데이터 압축 관리 시스템의 데이터베이스 재구성 기법*

이정화°, 황진호, 이승미, 손진현
한양대학교 컴퓨터공학과
jwlee°@database.hanyang.ac.kr, jhhwang@cse.hanyang.ac.kr,
smlee@database.hanyang.ac.kr, jhson@hanyang.ac.kr

The Data Compaction Mechanism in Compressed Data Management System

Jeongwha Lee, Jinho Hwang, Seung Mi Lee, Jin Hyun Son
Department of Computer Science and Engineering, Hanyang University in Ansan

요 약

기존의 연구 과제에서 모바일 기기에 쓰이는 데이터 압축 관리 시스템인 CDMS(Compressed Data Management System)을 제안하였다. 그러나 CDMS의 DB파일에서 Free Page의 발생, 또는 데이터가 늘어났다가 줄어들었음에도 불구하고 실질적인 파일크기가 줄어들지 않는 문제점이 발견되었다. 따라서 데이터들을 Compact하게 배치하여 데이터베이스를 재구성한 후 사용되어지지 않는 공간을 반환하여 다른 모듈에서 활용할 수 있도록 하기 위하여 Compaction Mechanism을 제안하였다.

1. 서론

모바일 정보기기에서는 일반적으로 저 전력으로 장시간의 구동이 가능하며, 부피가 작고 경량으로 소음과 진동이 없으며 물리적인 충격에 강해 휴대가 용이한 플래시 메모리를 보조기억장치로 사용한다 [1]. 하지만 플래시 메모리는 기존의 하드디스크에 비해 고비용과 데이터 I/O에 따른 수명을 가진다는 단점이 있다. 그래서 데이터의 효율적인 관리가 필요하게 되었다 [2, 3]. 이러한 필요로 인해 효율적인 데이터 압축 관리 시스템인 CDMS(Compressed Data Management System)가 제안되었다 [4, 5]. 하지만 이 관리 시스템(CDMS)은 DBMS가 관리하는 데이터의 효율적인 처리 및 관리를 제공 하면서 Free Page의 발생 및 데이터가 늘어났다가 줄어들었음에도 불구하고 실질적인 데이터베이스 파일 크기가 줄어들지 않는 문제점을 가지고 있었다. 따라서 본 논문에서는 이러한 CDMS의 문제점을 확인하고 그것에 대한 해결방법인 Compaction Mechanism을 제시한다.

본 논문의 구성은 다음과 같다. 2장에서는 CDMS에 대한 연구를 살펴본다. 3장에서는 효율적인 압축 데이터 관리 시스템인 CDMS의 문제점을 통해 본 연구의 필요성을 설명하며 4장에서는 CDMS의 문제점을 해결할 수 있는 Compaction Mechanism에 대해서 설명하고 5장에서는 제안하는 Compaction Mechanism을 실험하여 평가한다. 마지막으로 6장에서는 논문의 결론을 맺는다.

* 이 논문은 2007년도 정부(과학기술부)의 재원으로 한국과학재단의 지원을 받아 수행된 연구임(No. R01-2007-000-20135-0).

2. CDMS 구조 및 관리기법

이 장에서는 CDMS의 전체적인 구조 및 관리기법을 알아본다. 2.1절에서는 CDMS의 전체적인 구조 [4, 5]를 기술하고, 2.2절에서는 Dynamic 관리기법 [5]을 기술한다. 그리고 마지막으로 2.3절에서 Static 관리기법 [4]을 기술한다.

2.1 CDMS 구조

CDMS는 기존 DBMS의 변경 최소화를 위해 DBMS의 스토리지 관리자와 파일 시스템 사이에 들어가게 된다.

CDMS의 전체적인 구조도인 아래 그림 1에 나타나 있는 바와 같이 CDMS는 DBMS가 요청한 데이터를 압축된 데이터 파일에서 찾기 위해 데이터파일 내부에 맵핑정보(Mapping Information)를 포함시키고 있다. 이 맵핑정보는 논리적 블록번호, 압축된 블록의 위치나 크기 정보를 가지게 되며, 이러한 맵핑 정보를 이용해 압

축되지 않은 상태의 데이터파일과 압축된 데이터파일의 맵핑이 가능해진다. 이때 맵핑정보에 포함된 블록번호는 압축되지 않은 상태의 데이터파일을 논리적 블록의 크기(페이지)로 분할했을 때의 할당된 번호이다.

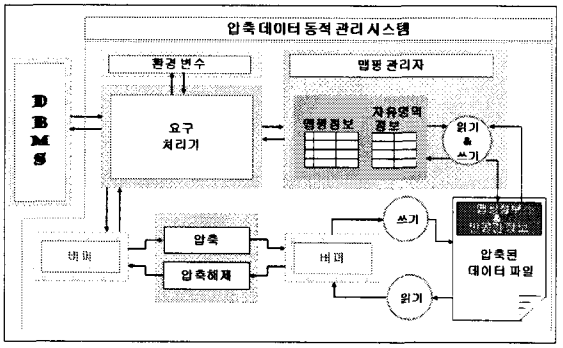

그림 1. CDMS 구조

또한 CDMS는 데이터파일 내부에 자유영역정보(Free Space Information)를 포함시키고 있다. 이 자유영역정보는 데이터의 무결성을 보장하기 위한 처리로 인해 발생한다. 구체적으로 설명하면 압축된 블록에 포함된 데이터가 DBMS에 의해 갱신되어 다시 쓰기를 수행할 때 CDMS에 의해 해당 데이터가 포함된 블록은 다시 압축되어 데이터 파일 내에 저장되어 진다. 이때 다시 압축된 블록은 갱신되기 전 압축된 상태의 블록크기와 동일한 크기를 가진다는 것을 보장 할 수 없다. 이러한 이유로 압축된 블록이 갱신되어 다시기록 되어야 할 때는 기존위치가 아닌 파일 내부의 새로운 공간에 저장되어야 하고, 압축된 블록이 존재하던 기존위치는 새로운 압축 블록의 쓰기를 위해 저장 가능한 공간(Free Space)으로 표시되어야한다. 이렇게 파일내부에 발생하는 저장가능공간의 정보를 자유영역정보로 가지고 있게 된다.

기본적인 모바일 DBMS의 데이터 읽기 쓰기 요청에 대한 CDMS의 처리절차는 표 1.1, 1.2와 같다.

표 1.1 CDMS를 사용한 DBMS의 데이터 읽기 요청 처리 절차

DBMS의 데이터읽기 요청
1) 요청된 데이터를 포함하는 압축된 블록의 위치 검색
2) 검색된 위치의 블록 읽기
3) 읽어진 블록의 압축해제
4) 압축해제 된 데이터를 DBMS에 반환

표 1.2 CDMS를 사용한 DBMS의 데이터 쓰기 요청 처리 절차

DBMS의 데이터쓰기 요청
1) 요청된 데이터 압축
2) 저장 가능한 자유영역 검색
3) 검색된 자유영역에 압축된 블록 쓰기
4) 사용된 자유영역 삭제
5) 맵핑정보 갱신
6) 자유영역정보 추가(데이터 갱신시)

표 1.1과 1.2 에서 나타난 처리절차는 기본적인 처리절차이다. 이 처리절차는 세부적으로 각 관리기법에 따른 절차를 따르게 된다.

2.2 Dynamic 관리기법

CDMS의 압축 데이터 관리기법 중 하나인 Dynamic 관리기법은 압축된 데이터의 크기에 따라 처리하는 방식이다. 이 Dynamic 관리기법은 세부적으로 2가지로 다시 나눠진다. 먼저 첫 번째로 압축된 블록이 나눠지지 않고 저장되는 Dynamic Continuous Writing 방식이고, 두 번째로 빈 공간이 압축된 블록보다 작더라도 앞쪽의 빈 공간부터 채우고 나머지는 또 다른 빈 공간에 넣어서 나눠진 블록을 링크로 연결하는 Dynamic Minimizing Data File Space 방식이다.

(1) Dynamic Continuous Writing 방식

Dynamic Continuous Writing 방식은 CDMS가 데이터 쓰기의 요청을 받았을 때, 압축된 데이터를 저장하기 위한 공간으로 블록 전체가 모두 들어갈 수 있는 크기의 공간을 선택 한다. 그리고 그 선택된 공간에 데이터를 쓰게 된다.

(2) Dynamic Minimizing Data File Space 방식

Dynamic Minimizing Data File Space 방식은 압축된 데이터들에 맞는 공간만을 찾다가 빈 공간이 많이 생기는 것을 제거하고자 압축된 데이터를 나눠서 앞쪽의 빈 공간부터 채워 가는 방식이다. 연속된 공간에 써야 할 데이터가 처음에 연속된 공간을 만난다면 그곳에 연속적으로 쓸 수 있지만 그렇지 않다면 먼저 빈공간의 크기만큼 쓰고 나머지는 다음 빈 공간에 쓰게 된다. 이때 나눠지는 데이터는 flag로써 나눠진 블록에 대한 정보를 자체적으로 보유하게 된다.

2.3 Static 관리기법

CDMS의 다른 하나의 관리기법은 Static 관리기법이다. 이 Static 관리기법은 각각 다른 크기의 빈공간이 무분별하게 생기는 것을 막기 위한 방법으로 가상의 슬롯을 이용하고 있다. 이러한 가상의 슬롯을 이용함으로 데이터 파일을 논리적으로 고정된 슬롯 크기로 나누어 압축된 데이터를 저장하게 된다.

Static 기법 또한 세부적으로 2가지로 나눠진다. 먼저

첫 번째로 압축된 블록이 떨어져서 저장되지 않고, 연속되는 슬롯에 저장되는 Static Continuous Writing 방식이고 두 번째로 빈 슬롯이 압축된 블록보다 작더라도 앞쪽의 빈 슬롯부터 채우는 Static Minimizing Data File Space 방식이다.

(1) Static Continuous Writing 방식

Static Continuous Writing 방식은 CDMS가 데이터 쓰기의 요청을 받았을 때 그 데이터의 압축된 블록이 슬롯의 크기보다 큰 경우 두 개 이상의 슬롯을 사용하게 되는데 이 때 무조건 연속되는 슬롯을 찾아 쓰게 된다. 연속되는 슬롯을 찾을 때는 파일의 제일 앞쪽부터 찾게 된다.

(2) Static Minimizing Data File Space 방식

Static Minimizing Data File Space 방식은 Static Continuous Writing 방식과 다르게 데이터 쓰기 요청을 받았을 때 앞쪽부터 빈 공간을 차례로 채워가는 방식이다. 연속된 공간에 써야 할 데이터가 연속된 공간을 만난다면 그곳에 연속적으로 쓸 수 있지만 그렇지 않다면 먼저 빈공간의 크기만큼 쓰고 나머지는 다음 빈 공간에 쓰게 된다. 이 때 나눠진 데이터는 링크정보로 연결하게 된다.

3. CDMS Mechanism의 필요성

현재 CDMS의 DB 파일에서는 데이터가 늘어났다가 줄어들 경우 실질적인 파일 크기는 줄어들지 않는다. 물론 데이터가 줄어들어 뒷부분에 빈공간이 생겼을 경우 이 공간은 후에 재사용 되어질 수 있다. 그러나 Mobile device라는 특수 상황에서, 저장 공간을 좀더 효율적으로 사용하기 위해 DB의 데이터가 줄어들 경우 DB파일의 크기를 줄여서 다른 부분에서 그 공간을 사용할 수 있도록 하기 위한 Compaction의 필요성이 대두되었다. 또한 Free Page가 존재하여서 의미없는 데이터 임에도 불구하고 CDMS DB파일 내에서 공간을 차지하고 있는 데이터가 존재하였다. 그래서 CDMS에 compaction을 적용하여 데이터베이스를 재구성함으로써 Free Page 처리 및 Free Space를 삭제하여 데이터들을 Compact하게 배치한 후 사용되어지지 않는 공간을 반환하여 다른 모듈에서 활용할 수 있게끔 하였다 [6].

DBMS가 아니라 CDMS에 Compaction Mechanism을 사용하려는 이유는 CDMS가 virtual page 형태로 page를 위치할 수 있기 때문이다. 만약 DBMS에서 compaction이 사용되게 된다면 많은 어려움이 따르게 된다. 예를 들면, DBMS에서 중간에 page가 free 되고 compaction이 요구 되었을 경우 page를 앞으로 옮기는 작업과 이에 따라 RID들이 변경되기 때문에 해당 인덱스들을 변경하여야 하고 abnormal power off를 대비해서 recovery 정책을 세워야 하므로 많은 오버헤드가 발생하게 될 것이다. 하지만 CDMS를 사용하였을 경우에는 데이터 이동 시 맵핑정보 혹은 자유영역정보의 변경으로 이를 지원할 수 있고, 이에 해당하는 recovery 부분도 쉽게 해결할 수 있을 것이라고 판단하였다. 즉, DBMS의 큰 수정 없이 Compaction 작업을 CDMS내에서 처리할 수 있도록 하였다.

4. Compaction Mechanism

CDMS Compaction Mechanism을 설명하기 전에 먼저 몇 가지 용어를 정의하였다.

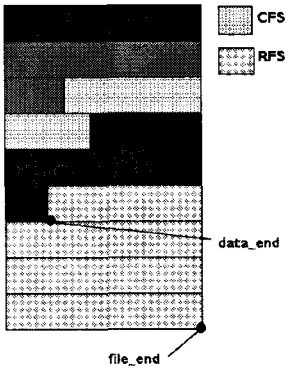

그림 2. CDMS DB파일 구조

◆ CDMS end-points

- **data_end** : CDMS가 관리하는 파일에서 실제 데이터가 들어있는 마지막 지점
- **file_end** : 실제 파일시스템에서 차지하고 있는 DB파일의 크기

◆ Free Space Types

- **Content Free Space(CFS)** : data_end 이전에 존재하는 Free Space(이산적으로 존재)
- **Rear Free Space(RFS)** : data_end 이후에 존재하는 Free Space(연속적으로 존재)

그림 2와 같이 Header정보 뒤에 데이터들이 존재하며 데이터들 사이에 CFS가 존재한다. 그리고 data_end와 file_end 사이에 RFS가 존재하게 된다.

CDMS Compaction Mechanism은 다음과 같다.

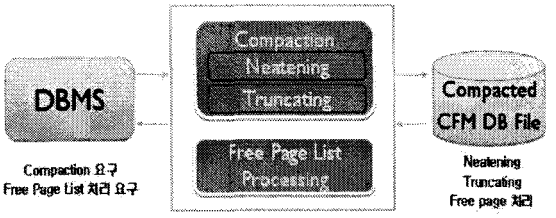

그림 3. CDMS-Compaction Mechanism

DBMS와 CDMS DB파일 사이에서 DBMS가 CDMS에게 Compaction과 Free Page 처리 와 같은 기능들을

요청하게 되면 CDMS는 CDMS DB파일에 대해 위와 같은 기능들을 수행하게 된다. 여기에서 Compaction은 Neatening 과 Truncating 이 두 개의 기능을 포함하게 된다. Neatening이란 DBMS의 요구에 의해 CDMS 모듈이 데이터를 앞쪽으로 모아서 data_end 앞부분의 CFS를 제거하는 동작을 의미하고 Truncating이란 DBMS의 요구에 의해 CDMS 모듈이 data_end 뒷부분에 존재하는 RFS를 제거하여 파일 크기를 줄이는 동작을 의미한다. 따라서 RFS를 최대로 하여 Truncating의 효율을 높이기 위하여 Neatening 은 항상 Truncating 보다 선행되어져서 수행되어 져야 한다. 따라서 Neatening 과 Truncating을 하나로 묶어 Compaction 이라 명칭 하였다. 각 기능들에 대한 자세한 설명은 뒷장에서 논의하도록 한다.

4.1 Free Page 처리

free page란 그 안의 데이터가 더 이상 의미없는 데이터이므로 정보를 보존하지 않아도 되는 page를 의미한다. 예를 들면 free page는 table drop시 발생한다고 예측할 수 있는데, free page 결정시에 DBMS가 가지고 있는 free page list를 넘겨주어서 free page를 판별하여야 한다. free page 처리방법은 free page를 맵핑 정보에서 검색 후 해당하는 page를 삭제한 후 이 삭제된 영역을 재사용 가능하게끔 자유영역정보에 올린다. free page 처리 절차는 표2 와 같다.

표2. Free Page 처리 절차

Free Page 처리 절차
1) DBMS가 Free Page List를 CDMS에게 넘겨준다.
2) 이 List를 기반으로 해당하는 Free Page를 Mapping Table에서 검색한 후 삭제한다.
3) 삭제한 영역을 Free space Table에 올린다.

여기에서 Free Page처리로 인해 CDMS DB파일 내의 data_end 앞에 CFS가 추가적으로 생길 수 있다. 따라서 Free Space처리를 위한 별도의 메커니즘인 Neatening을 필요로 하게 되었다.

4.2 Neatening

Free Page 처리과정 중에 발생하는 Free Space와 data_end 안에 존재하는 Free Space를 제거하기 위하여 Neatening 기법이 제시되었다. Neatening 이란 CFS를 제거하고 데이터를 재배치하는 메커니즘 이다. 이것은 Neatening을 Truncate 이전에 적용함으로써 데이터들을 좀더 Compact하게 배치함과 동시에 CFS를 없애고 data_end를 앞으로 당기는 동작을 의미한다. 따라서 Truncate로 인하여 삭제되는 Free Space가 최대가 되도록 한다. Neatening 방법은 in-place로 진행되며 처리 절차는 표3 과 같다.

표3. Neatening 처리 절차

Neatening 처리 절차
1) 제일 앞의 8k(페이지 크기) 공간을 비워서 해당하는 데이터를 data_end 뒤로 위치시킨다.
2) 8k 뒤에 존재하는 한 페이지에 해당하는 데이터들을 하나씩 앞으로 이동하여 free space를 제거한다.
3) data 이동시 발생하는 읽기 및 쓰기 요청은 위에서 제시한 CDMS의 처리절차를 그대로 따른다.

위의 처리 절차중 8k의 공간을 비워서 데이터를 앞으로 이동시키는 이유는 페이지의 크기가 8k이고, 페이지 단위로 압축을 하게 되므로 항상 8k이하로 데이터 크기가 제한되게 되기 때문이다. 따라서 이동되어야할 공간에 항상 8k이상의 여유공간만 존재한다면 in-place로 데이터를 하나씩 이동시키는 것이 가능하게 된다. Neatening 작업에 고려되어야 할 recovery는 CDMS의 읽기 및 쓰기 처리절차를 그대로 따른다. 즉, 앞에서 설명한 CDMS의 무결성을 위한 갱신절차를 따르게 됨으로써 recovery문제는 간단히 해결될 수 있다. 따라서 Neatening 작업중 abnormal power off가 발생하더라도 나중에 Neatening 작업을 다시 수행함으로써 recovery 문제가 해결될 수 있다.

기본적으로 Neatengin은 Continuous방식에서 동작하게 된다. Minimizing 방식에 Neatening을 적용하지 않은 이유는 첫째로, Minimizing 정책의 특성상 앞쪽 빈 공간부터 채움으로 인해 공간 효율성이 좋기 때문이다. 따라서 CFS의 발생이 continuous방식에 비해 적게 발생하기 때문에 Neatening 효율이 낮기 때문이다. 둘째는, Minimizing방식에서 Neatening을 수행 시 데이터 이동에 따른 오버헤드가 발생하기 때문이다. 즉, data가 분할되어서 저장되어 있고 각각의 분할된 데이터는 링크정보로 연결되어 있기 때문에 분할된 데이터의 이동시 앞쪽 데이터의 링크정보 또한 매번 바꾸어 주어야 한다. 이와 같은 이유로 인해 Static Continuous 방식과 Dynamic Continuous 방식에 Neatening을 적용하였다.

Neatening 과정 후에 RFS를 제거하는 Truncating을 적용시킴으로써 Compaction 과정이 끝마치게 된다.

5. 실험

5.1 Neatening 및 Truncating 효율

Compaction Mechanism을 평가하기 위해 이엠웨어(주)의 MobileLite-Linux 환경을 사용하였다. 먼저

Neatening 및 Truncating 효율을 분석한 다음 Neatening 및 Truncating에 걸리는 시간을 비교하였다. 다음은 Neatening 및 Truncating의 효율을 비교한 것이다.

Neatening 효율 = Neatening size / file size

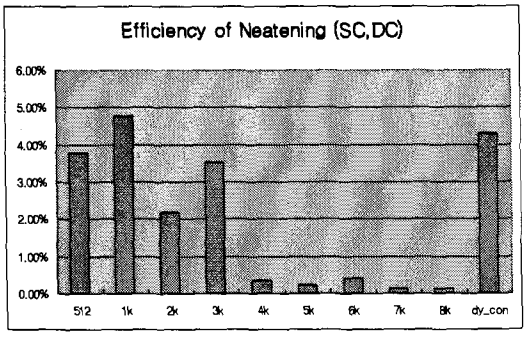

그림 4. Neatening 효율(SC, DC)

그림4는 SC와 DC의 Neatening 효율을 퍼센트로 나타낸 것이다. Ststic Continuous의 경우 슬롯크기를 512byte에서 8kbyte까지 다양하게 변화시켜가며 테스트 하였다. Neatening 효율이 평균 3.5%밖에 되지 않는 것은 Compaction을 적용하기 전의 CDMS가 free space를 효율적으로 관리하여서 소수의 CFS만이 존재하기 때문이다. 그림 5과 그림 6은 Truncate 효율을 나타낸다.

Truncating 효율 = Truncating size / file size

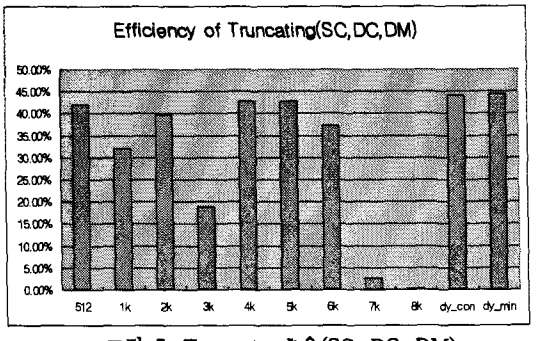

그림 5. Truncate 효율(SC, DC, DM)

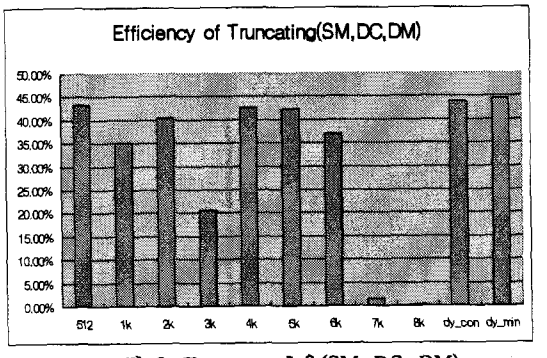

그림 6. Truncate 효율(SM, DC, DM)

Truncating 효율은 파일크기와 data_end의 차이가 크면 높은 효율을 보이는 것을 알 수 있었고, 전반적으로 DM이 가장 큰 효율을 보이고 SC 및 SM이 낮은 효율을 보이는 것을 알 수 있었다. 이 이유는 슬롯을 사용하는 Static 방식에서 발생하는 internal fragment 때문에 Dynamic 방식에 비해서 효율이 낮은 것을 알 수 있었다. 또한 7k, 8k의 경우 Truncate 효율이 극히 낮은데, 이는 page크기와 비슷한 크기의 slot 크기를 가지기 때문에 매우 큰 internal fragment로 인하여 RFS가 거의 존재하지 않기 때문에 낮은 효율을 보인다.

5.2 Neatening 소요시간 및 Truncating 소요시간

다음은 Neatening 과 Truncating 소요시간을 분석한 것이다.

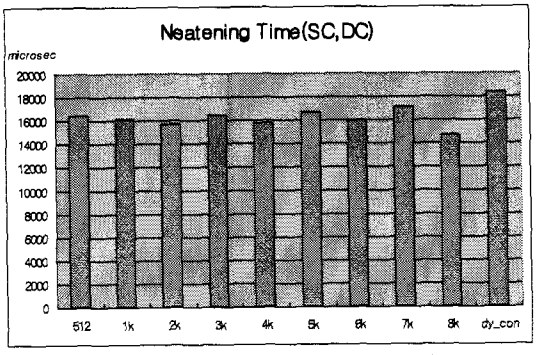

그림 7. Neatening 시간(SC, DC)

그림 7은 각 슬롯크기 및 압축 레벨별로 Neatening 시간을 microsecond로 표현한 것이다. 시간이 가장 많이 걸리는 DC의 경우에도 최대 20 millisecond를 넘지 않는 것을 볼 수 있다. 즉 Neatening 작업이 비교적 짧은 시간 내에 이루어진다는 것을 알 수 있다.

그림 8과 9는 Truncating에 걸리는 시간을 나타낸다.

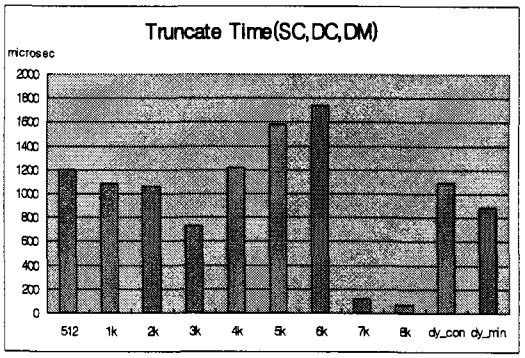

그림 8. Truncating 시간(SC, DC, DM)

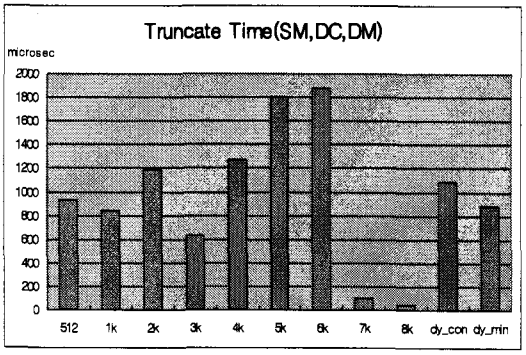

그림 9. Truncating 시간(SM, DC, DM)

위 그림 8, 9에서 알 수 있듯이 Truncating에 걸리는 시간은 2millisecond 정도로 매우 적게 걸리는 것을 알 수 있다. 위의 실험 결과에서 볼 수 있듯이 Compaction 작업으로 인해 매우 큰 공간을 줄일 수 있었다. 또한 일반적으로 모든 경우 Truncating 크기와 시간이 비례하는 것을 알 수 있었으며 7k, 8k 의 경우 Truncating 시켜야 되는 것이 거의 없기 때문에 Truncating 시간 또한 거의 걸리지 않는 것을 알 수 있다.

6. 결론

본 논문에서는 Compaction Mechanism을 CDMS에 적용하여 CDMS DB파일의 크기를 줄여 다른 모듈에서 이 나머지 공간을 사용할 수 있게끔 하였다. Test에 사용된 query set이 삽입/삭제의 반복으로 이루어져 있어 실제 환경과 차이가 있을 수 있지만 Compaction Mechanism으로 인하여 40%~90%의 공간 활용을 할 수 있었다. 비록 Compaction 작업이 수 millisecond 에 해당하는 짧은 작업이지만 Compaction 작업 수행 중 interrupt가 걸려서 도중에 중단해야 하는 경우가 발생할 수 있기 때문에 Neatening 작업을 좀 더 단순화하거나 혹은 작은 단위로 처리하는 기능을 지원해야 할 것으로 보인다. 즉, 조금씩 Neatening을 하고 있다가 어떤 조건(DB file 크기, 사용자 요청 등)에서 truncate 요청이 있을 때 CDMS에서 Compaction을 적용하는 방법이 필요할 것으로 보인다.

7. 참고문헌

[1] M.-L. Chiang , R.-C. Chang "Cleaning policies in mobile computers using flash memory", Journal of Systems and Software, Vol.48, No.3, p.213-231, 1999

[2] K.S. Yim, H. K. Bahn, K. Koh, "A Flash Compression Layer for SmartMedia Card Systems", IEEE Transactions on Consummer Electronics, Vol 50, No 1, 2004

[3] Chung, T.-S. ; Park, D.-J. ; Park, S. ; Lee, D.-H. ; Lee, S.-W. ; Song, H.-J. "System Software for Flash Memory: A Survey", Lecture notes in computer science, Vol.4096, pp.394-404, 2006

[4] 신영재, 황진호, 박정업, 김학수, 이승미, 손진현 "Mobile DBMS를 위한 효율적인 압축 데이터 관리 시스템", 2006년도 한국정보과학회 가을 학술대회, 33권, 2(C)호, pp.6-11, 세종대학교, 20 Oct, 2006

[5] 신영재, 장진근, 이정화, 김학수, 박찬희, 손진현 "모바일 DBMS를 위한 효율적인 압축 데이터 동적 관리 시스템", 제 27회 한국정보처리학회 춘계학술발표대회, 14권, 1호, pp.42-45, 경원대학교, 12 May, 2007

[6] P. Scheuermann, Y. C. Park, and E. Omiecinski, "Heuristic Reorganization of Clustered Files", 3rd International Conference, FODO, 16-30, 1989