며, 논 클러스터링 방법은 각 페이지의 빈 공간을 남겨두지 않고 가득 채운다. 전체 검색 연산 중 80%는 하나의 레코드 검색 연산을 수행하였으며, 나머지 20%는 범위 질의를 수행한다. 범위 질의는 선택도(selectivity)가 각각 0.01%와 0.05%인 두 종류의 범위 질의를 수행한다. 연산에 사용되는 레코드의 키는 1에서 1000만 사이에서 임의로 생성한다.

실험의 성능 척도로는 모든 연산을 수행한 후, 페이지 읽기, 쓰기, 소거 연산의 총 횟수를 측정한다. 또한, 읽기, 쓰기, 소거 연산의 횟수에 각각의 가중치(읽기: 1, 쓰기: 13, 소거:130)를 곱한 값을 더해 전체 연산의 비용을 계산한다[8]. 이와 더불어, 저장 공간 사용 효율을 확인하기 위해 데이터가 저장된 페이지 수를 측정한다.

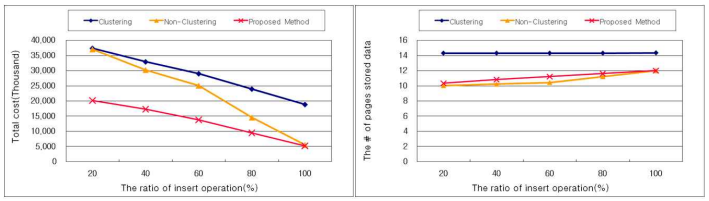
본 실험은 크게 두 가지로 구성된다. 실험 1에서는 플래시 메모리 환경에서 클러스터링 방법과 논 클러스터링 방법, 그리고 제안하는 방법의 성능을 각각 측정한다. 실험 2에서는 여러 가지 환경 변수의 변화에 따른 제안하는 방법의 성능을 측정한다. 이 외에도 한 페이지 안에 존재하는 레코드 개수와 DBMS의 버퍼의 개수를 변경하며 실험하였으나, 전체적인 경향이 유사하므로 본 논문에서는 생략한다.

3.2 실험 결과

실험 1. 클러스터링 방법, 논 클러스터링 방법과 제안하는 방법 간의 성능 비교

본 실험에서는 플래시 메모리 환경에서 클러스터링 방법과 논 클러스터링 방법, 그리고 제안하는 방법의 성능을 측정한다. 이를 위하여 전체 연산 중 검색 연산의 비율을 80%로 고정한 후, 나머지 20%의 삽입, 삭제 연산 중 삽입 연산의 비율을 20%, 40%, 60%, 80%, 100%로 증가시켜가며 성능을 측정하였다. 이 때, 제안하는 방법의 임계값은 30%로, 최대 엔트리 개수(k)는 10으로 고정하였다. 실험 1의 결과는 그림 4를 통해 알 수 있다. 그래프의 x축은 매개 변수인 삽입 연산의 비율의 변화를 나타내고, y축은 측정한 성능 척도를 나타낸다.

그림 4(a)는 삽입 연산의 비율을 증가시켜가며 측정한 전체 연산의 비용 변화를 나타낸다. 실험 결과, 제안하는 방법의 성능은 클러스터링 방법과 논 클러스터링 방법에 비해 우수하였다. 이는 제안하는 방법이 그룹쓰기 전용 버퍼와 임계값 리스트로 인하여 쓰기 연산의 수가 크게 줄어들고, 이로 인해 소거 연산의 수도 크게 줄어들기 때문이다.



(a) 전체 연산의 비용. (b) 저장 공간 사용 효율.

그림 4. 삽입 연산의 비율 변화에 대한 실험 결과.

논 클러스터링 방법과 제안하는 방법 간의 성능의 차이는 삽입 연산의 비율이 증가할수록 줄어든다. 논 클러스터링 방법은 반복적인 레코드의 삽입, 삭제로 인한 빈 공간 리스트 관리 오버헤드가 매우 크다[9]. 삽입 연산의 비율이 증가할수록 수행되는 삭제 연산의 수가 줄어들어, 빈 공간 리스트 관리 오버헤드가 줄어들게 된다. 이로 인해 논 클러스터링 방법의 성능이 향상되어 제안하는 방법 간의 성능의 차이가 줄어들게 된다.

그림 4(b)는 삽입 연산의 비율을 증가시켜가며 측정한 저장 공간 사용 효율의 변화를 나타낸다. 클러스터링 방법은 벌크로드 시 추후 레코드의 삽입을 위한 공간을 확보해두었기 때문에 저장 공간 사용 효율이 제일 낮았다.

제안하는 방법은 논 클러스터링 방법에 비해 저장 공간 사용 효율이 낮았다. 이는 제안하는 방법의 경우, 빈 공간이 있는 페이지가 존재하더라도, 임계값 리스트가 비어있으면 새로운 페이지를 할당 받기 때문이다.

---

1) 특정 페이지에 빈 공간이 많이 존재함에도 불구하고, 해당 페이지에 검색 연산이 발생하지 않아 해당 페이지가 임계값 리스트에 추가되지 않을 수 있다. 그러나 일반적인 경우 검색 연산은 상대적으로 많이 발생하며, 페이지들의 액세스 패턴이 유니폼하기 때문에 위와 같은 일은 거의 발생하지 않는다.

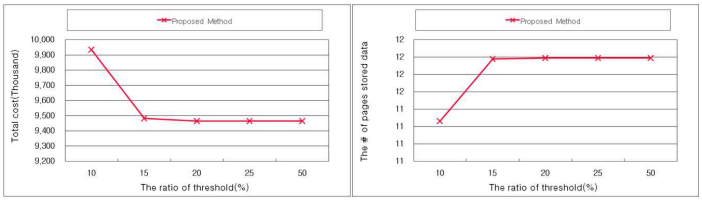
2) 그러나 레코드의 검색이 활발한 시스템에서는 해당 페이지의 임계값 리스트 추가 여부를 자주 확인하므로 적은 수의 엔트리 개수만을 관리하여도 저장 공간 사용 효율을 높일 수 있다. 이는 실험 2-2를 통해 알 수 있다.

3) 본 논문에서는 실험 설계 과정에서 [5]를 참조하였다.

실험 2. 여러 가지 환경 변수의 변화에 따른 제안하는 방법의 성능을 측정

실험 2-1. 임계값의 변화에 따른 제안 방법의 성능 변화

실험 2-1은 임계값의 변화에 따른 제안 방법의 성능 변화를 측정한다. 이를 위하여 임계값을 10%, 15%, 20%, 25%, 30%로 증가시켜가며 성능을 측정하였다. 이 때, 전체 연산 중 검색 연산의 비율을 80%로 고정한 후, 나머지 20%의 삽입, 삭제 연산 중 삽입 연산의 비율을 80%로, 최대 엔트리 개수(k)는 10으로 고정하였다. 실험 2-1의 결과는 그림 5를 통해 알 수 있다. 그래프의 x축은 매개 변수인 임계값의 변화를 나타내고, y축은 측정한 성능 척도를 나타낸다.



(a) 전체 연산의 비용. (b) 저장 공간 사용 효율.

그림 5. 임계값의 변화에 대한 실험 결과.

그림 5(a)는 전체 연산의 비용을 측정한 결과를 나타내며, 그림 5(b)는 데이터가 저장된 페이지 수를 나타낸다. 임계값이 높게 설정될수록, 쓰기 연산의 수가 감소하여 전체 연산의 비용이 감소하게 되는 반면, 새로운 페이지를 할당받는 경우가 많아져 저장 공간 사용 효율은 낮아졌다. 본 실험에서는 임계값이 30%부터는 전체 연산의 비용과 데이터가 저장된 페이지 수에 변화가 없었다. 이는 30% 이상의 빈 공간을 가지는 페이지가 존재하지 않아 대부분의 경우 새로운 페이지를 할당받기 때문이다.

또한, 임계값의 10%와 15% 사이에서 급격한 성능의 변화가 있었다. 임계값이 10%인 경우, 빈 공간이 거의 존재하지 않는 페이지에 레코드를 삽입하는 경우가 많아 이로 인한 성능 저하가 매우 컸다. 그러나 본 실험에서는 15% 이상의 빈 공간을 가지는 페이지가 거의 존재하지 않아, 15% 이상부터는 성능의 변화가 거의 없었다.

실험 2-2. 최대 엔트리 개수의 변화에 따른 제안 방법의 성능 변화

실험 2-2는 최대 엔트리 개수의 변화에 따른 제안 방법의 성능 변화를 측정한다. 이를 위하여 최대 엔트리 개수를 5, 10, 15, 20으로 증가시켜가며 성능을 측정하였다. 이 때, 전체 연산 중 검색 연산의 비율을 80%로 고정한 후, 나머지 20%의 삽입, 삭제 연산 중 삽입 연산의 비율을 80%로, 임계값은 30%로 고정하였다. 실험 2-2의 결과는 표 1을 통해 알 수 있다.

실험 결과, 최대 엔트리 개수의 변화에 따른 제안하는 방법의 성능 변화가 없었다. 이는 레코드의 검색이 활발하여 임계값 리스트에 페이지들의 추가 여부를 자주 확인하기 때문이다. 이렇듯, 적은 수의 엔트리만을 관리하더라도, 많은 수의 엔트리를 관리하는 것과 유사한 성능을 나타낼 수 있다.

표 1. 최대 엔트리 개수의 변화에 대한 실험 결과

k	5	10	15	20
Total Cost	9,463,176	9,463,176	9,463,176	9,463,176
Page Utilization	11,597	11,597	11,597	11,597

IV. 결론

플래시 메모리는 읽기, 쓰기, 소거 연산의 수행 속도의 차이가 크다. 특히, 쓰기 연산은 비용이 가장 큰 소거 연산을 유발하고, 플래시 메모리의 수명을 단축시키는 요인이다. 그러므로 플래시 메모리에서는 쓰기 연산의 수를 감소시키는 것이 중요하다. 본 논문에서는 이러한 특성을 기반으로 플래시 메모리 환경에 적합한 새로운 레코드 관리 방법을 제안하였다.

본 논문의 공헌은 다음과 같다. 첫째, 플래시 메모리 환경을 위한 효율적인 레코드 삽입 방법인 그룹쓰기를 제안하였다. 둘째, 그룹쓰기를 효과적으로 수행하기 위한 그룹쓰기 전용 버퍼와 임계값 리스트를 제안하였다. 셋째, 다양한 실험을 통하여 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] E. Gal and S. Toledo, "Algorithms and Data Structures for Flash Memories," *ACM Computing Surverys*, Vol. 37, No. 2, pp. 138-163, 2005.

[2] A. Kawaguchi, S. Nishioka, and H. Motoda, "A Flash-Memory Based File System," In *Proc. USENIX Technical Conf. on Unix and Advanced Computing Systems*, pp. 155-164, 1995.

[3] S. Kim et al., "A Development Framework for Reliable Flash Memory Software," *SK Telecommunications Review*, Vol. 15, No. 4, pp. 638-646, 2005.

[4] 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.

[5] J. Rao and K. Ross, "Making B+-Trees Cache Conscious in Main Memory," In *Proc. ACM Int'l. Conf. on Management of Data*, ACM SIGMOD, pp. 475-486, 2000.

[6] Samsung, 2G NAND Flash Memory, http://www.samsung.com/products/semiconductor/NANDFlash/, 2008.

[7] C. Wu, L. Chang, and T. Kuo, "An Efficient B-Tree Layer for Flash-Memory Storage Systems," In *Proc. Int'l. Conf. on Real-Time and Embedded Computing Systems and Applications*, RTCSA, Vol. LNCS 2968, pp. 409-430, 2003.

[8] K. Yim, "A Novel Memory Hierarchy for Flash Memory Based Storage Systems," *Journal of Semiconductor Technology and Science*, Vol. 5, No. 4, pp. 262-269, 2005.

[9] D. Bae, J. Chang, and S. Kim, "Performance Analysis of Clustering and Non-clustering in Flash Memory Environment," *The Korean Institute of Information Scinetists and Engineers*, KIISE, Vol. 5, No. 4, pp. 25-26, 2007.

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