유비쿼터스 데이터베이스를 위한 이미지 데이터 처리 기법

서동운*, 최진영
고려대학교 컴퓨터정보통신대학원
e-mail : dongwun@korea.ac.kr*, choi@formal.korea.ac.kr

Image Data Processing for Ubiquitous Database

Dong-Wun Seo*, Jin-Young Choi
Graduate school of Computer and Information, Korea University

요 약

유비쿼터스 컴퓨팅 환경으로 발전하면서 문자열 위주의 획일적 형태에서 음성, 이미지 등 다양한 형태의 데이터들을 처리하게 되었으며, 또한 빠르고 정확하게 처리되기를 요구하고 있다. 현재 데이터 처리 중심부에 있는 Database 는 대부분이 Relation DB 위주로 되어 있어 Datafile 에 데이터를 저장하고 있어 대용량의 이미지 데이터 처리에 적합하지가 않다. 본 논문에서는 이러한 단점을 보강하기 위해 Relation DB 하에서 대용량의 이미지 데이터 처리를 가능하게 하는 기법을 제시한다. 이렇게 함으로써 이미지 데이터를 Upload, Download 시 따른 응답 속도를 보장 할 수 있도록 LRU 알고리즘 기반으로 제안을 하였다. 본 논문에서 제안된 기법은 시뮬레이션을 통해 ①기존 RDB(Relational Database)의 BLOB(Binary Large Object)필드를 이용한 이미지 데이터 처리 방식, ②별도의 저장 공간에 이미지 데이터를 입/출하는 방식, ③별도의 저장 공간에 이미지 데이터를 입/출력할 때 LRU(least Recently Used)알고리즘을 이용하는 방식에 대하여 성능 평가를 하였다. 그 결과 ③별도의 저장 공간에 LRU(least Recently Used)알고리즘을 이용하여 입/출력하는 방식이 ①기존의 RDB(Relational Database)형태에 BLOB(binary large object)필드를 이용한 것 보다 성능이 높음을 확인하였다.

1. 서론

현재 우리가 주로 사용하고 있는 RDB(Relational Database)형태는 유비쿼터스 컴퓨팅환경에서 센싱 네트워크에 의해 생성된 다양한 종류의 데이터를 신속 정확하게 처리하기에는 적합하지 않다[1][2]. 또한 기존의 RDB(Relational Database)의 경우 데이터 타입을 대부분 문자나 숫자타입으로 Table 필드에 저장하고, 데이터파일과 Disk 에 Extent 를 할당하여 기록하는 형태를 띠고 있다[1]. 그러나 이미지나 동영상 등 대용량 데이터를 처리하기에는 많은 취약점과 문제점을 가지고 있다. 간혹, 메모리 DB 를 사용하기도 하지만, 비용대비 처리량이 적어 이미지 데이터를 처리 하기에는 적합하지 않다. 이에 본 논문은 현재 RDB(Relational Database)에서 사용하고 있는 LRU(Least Recently Used)알고리즘[3]과 별도의 물리적 저장 공간을 이용하여 이미지 데이터를 효율적으로 처리 할 수 있는 이미지 처리방식을 제안 한다.

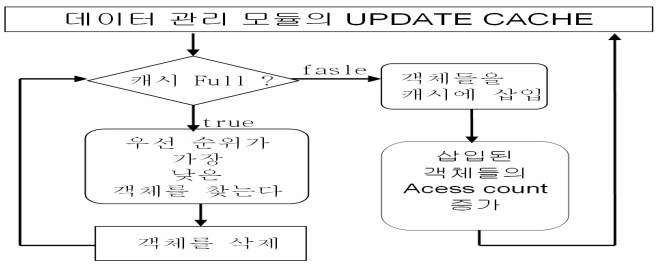
본 논문의 구성은 2 장에서는 기존 RDB(Relational Database)의 데이터 저장 방식 및 메모리 버퍼 캐시의 page 교체방식 시 사용되는 LRU(Least Recently Used)에 대해서 기술하고[4], 3 장에서는 유비쿼터스 컴퓨팅환경에서 이미지데이터 처리에 적합한 Database 의 요구사항을 알아본 후, 이미지 데이터처리 위한 데이터베이스 구조를 제시한다. 4 장에서 시뮬레이션을 통해 ①기존 RDB(Relational Database)에 Table 필드를 BLOB(binary large object)[5]타입으로 생성 후 이미지 데이터를 데이터파일에 입/출력 처리하는 방식과 ② RDB(Relational Database)의 변형된 형태로 별도의 물리적 공간에 이미지 데이터를 입/출력 처리하는 방법, 그리고 ③별도의 물리적 공간에 이미지 데이터를 입/출력 하고, 이 때 LRU(least Recently Used)알고리즘을 사용하는 방식에 대하여 성능 평가 후 ③방식의 우수성을 증명한다. 그리고 5 장에서 향후 과제에 대해서

언급한 후 결론을 맺는다.

2. 관련연구

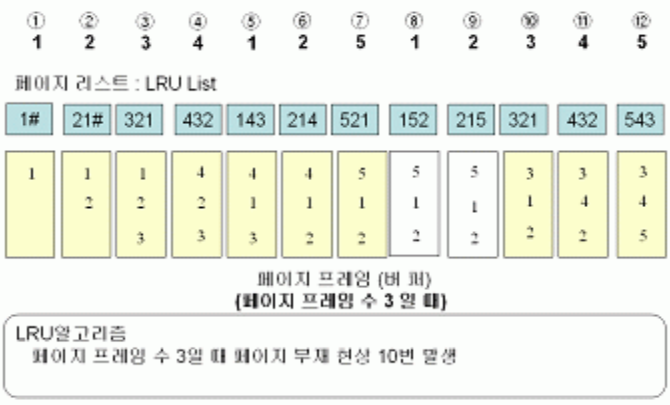
2.1 Page 교체 알고리즘 LRU(Least Recently Used)

LRU(Least Recently Used)은 대표적인 페이지교체 알고리즘으로 각 페이지들이 참조 될 때 마다 그 때의 시간을 기억시킨 후 마지막으로 사용된 시간을 체크하여 최근의 시점에서 가장 오래 전에 사용되었던 페이지를 교체 대상으로 하는 것으로 Locality 를 고려한 알고리즘이다[3]. 또한 다른 알고리즘에 비하여 페이지 교체 효율이 좋고, Belady 변이가 발생하지 않는 이점을 가지고 있다. 현재 데이터베이스에서 메모리 버퍼 데이터를 데이터파일로 내려 쓸 때와 기존에 사용하였던 쿼리문에 대한 재 사용성을 최적화하기 위하여 사용 되고 있다[4]. 그림 1 은 LRU 에 의한 페이지교체 프로세스로 데이터 관리 모듈에 의해서 필요 데이터가 캐시에 존재하는지 여부를 확인하고 존재 여부에 따라 우선순위가 낮은 객체를 삭제하고 필요 객체를 캐시에 삽입하게 된다.



(그림 1) LRU 알고리즘에 의한 페이지교체 프로세스

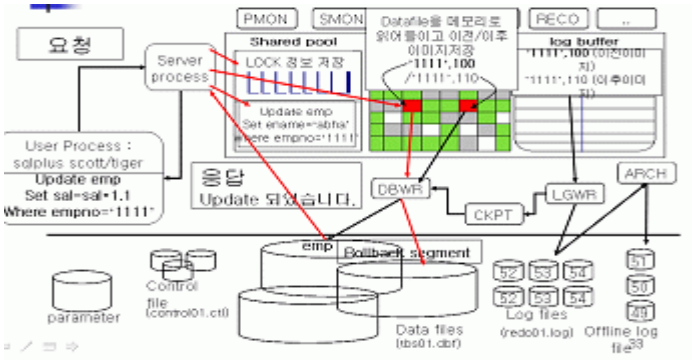
LRU(Least Recently Used)알고리즘의 동작원리는 새로운 객체가 들어오면 페이지 리스트(LRU List)에 정보가 기록이 되고 이 값은 실 페이지에 저장하게 된다. 그리고 계속적으로 객체가 들어 올 때마다 페이지 리스트와 페이지 프레임에 객체를 저장하게 된다. 이때 LRU(Least Recently Used)알고리즘에 의해서 최근 가장 사용 빈도수가 낮은 객체를 삭제하게 된다. 그림 2 는 페이지 프레임수가 3 이고 입력 객체가 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5 일 때 동작하는 LRU(least Recently Used)알고리즘을 나타낸 것이다.



(그림 2) LRU 알고리즘의 동작원리)

2.2 LRU를 이용한 데이터베이스 Read/Write 방식

LRU(least Recently Used)알고리즘을 이용한 데이터베이스의 Read/Write 방식은 메모리 버퍼캐시를 통해 물리적 공간인 데이터파일 또는 Disk 를 직접 작은 Extent 단위로 분할해서 기록하는 형태를 띠고 있다[1]. 그래서 Client 혹은 프로그램의 요구사항 쿼리가 전달되면, 이 쿼리를 메모리 버퍼 캐시 내에서 분석하여 데이터가 메모리 버퍼캐시에 존재하는지 여부를 판단하게 된다. 그리고 메모리 버퍼 캐시에 데이터가 존재하지 않으면 물리적 공간인 데이터 파일로부터 데이터를 읽어서 메모리 버퍼에 올리게 된다. 이때 메모리 버퍼의 기존 데이터들에 이하여 캐시가 Full 되어 빈 버퍼가 없을 경우 LRU(Least Recently Used)알고리즘에 의해서 객체를 삭제하고 요구 데이터를 올리게 된다.



(그림 3) LRU알고리즘을 이용한 데이터Read/Write

3. 유비쿼터스 데이터베이스의 환경적 요구사항

3.1 기존 데이터베이스 처리 방식의 문제점

현재 사용되고 있는 RDB(Relational Database)의 경우 기존 데이터 처리를 메모리 버퍼 캐시를 통해서 데이터파일에 저장하고 있다. 하지만 이런 메모리 버퍼 캐시를 통한 데이터 처리 방법은 유비쿼터스 환경에서 발생되는 다양한 형태의 이미지 및 동영상 데이터를 처리하기에는 적합하지 않다. 기존에 RDBMS (Relational Database Management System)의 경우에도 이미지 데이터를 저장하기 위하여 BLOB(Binary Large Object)데이터 타입을 지원하고 있고 4Gbyte 까지 저장할 수 있으며, 이을 통해서 이미지 데이터를 입/출력할 수 있다. 하지만 센싱 네트워크에 의해서 발생되는 많은 이미지 데이터의 트래픽과 저장공간의 급속한 증가를 감당 할 수는 없다.

3.2 이미지데이터 처리에 적합한 데이터베이스 요구사항 및 구성

유비쿼터스 환경에서 이미지데이터를 처리하기 위해서는 시스템적으로 다음과 같은 요구사항들을 만족해야 한다.

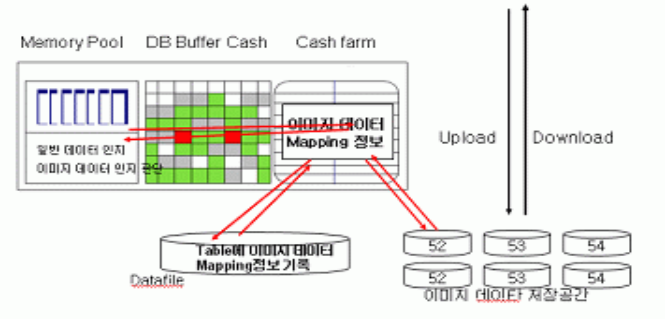
①다양한 이미지데이터를 입/출력 할 수 있어야 한다.
②대량의 이미지데이터를 입/출력 할 수 있어야 한다.
③이미지 데이터의 입/출력이 신속 정확해야 한다.
④저장 공간의 확장성이 있어야 한다.
⑤전송 속도 성능이 좋아야 한다.

이와 같은 요구사항을 만족하기 위해서는 다음과 같

이 데이터베이스 설계가 되어야 한다.
①기존데이터와 별도의 다른 저장공간이 있어야 한다.
②대용량 저장 공간이 필요하다.
③기존데이터 처리방식인 메모리 버퍼 캐시에 이미지 데이터를 올려서 전송하는 방식이 아닌 직접 전송할 수 있는 프로세스가 있어야 한다.
④이미지데이터 저장 공간의 확장이 자유로워야 한다.
⑤이미지데이터를 입/출력할 때 효과적으로 가져 올 수 있는 알고리즘이 있어야 한다.

3.3 유비쿼터스 환경에서 이미지 데이터를 처리하기 위한 데이터베이스의 구조 제안

위에서 언급된 이미지 처리에 적합한 Database 구조는 그림 4 와 같다. 유비쿼터스 환경에서 신속하게 이미지 데이터를 처리하기 위해서는 기존 데이터처리 방식에 별도의 이미지 데이터를 처리 할 수 있는 저장 공간이 필요하다. 즉 같은 속성을 가진 이미지 데이터를 위한 별도의 물리적 공간과 사용자의 요구에 의한 쿼리를 분석할 때 쿼리문이 기존 일반 데이터 처리를 위한 것 인지, 이미지 데이터 처리를 위한 것인지 분석 할 수 있어야 한다. 그리고 데이터 처리를 위한 분석을 마치면, 일반 데이터 처리인 경우는 버퍼 캐시를 통해서 데이터파일에 저장 후 입/출력 하는 형태가 되고, 이미지 데이터의 경우는 쿼리문 분석 후 별도의 Cash Farm 에 이미지 정보를 캐시 한 후 데이터파일의 Table 에 이미지 데이터가 실지로 저장된 물리적 저장 위치정보를 기록하게 된다. 그리고 이 정보에 의하여 실질적인 저장 장소에 이미지 데이터를 Upload 하게 된다. 이미지 데이터 처리 방식에 사용되는 Cash Farm 은 할당 메모리 형태로서 이미지 저장 장치의 위치정보를 가지고 있게 된다. 메모리 버퍼처럼 Cash Farm 의 Full 여부에 따라 LRU(least Recently Used)알고리즘에 의하여 이미지 Mapping 정보를 데이터파일의 Table 에 Read/Write 한다. 또한 별도의 저장 공간에 저장된 이미지 데이터도 사용빈도수에 의하여 사용 빈도수가 높은 이미지는 Cash Farm 에서 이미지 데이터를 캐시 하게 되고, 사용빈도 수가 낮은 데이터는 데이터 저장 공간에서 저장되어 바로 제공하게 된다. 물론 이미지 데이터를 저장 공간에 저장할 때도 LRU(least Recently Used)알고리즘에 의하여 저장한다.



(그림 4) 이미지 데이터 처리에 적합한 DB 구조

그림 4 는 이미지 데이터 처리를 위한 데이터베이스 구조를 나타낸다. 기존의 일반 Text 형 데이터의 처리 단계는 ①사용자 요구 쿼리문 분석 ②요구사항 데이터가 버퍼캐시에 존재 하는가 확인 ③버퍼 캐시에 존재 시 버퍼캐시에서 결과 데이터를 보내준다. ④버퍼캐시에서 데이터가 존재 하지 않을 경우 데이터파일에서 메모리 버퍼로 데이터를 캐시 한다. ⑤캐시 된 데이터를 출력 한다. 이미지 데이터 처리단계는 ①사용자 요구 쿼리문 분석 ②이미지 데이터 처리 요구사항인가 확인 ③Cash Farm 에 이미지 저장 공간 정보 선정 및 데이터파일 내 Table 에 기록 ④이미지 저장 공간에 이미지 Upload/Download ⑥사용빈도수에 의하여 LRU(least Recently Used)알고리즘을 이용 Cash Farm 에 캐시 한다.

4. 실험 평가

본 장에서는 2 장과 3 장에서 언급된 ①기존 데이터베이스의 Table 필드에 BLOB(Binary Large Object) type 의 필드를 만들어 이미지를 입/출력하는 방식 ②별도의 저장 공간에 이미지를 입/출력 하는 방식 ③별도의 저장 공간에 데이터 입/출력 시 LRU(least Recently Used)알고리즘 사용하는 방식에 대한 성능 평가 비교를 하였다. 편의상 위의 ①을 BLOB(Binary Large Object)방식 ②을 별도 이미지저장 처리방식 ③을 별도의 이미지저장 & LRU(least Recently Used)사용 처리 방식으로 명칭 한다.

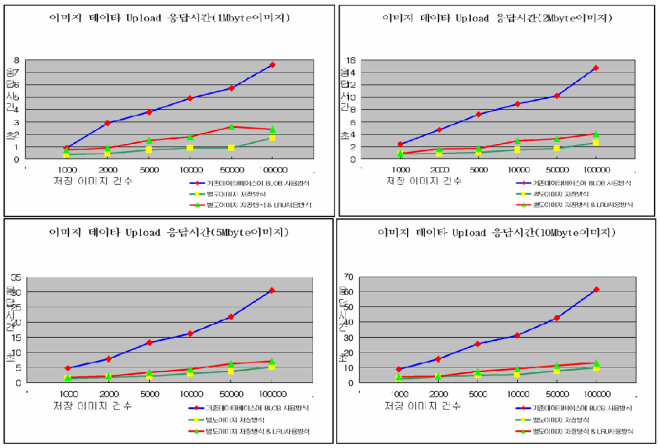
4.1 성능평가 시뮬레이션 환경과 요소

성능평가를 위하여 SUN Microsystems 사의 v890 Unix 시스템에 2Cpu, 1.3G 메모리를 탑재하였으며, 데이터베이스로 Oracle 9i 를 사용하였다. ① BLOB(Binary Large Object)방식에 대한 성능평가를 위해 SUN Microsystems 사의 x3510 스토리지를 Row device 로 마운트하여 데이터파일을 만들었으며, Table 내에 BLOB(Binary Large Object)필드를 만들어 최대 4G 까지의 이미지 데이터를 저장 할 수 있도록 하였다. ②별도의 이미지저장 처리방식에 대한 성능평가를 위해 별도의 이미지 저장 공간으로 같은 x3510 스토리지에 별도의 공간을 파일 시스템으로 300G 를 할당했다. 또한 ③별도의 이미지저장 & LRU(least Recently Used)사용 방식에 대한 성능 평가를 위해 이미지 입/출력 시 사용되는 LRU(least Recently Used)알고리즘을 C++로 구현 했으며, 데이터파일 내에 이미지 Mapping 정보를 Table 필드로 저장하여 SQL 쿼리문을 통해 이미지 저장 위치를 알 수 있도록 구현하였다. 처리 대상 데이터는 1Mbyte, 2Mbyte, 5Mbyte, 10Mbyte 의 이미지 파일로, 로드 발생기를 통해서 동시에 트래픽을 1000, 2000, 5000, 10000, 50000, 100000 씩 증가시킬 때 upload 응답속도를 성능평가 하였다. Download 성능 평가는 1Mbyte, 2Mbyte, 5Mbyte, 10Mbyte Size 의 이미지 파일을 별도의 저장 공간에 1000, 2000, 5000, 10000, 50000, 100000 개씩 저장, 파일들이 같이 존재할 때 이미지를 검색 속도에 의한 Download 응답 시간을 성능평가 한다.

4.2 성능평가 결과

4.2.1 동시 Updater 별 Upload 응답 시간 성능 비교

그림 5 는 1M, 2M, 5M, 10M 크기의 이미지 데이터를 동시 Updater 수가 1000, 2000, 5000, 10000, 50000, 100000 일 때 이미지 데이터 Upload 에 대한 응답시간을 나타낸 것이다.



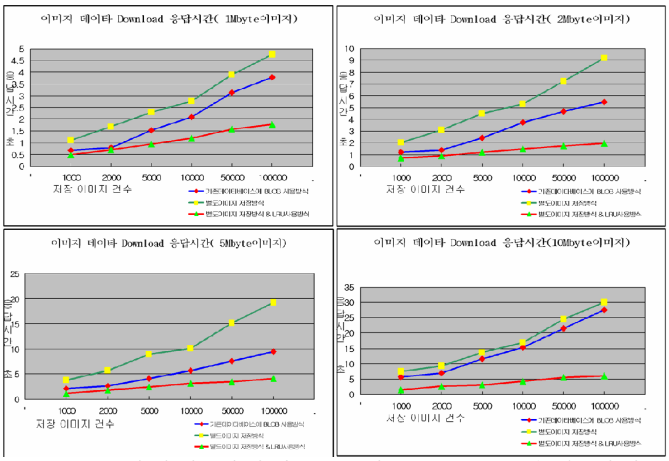
(그림 5) 동시 Updater 증가 별 Upload 응답 성능비교

파란색선 부분은 ① BLOB(Binary Large Object)방식이다. 붉은색선 부분은 ③별도의 이미지 저장 & LRU(least Recently Used)사용 방식이다. 녹색선 부분은 ②별도의 저장 공간 이용한 Upload 하는 방식이다. 그림 5 에서 알 수 있듯이 Upload 시 성능 평가는 ② 별도의 이미지 데이터 저장 처리방식이 가장 좋은 성능을 나타냈으며, 이유는 이미지 데이터를 Upload 할 때는 데이터의 위치정보를 LRU(least Recently Used)에 의해서 할당 받는 것보다 직접 쓰기를 하는 것이 캐시 하는데 소요되는 시간을 단축 할 수 있다는 것을 알 수 있었다. 반면 ①BLOB(Binary Large Object)방식은 데이터 크기가 커지고 Updater 가 많아지면 많아질수록 Disk 에 읽고, 쓸 때 I/O 병목 현상이 증가 하는 것을 알 수 있다. 그림 5 는 동시 Updater 를 증가 시켜가며, 각 각의 응답 시간을 성능 평가 한 것이다.

4.2.2 이미지 데이터 Download 응답시간 성능비교

그림 6 은 이미지 데이터 size 별 Download 응답시간을 성능 비교한 것이며, 이미지 데이터가 1000, 2000, 5000, 10000, 50000, 100000 있을 때, 해당 이미지 데이터를 찾아서 Download 하는데 까지 걸리는 시간을 ① BLOB(Binary Large Object)방식 ②별도의 이미지저장 방식 ③별도의 이미지저장 공간 & LRU(least Recently Used)사용 방식 대하여 성능 평가 한 것을 나타낸다.

그림 6 의 녹색선 부분은 ②별도의 이미지저장 방식을 나타내고, 파란색선 부분은 ①BLOB(Binary Large Object)방식 나타낸다. 그리고 붉은색선 부분은 ③별도의 이미지저장 & LRU(least Recently Used)사용 방식에 대한 성능평가를 나타낸다. 성능평가 결과 Download 시 가장 빠른 응답속도를 보인 방식은 ③별도의 이미지 데이터 저장 & LRU(least Recently Used) 사용 방식이다. 이 방식은 ②별도의 이미지저장 방식의 약 4 배, ①BLOB(Binary Large Object)방식의 약 3 배 정도의 빠른 응답속도를 보였다. 이는 LRU(Least Recently Used)알고리즘을 통해서 별도의 이미지 데이터 공간에서 이미지를 찾는데 응답속도가 빠르기 때문이다.



(그림 6) 이미지 데이터 Size 별 Download 응답 시간

5. 결론 및 향후 과제

본 논문에서는 유비쿼터스 환경에서 발생되는 다양한 이미지 데이터들을 효율적으로 처리 할 수 있는 데이터베이스 구조 및 기법을 제안하여 시뮬레이션을 통해서 성능평가를 하였다. 성능평가를 통해 알 수 있는 것은 ① BLOB(Binary Large Object)방식을 이용해서 이미지 데이터를 처리하는 방식이나 ②별도의 이미지저장 처리방식 보다 ③별도의 이미지 데이터저장 & LRU(least Recently Used)사용 방식이 이미지 데이터를 처리하는데 있어 보다 우수한 성능을 나타내는 것을 알 수 있다. 또한 이러한 데이터베이스가 유비쿼터스 환경에서 사용되어질 때, 다양한 속성의 이미지 및 동영상 데이터를 처리하는데 높은 성능을 보일 가능성이 높고, 유비쿼터스 환경에 적합한 데이터베이스 구조임을 입증하였다. 향후 과제는 본 논문에서 제시한 데이터베이스 구조의 단순화와 상호보완을 통해 보다 나은 성능 향상의 데이터베이스 구조가 나올 수도 있음을 기대하며 향후 연구과제로 남긴다.

참고문헌

[1] Hector Garcia-Molina, Jeffrey D. Ullman Jennifer Windom, "DATABASE SYSTEMS THE COMPLETE BOOK," Prentice-Hall, pp.503-604, 2002.

[2] M. Satyanarayanan, "Pervasive Computing: Vision and Challenges," IEEE Personal Communications, August 2001.

[3] E. J. O'Neil, P.E. O'Neil, G. Weikum. "The LRU-K Page Replacement Algorithm For Database Disk Buffering", Proc. Of the 1993 ACM SIGMOD Conference, pp 297-306, 1993

[4] W. Effelsberg, T. Harder "Principles of Database Buffer Management," ACM Transactions on Database System, Vol. 9, No. 4, pp.560-595 December 1984

[5] M. Rennhackkamp, "The Return of LOBs", DBMS online, April 1997