# 리눅스에서 주기억 데이터베이스를 위한 효율적인 검사점 기법의 구현

김수창 전홍석 노삼혁
홍익대학교 컴퓨터공학과

# Implementation of an Efficient Checkpoint Scheme for Main Memory Database on LINUX

Su Chang Kim H. Seok Jeon Sam H. Noh
*Dept. of Computer Engineering, Hong-Ik University*

**요 약**

주기억 데이터베이스 시스템은 주기억장치에 데이터베이스 전체를 상주시킴으로써 빠른 성능을 보장하므로 실시간 데이터베이스 시스템에 적합하다. 그러나, 시스템에 장애가 발생했을 때는 주기억 데이터베이스의 내용 전체가 손실될 수 있다. 그러므로, 주기억 데이터베이스 시스템의 회복 작업은 매우 중요하다. 또한 빠른 회복을 해줄 수 있어야 실시간 환경에 적합할 것이다. 로그를 사용하는 주기억 데이터베이스 시스템에서 빠른 회복을 위해서 검사점 방법을 사용한다. 검사점을 사용하여 주기적으로 변경된 내용을 디스크로 옮김으로써 회복할 때 분석해야 할 로그의 양을 줄일 수 있다. 본 논문에서는 기존의 검사점 방법들 중 주기억 데이터베이스 환경에 가장 좋은 성능을 보이는 퍼지 검사점에 관한 방법들을 분석 및 보완하여 빠른 회복을 위한 새로운 기법을 제안하고 이를 FastDB 주기억 데이터베이스 시스템에 구현하였다. 구체적으로, FastDB를 로그를 사용하는 회복 방법으로 바꾸고, FastDB가 사용하는 메모리 영역을 n개의 파티션으로 나눈다. 그리고 파티션별 갱신 횟수에 따라 일정한 검사점 수행 간격을 유지하여 회복시 필요한 로그의 양을 효과적으로 줄일 수 있는 일정 간격 퍼지 검사점 기법을 구현하였다. 실험 결과에 의하면 일정 간격 퍼지 검사점 기법을 사용한 시스템이 기존 방법을 사용한 시스템보다 회복 성능에서 우수함을 보여준다.

## 1. 서 론

주기억 데이터베이스 (MMDB) 시스템은 데이터베이스 전체를 주기억장치에 상주시킨 후 데이터베이스 연산을 수행하는 시스템을 말한다. 주기억 데이터베이스 시스템은 모든 연산을 주기억장치에서 수행하므로 디스크 입출력이 발생하지 않는다. 그러므로, 기존의 디스크 기반 데이터베이스 시스템의 문제점인 디스크 입출력횟수를 크게 줄여 실시간 응용에 적합한 빠른 속도를 얻을 수 있다.

그러나, 전원이나 메모리에 이상이 발생하여 시스템을 다시 시작하여야 하는 경우에는 주기억장치에 있는 데이터베이스 전체를 잃어버리게 된다. 따라서 주기억 데이터베이스 시스템에서는 시스템에 문제가 생겼을 경우 안정적인 저장장치로부터 데이터베이스를 빠른 시간 내에 다시 주기억장치에 상주시키고, 동시에 데이터베이스를 일관된 상태로 복구해주는 회복작업이 매우 중요하다. 시스템 장애 발생 후 회복작업에 필요한 회복기법에 관하여 많은 연구들이 제안되고 연구중이지만 아직도 주기억 데이터베이스의 가장 큰 문제점 중 하나로 남아있다 [1, 2].

본 논문에서는 회복 성능을 향상시키기 위한 방법 중의 하나인 파티션 퍼지 검사점 기법을 개선하고 [3], 이를 FastDB 주기억 데이터베이스 시스템에 구현하였다 [4]. 구체적으로, 기존의 파티션 퍼지 검사점 방법은 로컬 검사점을 수행하는데 파티션 사이의 간격을 고려하지 않아서 회복 성능이 느려질 수 있다. 이러한 문제점을 개선하여 보다 빠른 회복 성능을 얻을 수 있도록 제안된 일정 간격 퍼지 검사점 기법을 구현하였다 [5]. 본 논문의 나머지는 다음과 같이 구성되어있다. 2 절에서는 FastDB 주기억 데이터베이스 시스템에 대하여 설명하고, 3 절에서는 FastDB에 일정 간격 퍼지 검사점 기법을 구현한 시스템에 대하여 설명하고, 4 절에서는 기존의 검사점 방법과 일정 간격 퍼지 검사점 기법에 대한 실험을 통하여 성능 평가를 한다. 마지막으로 제 5 절에서는 결론을 맺는다.

본 연구는 한국과학재단 특정기초연구과제(과제번호:98-0102-09-01-3)의 지원을 받았음.

## 2. FastDB 주기억 데이터베이스 시스템

본 절에서는 본 연구에서 구현한 주기억 데이터베이스 시스템의 기본 모델이 되는 FastDB v2.11에 대하여 설명한다. FastDB는 현재 국내 리눅스 배포본에 패키지 형태로 들어있는 주기억 데이터베이스 시스템으로서, 현재 널리 알려진 주기억 데이터베이스 시스템이다. 현재 2.11버전까지 나와있고, Unix, Linux, Windows NT 환경에서 동작되며, 무료로 배포된다.

FastDB는 데이터에 대하여 다른 하나의 복사본을 생성하여 사용하는 쉐도우 회복기법을 사용한다 [6, 7]. 그림 1은

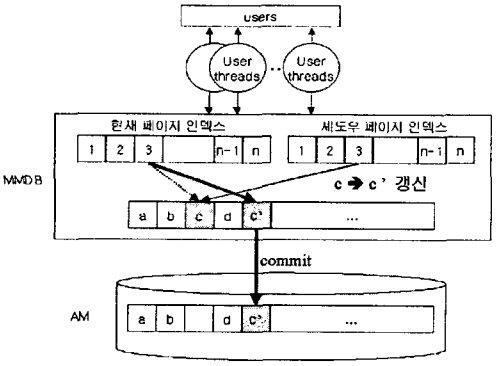


그림 1. FastDB 데이터베이스 시스템에서의 회복

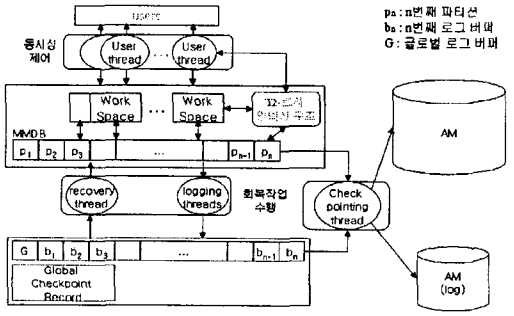


그림 2. 수정한 FastDB 구조도

FastDB 시스템에서의 회복 방법을 보여준다.

사용자가 MMDB에 접근하여 c의 내용을 c'로 수정하려고 할 때, 현재 페이지 인덱스와 섀도우 페이지 인덱스는 모두 c를 가리키고 있다. c에 대하여 수정이 생기면 c를 복사한 후 현재 페이지 인덱스가 c'를 가리키도록 한다. 그리고 모든 수정 작업은 c'에서만 적용을 시킨다. 모든 작업이 끝나고 commit이 되면 섀도우 페이지 인덱스도 c'를 가리키게 하고 c를 해제(free)한 후 바뀐 내용을 Archive Memory (AM)에 반영한다. 시스템에 문제가 생겨서 트랜잭션이 비정상적으로 끝나거나 commit을 하지 못하게 되면, 현재 페이지 인덱스는 c'의 연결을 끊고 수정 전의 내용을 가지고 있던 c를 가리키게 함으로써 수정 전의 상태로 회복할 수 있다.

FastDB는 현재 페이지 인덱스의 연결을 바꾸기만 하면 회복작업이 수행되므로 회복이 빠른 반면, 데이터베이스 단위의 락을 사용하고, commit할 때마다 AM에 갱신된 내용을 반영해야 하므로 트랜잭션의 수행 성능은 매우 나쁘다.

## 3. 일정 간격 퍼지 검사점 기법 구현

본 연구에서는 FastDB의 회복기법과 관련된 부분을 수정하였다. 우선 FastDB의 섀도우 회복기법을 로그를 사용한 회복 기법으로 수정하고, 회복 성능을 증가시키기 위해 지연 갱신 기법을 사용하였다 [8, 9]. 지연 갱신은 트랜잭션

```
Algorithm. RegularIntervalFC
    step 1. partition MM based on their update frequencies
    step 2. calculate checkpoint frequency ( CF_i )
    step 3. order partitions by CF_i
    step 4. call makeCheckpointList
    step 5. perform a local checkpoint based on the checkpoint list
    step 6. reset CF_i for all partitions
            and goto step 3
```

그림 3. 일정 간격 퍼지 검사점 알고리즘 (1)

```
Algorithm. makeCheckpointList
    step 1. i=1, p=1
    step 2. IV_i = ⌈ C_i / CF_i ⌉
    step 3. calculate G_ij
    step 4. if ( orderlist[p] is set ) begin
                if ( orderlist[p-1] is not set )
                    set orderlist[p-1] to partition number of G_ij
                else begin
                    p = p+1 and goto step 4
                end
            end
            else begin
                set orderlist[p] to partition number of G_ij
            end
    step 5. for all entry of G_ij, CF_i = CF_i-1
    step 6. if ( CF_i>0 ) then p=i+IV_i
            and goto step 4
    step 7. i=j+1, p=i
            if ( i<=n )  goto step 1
            else return  orderlist
```

그림 4. 일정 간격 퍼지 검사점 알고리즘 (2)

이 commit하기 전까지의 모든 데이터베이스의 변경을 지연시킨 다음 commit이 되면 지연시킨 내용을 데이터베이스에 반영하는 방법이다. 이 방법은 트랜잭션이 commit 되기 전에 abort되거나 시스템 failure가 생겼을 때 저장해 두었던 데이터를 그냥 버리고 무시하면 된다. 그러므로 회복 시에 Redo 로그만이 필요하므로 빠른 회복을 할 수 있다 [10]. 그리고 본 논문에서는 로그를 비휘발성 메모리에 저장하여서 회복 성능을 향상시키도록 하였다 [11]. 그림 2는 본 연구에서 구현한 주기억 데이터베이스 시스템의 구조도이다.

그림 2에서 MMDB는 휘발성 메모리에 있으며 크기를 32MByte로 고정시키고, 이를 n개의 파티션으로 나누어 사용하였다. 하나의 파티션은 여러 페이지들로 구성되어 있으며 물리적으로 연속하지 않은 페이지들도 같은 파티션에 속할 수 있다. 비휘발성 메모리에는 n개 파티션의 로그를 저장할 수 있도록 n개의 로컬 로그 버퍼와 1개의 글로벌 로그 버퍼를 사용하고, 회복시 빠르게 로그를 분석하기 위하여 Global Checkpoint Record (GCR)를 사용하였다. GCR에는 각 파티션의 가장 최근의 완전하게 끝난 검사점의 위치를 저장한다. Workspace는 트랜잭션 단위로 생성되는데, 그 트랜잭션이 갱신한 모든 내용을 저장하였다가 commit시에만 MMDB에 반영하는데 사용한다. User thread들은 MMDB에 접근하여 사용자들의 트랜잭션을 수행하며, logging thread는 MMDB의 갱신된 내용들을 로그버퍼에 기록하는 작업을 수행하고, recovery thread는 회복시 로그 버퍼의 내용을 가지고 MMDB를 복구하는 작업을 수행한다. Checkpointing thread는 MMDB와 로그 버퍼의 내용을 AM으로 반영해주는 작업을 수행하는데 일정 간격 퍼지 검사점 기법을 적용하였다. 그 알고리즘은 그림 3, 그림 4와 같다.

표 1. 실험 환경

| CPU | Pentium III 550 MHz |
|---|---|
| 메모리 | 128MB |
| 운영체제 | LINUX kernel version 2.2.16 |
| MMDB 크기 | 32 MByte |
| 파티션의 수 | 8 개 |
| 입력 데이터 갯수 | 5,000개 * 5개 트랜잭션 |

## 4. 실험 및 성능평가

실험은 수정한 FastDB 주기억 데이터베이스 시스템에서 진행하였으며, 실험 환경은 표 1과 같다.

주기억 데이터베이스 시스템에 failure가 발생하였을 때 회복에 필요한 로그의 크기를 시스템의 작업량에 따라 측정하였다. 5개의 트랜잭션이 동시에 수행되며 각 트랜잭션은 5,000개의 데이터를 입력하는 작업을 수행한다. 시스템의 failure는 특정 시점에서 항상 발생하도록 하였으며, failure 발생 후 로그 버퍼에 있는 회복에 필요한 총 로그의 크기를 KByte 단위로 측정하였다. 본 논문에서 제안한 일정 간격 퍼지 검사점 기법과 기존의 퍼지 검사점 기법을 수정한 FastDB에 사용했을 때의 로그 크기를 비교하였다.

그림 5는 시스템의 작업 부하에 따라서 회복시에 필요한 로그의 크기를 비교한 실험이다. X 축은 5개의 트랜잭션이 각각 5,000개의 데이터를 입력할 때 검사점의 횟수가 1번 일어났을 때부터 10번 일어났을 때까지를 나타낸다. 검사점의 횟수가 1번 일어난 것은 검사점 1회 실행동안 25,000 개의 데이터가 입력 된 것이므로 매우 작업량이 많은 시스템을 나타내고, 10번 발생한 것은 작업량이 매우 적은 시스템을 나타낸다. 그림 5에서 작업량이 많은 시스템에서 일정 간격 퍼지 검사점 기법을 사용한 시스템은 갱신이 많은 파티션의 검사점을 일정한 간격을 두고 수행해주므로 회복에 필요한 로그의 크기가 많이 줄어 빠른 회복을 할 수 있다. 특히 시스템의 작업량이 매우 많을 때를 살펴보면 기존의 검사점 방법을 사용했을 때에 회복에 필요한 로그 크기보다 53%정도 적은 로그만이 필요하다. 하지만 시스템의 작업량이 작을 때에는 기존 방법보다 약간의 로그가 더 필요하지만 이 값의 차이는 매우 작으므로 회복 성능에는 큰 차이가 없다. 전체적으로 기존 방법보다 약 50%정도 적은 로그만이 회복에 필요하게 되므로 매우 빠른 회복 작업을 수행 할 수 있다.

## 5. 결 론

주기억 데이터베이스 시스템에서 기존의 로그를 이용한 회복 방법은 회복 시 방대한 양의 로그를 검색하고, 분석하는 시간 비용 문제 때문에 회복 시간이 지연된다. 본 논문의 일정 간격 퍼지 검사점 기법은 기존의 파티션 단위의 퍼지 검사점 수행 방법의 문제점을 해결하여 어느 시점에서나 로그의 크기가 적게 필요하여 보다 빠른 회복 성능을 보여 준 다

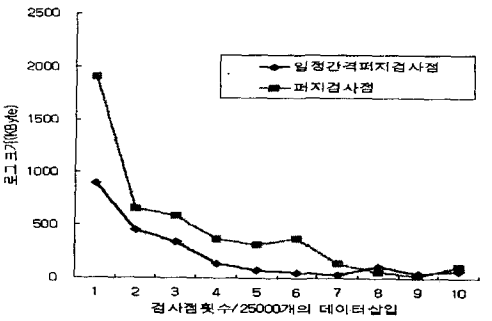


그림 5. 시스템의 작업 부하에 따른 회복에 필요한 로그 크기 실험

## 참고문헌

[1] L. Gruenwald, J. Haung, M. H. Dunhan, J. L. Lin, A. C. Peltier, Recovery in Main Memory Databases, *International Journal of Engineering Intelligent System*, Vol. 4, No. 3, 1996

[2] H. V. Jagadish, A. Silberschatz and S. Sudarshan, Recovering from Main-Memory Lapes, *In Proceedings of the 19th VLDB Conference*, 1993

[3] J. Huang and L. Gruenwald, An Update-Frequency-Valid-Interval Partition Checkpointing Technique for Real-Time Main Memory Databases, *First International Workshop on Real-Time Databases*, 1996

[4] *http://www.ispras.ru/~knizhnik*

[5] 김수창, 전홍석, 노삼혁, 주기억 데이터베이스 시스템에서의 일정 간격 퍼지 검사점 기법, *한국정보과학회 추계 학술발표회의 논문지*, 제26권 2호, 1999

[6] J. Gray, The Recovery Manager of the System R Database Manager, *ACM Computing Surveys*, Vol. 13, No. 2, 1981

[7] T. Ylonen, Shadow paging is Feasible, *Technical Report*, Department of Computer Science, Helsinki University of Technology, 1994

[8] L. Gruenwald, Y. Chen and J. Huang, Effect of Update Techniques on Main Memory Database System Performance, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 10, No. 5, 1998

[9] 우승균, 이윤준, 김명호, 주기억장치 데이터베이스 시스템에서의 회복기법에 대한 고찰, *한국정보과학회지*, 제14권 2호, 1996

[10] D. Lomet and M. R. Tuttle, Redo Recovery after System Crashes, *In Proceedings of the 21th VLDB Conference*, 1995

[11] 백영석, 진성일, 비휘발성 메모리를 이용한 데이터베이스 회복 구조 및 기법, *한국정보과학회지*, 제14권 2호, 1996