주기억 데이터베이스에서의 일정 간격 퍼지 검사점 기법

김수창 전홍식 노삼혁
홍익대학교 컴퓨터공학과

Regular Interval Fuzzy Checkpointing Technique for Main Memory Databases

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 현재 실시간 데이터베이스 시스템으로 가장 많이 사용되고 있다 그러나, 시스템에 장애가 발생했을 때는 주기억 데이터베이스의 내용전체가 손실될 수 있다 그러므로, 주기억 데이터베이스 시스템의 회복 작업은 매우 중요하다. 또한 빠른 회복을 해 줄 수 있어야 실시간 환경에 적합할 것이다 빠른 회복을 위한 방법중의 하나는 검사점을 사용하여 회복할 때 분석해야 할 로그의 양을 줄이는 것이다 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 제안한다. 구체적으로, 주기억 데이터베이스를 갱신 횟수에 따라 파티션을 나눈 후 각 파티션 단위로 퍼지 검사점을 수행할 때 기존 방법은 검사점수행 순서가 비효율적이서 회복시 필요한 로그의 양을 효과적으로 줄일 수 없다. 본 논문에서 제안하는 알고리즘은 파티션별 갱신횟수에 따라 일정한 검사점 수행 간격을 유지하므로 회복시 필요한 로그의 양을 효과적으로 줄임으로써 보다 빠른 회복이 가능하다

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 데이터베이스의 문제점인 디스크 입출력횟수를 크게 줄여 실시간 응용에 적합한 빠른 속도를 얻을 수 있다.

그러나, 전원이 나간다거나 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.

본 논문에서는 회복성능을 향상시키기 위한 방법중의 하나인 퍼지 검사점 기법을[1] 개선하여 보다 빠른 회복성능을 얻을 수 있는 새로운 퍼지 검사점 기법을 제안한다.

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

본 논문의 구성은 다음과 같다. 제 2 절에서는 주기억장치 데이터베이스 시스템의 기존의 회복기법과 검사점 기법에 대해서 알아본다. 제 3 절에서는 주기억장치 시스템 구조를 제안하고, 제 4 절에서는 일정 간격 퍼지 검사점 기법이라는 새로운 검사점 기법을 제안하고, 제 5 절에서는 결론을 맺는다.

2. 관련연구

본 절에서는 주기억장치 데이터베이스 시스템의 회복 성능을 향상시키기 위한 기존의 방법들을 살펴보도록 하겠다.

데이터베이스 시스템의 회복 성능을 향상시키기 위한 검사점 수행은 시스템의 빠른 재시동을 위하여 주기적으로 데이터베이스의 상태를 안전한 기억장소로 반영해주는 일련의 과정을 말한다. 검사점을 수행해 줌으로 해서 복구시 처음부터의 로그를 분석해야 하는 것이 아니라 가장 최근의 완전한 검사점부터의 로그만을 분석하여 복구를 해주면 된다.

검사점 수행 방법은 트랜잭션일치, 액션일치, 그리고 퍼지 검사점 수행 방법이 있다 [1] 트랜잭션일치 검사점 방법은 검사점이 시작되면 모든 트랜잭션이 완료될 때까지 기다린 후 검사점이 수행된다. 액션일치 검사점 방법은 트랜잭션을 액션의 연속으로 보고 액션이 완료될 때까지 기다린 후 검사점을 수행하게 된다. 이들 두가지 방법은 검사점 수행도중에 모든 트랜잭션의 생성이나 갱신연산을 허용하지

많음으로 해서 데이터베이스의 성능을 저하시키게 한다

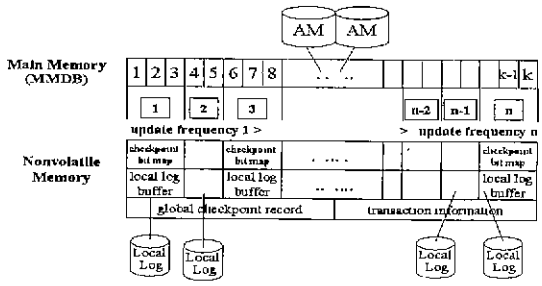
이러한 단점을 극복하기 위하여 나온 것이 퍼지 검사점 방법이다 [1] 이 방법은 검사점 수행도중 트랜잭션의 갱신 연산과 생성을 중지하지 않고 검사점과 트랜잭션을 동시에 수행함으로써 빠른 성능을 얻을 수 있다. 반면에 주기억장치의 데이터베이스와 안정저장장치의 일관성이 깨질 수 있는 문제점이 발생한다 그러나 이 문제점들은 비휘발성 메모리나 핑퐁기법을[2] 사용함으로서 해결을 할 수 있다.

퍼지 검사점 방법은 로그에 수행중인 트랜잭션과 갱신된 페이지들의 리스트를 기록하여 이것을 가지고 검사점을 수행하는 방법과 검사점이 시작되면 갱신된 페이지들을 안정저장장치로 덤프 하는 방법이 있다. 첫번째 방법은 트랜잭션의 수행이 많은 주기억 데이터베이스의 특성 때문에 로그의 양이 매우 커지게 되어 회복시 성능이 많이 저하되게 된다 반면에 두 번째 방법은 분석해야할 로그의 양이 적으므로 회복이 보다 빠르다 하지만 여전히 로그의 양이 많으므로 로그의 양을 더 줄이기 위하여 두 번째 방법을 개선시킨 세그먼트 퍼지 검사점[3], 파티션 검사점[4] 기법등이 연구되었다.

세그먼트 퍼지 검사점 방법은 주기억장치를 순서대로 n개의 세그먼트로 나누어 각 세그먼트를 순서대로 검사점을 수행시키는 방법으로 복구시 가장 최근의 검사점 수행이 끝난 n개의 세그먼트부터의 로그의 양만을 필요로 하게된다 파티션 검사점 기법은 주기억장치의 페이지들을 많은 갱신이 일어난 순서대로 정렬을 하여 파티션을 나눈 다음, 파티션 단위로 검사점을 수행하는데 많은 갱신이 일어난 파티션은 더 많은 검사점을 수행하게 된다. 그러므로, 갱신이 많이 일어나는 파티션의 로그의 양을 줄여서 빠른 회복이 가능하다. 그러나, 검사점 수행횟수가 많은 파티션부터 검사점을 수행하므로, 갱신이 많이 일어나는 파티션은 검사점 수행을 앞부분에는 자주 해주지만 뒷부분으로 갈수록 더 적게 수행해주게 된다 즉, 앞부분은 자주 검사점을 수행해주므로 로그의 양이 작은 반면 뒷부분은 로그의 양이 매우 커지게 되어서 효율적으로 로그의 크기를 줄일 수가 없다는 문제점이 있다

3. 시스템 구조

본 논문에서 제안한 일정 간격 퍼지 검사점 기법을 적용하기 위한 시스템 구조는 <그림 1>과 같다.



<그림 1> 시스템 구조

보통의 트랜잭션 연산을 수행하는 데이터베이스 프로세서와 회복에 관련된 연산을 수행하는 회복 프로세서를 사용하여 동시에 수행되도록 하였으며, <그림 1>에서와 같이 비휘발성 메모리를 로그버퍼와 checkpoint bit map등에 사용하였다 주기억장치는 페이지 단위로 구성되어 있는데 이 페이지들을 갱신 횟수기 많은 순서대로 묶어서 n개의 파티션으로 나누었다 각 파티션의 페이지들은 물리적인 순서대로 묶일 필요는 없지만 논리적으로 연결되어 있어야 한다. 또한 각 파티션의 사이즈는 같지 않아도 된다 각 파티션별로 지난번 검사점 후의 갱신여부를 알 수 있도록 checkpoint bit map을 두고, local log buffer에 로그를 저장하도록 하였다. local log buffer의 내용은 주기적으로 local log에 백업된다. global checkpoint record에는 각 파티션의 가장 최근의 완전하게 검사점이 수행된 위치를 기록하며, transaction information에는 트랜잭션의 abort 리스트를 유지한다. 주기억장치의 데이터베이스는 Archive Memory (AM)에 백업되어 유지된다. 검사점은 파티션 단위로 이루어지며 이를 로컬 검사점이라 부르고, 모든 파티션들의 검사점 수행이 이루어지면 이를 글로벌 검사점 수행이 완료 됐다고 한다. 갱신이 많은 파티션은 로그의 양을 줄이기 위하여 갱신이 적은 파티션 보다 더 많은 검사점을 수행해준다 검사점 수행시 파티션의 갱신된 내용을 AM으로 백업을 한다.

4. 일정 간격 퍼지 검사점 기법

일정 간격 퍼지 검사점 방법은 본 논문에서 회복성능을 높이기 위하여 제안한 검사점 기법으로써, 같은 파티션의 검사점의 수행순서에 일정한 간격을 줌으로 회복성능을 향상 시켜준다 기존의 파티션 퍼지 검사점 기법에서는[4] 검사횟수가 많은 파티션에 대하여 먼저 검사점을 수행한다. 그래서 갱신이 많은 파티션의 검사점 수행순서가 앞부분에는 몰려 있고 뒷부분은 띨어져 있게되어 효율적으로 로그의 양을 줄일 수가 없다. 본 논문에서 제안한 일정 간격 퍼지 검사점 방법은 검사점 수행순서에 간격을 부여하여 일정하게 유지해 줌으로써 어느 시점에서나 로그의 양이 작게 유지되어서 빠른 회복이 가능하다. 검사점 수행리스트를 만들기 위한 오버헤드는 파티션의 갯수가 수십개 이내이고, 회복 프로세서를 사용하므로 매우 작다. <그림 2>과 <그림 3>는 일정 간격 퍼지 검사점 방법의 알고리즘이다 CFi는 i 파티션의 검사점 수행 횟수를 나타내고, Ct는 총 검사점 수행 횟수, IVi 는 i 파티션의 검사점 간격, orderlist는 검사점 수행 리스트, Gij는 i-j 파티션의 집합 (단, CFi = CFi+1 = ... = CFj-1 = CFj), p는 orderlist에 기록할 위치를 나타내는 포인터이다.

<그림 4>에서 총 파티션의 수는 10개이고 1번 파티션의 이름은 'a'이고 CF1=5 이다. a 파티션의 CF값이 5이므로 검사점 수행을 5번 해주어야 한다. CF값이 큰 파티션 일수록 검사점 수행을 더 많이 해준다. 나머지 파티션도 각각 CF2=3, . , CF10=1 의 값을 갖는다. 글로벌 검사점을 수행하려면 20번(Ct=20)의 로컬 검사점을 수행하여야 한다.

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

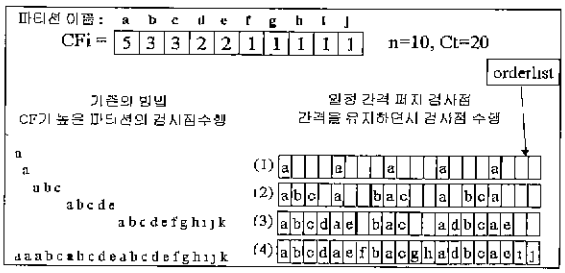
<그림 2> 일정 간격 퍼지 검사점 알고리즘

```
Algorithm. makeCheckpointList
    step 1 i=1, p=1
    step 2 IVi = Ct/CFi 값의 반올림
    step 3 calculate Gij
    step 4 orderlist에 p위치부터 빈곳에 Gij의 파티션번호를 기록
           p 위치가 채워져 있으면 가장 가까운 빈곳에 기록
           빈곳의 위치가 같으면 왼쪽부터 기록
           (i번 파티션의 기록한 위치를 p로 함)
           Gij의 i 파티션의 CF값을 1씩 감소
    step 5. if CFi > 0 then p=i+IVi and goto step 4
    step 6 i=i+1, p=i
           if i<=n goto step1
           else return orderlist
```

<그림 3> 검사점 수행 순서를 만드는 알고리즘

<그림 4>와 같은 예가 주어졌을 때 기존의 검사점 수행은 CFi의 값이 큰 파티션부터 검사점을 수행해준다 그러므로 CF값이 5인 a 파티션부터 검사점이 수행된다. 그리고 a값이 1감소되고 역시 4로서 가장 크므로 한번 더 a의 검사점이 수행되고, 다음에 a 의 값이 1 감소되어 3으로 되었으므로 3과 같은 값을 가지는 a, b, c가 수행된다. 이렇게 모든 파티션의 CF값이 0이 될 때까지 수행하면 된다.

본 논문에서 제안한 일정 간격 퍼지 검사점 방법에서는 제일 큰 CF값을 가지는 a 파티션부터 orderlist라는 빈 배열에 간격을 주어서 넣는다. a 파티션의 간격은 20/5 의 반올림 값이므로 4가 된다. a 파티션을 orderlist에 넣으면 <그림 4>의 (1)과 같이 된다. 다음에는 b, c 파티션이 가장 큰 값을 가지므로 b, c 파티션을 간격을 유지하여 orderlist에 넣는다 b 파티션의 간격은 20/3의 반올림 값이므로 8이 된다. b, c 파티션을 8의 간격을 유지하면서 orderlist에 넣는다 넣을 자리가 이미 채워져 있다면 그 위치에서 가장 가까운 빈자리에 넣는다. 이런 방법으로 모든 파티션에 대하여 수행하면 (4)와 같은 orderlist를 만들어진다. 이 orderlist의 순서대로 검사점을 수행하기만 하면 된다



<그림 4> 일정 간격 퍼지 검사점 기법 수행 예

<그림 4>에서 기존의 방법에 의한 검사점 수행 결과는 "aaabcabcdeabcdefghijk"로서 갱신이 많은 파티션 a에 대하여 앞부분은 검사점을 여러번 연속해서 수행해 주어서 로그의 양을 많이 줄였지만 뒷부분은 검사점 수행의 간격이 넓어서 로그의 양이 많다 그러므로 회복 성능이 좋지 않게 된다. 하지만 일정 간격 퍼지 검사점 방법에 의한 결과를 보면 "abcdaefbacghadbcaeij"로서 a파티션 뿐만 아니라 다른 모든 파티션의 검사점 수행 간격이 어느 시점에서나 일정하다 그러므로, 항상 로그의 양이 일정하고 작게 유지되므로 회복성능이 보다 향상된다.

5. 결론

주기억 데이터베이스 시스템에서 기존의 로그를 이용한 회복 방법은 회복시 방대한 양의 로그를 검색하고, 분석하는 시간 비용 문제 때문에 회복시간이 지연된다 본 논문에서 제시한 일정 간격 퍼지 검사점 기법은 기존의 파티션 단위의 퍼지 검사점 수행 방법의 문제점을 해결하여 보다 빠른 회복 성능을 보여준다 일정 간격 퍼지 검사점 기법은 갱신이 많은 파티션의 검사점 수행에 일정한 간격을 줌으로 해서 갱신이 많은 파티션의 로그의 양을 효과적으로 줄일 수 있다 그렇기 때문에 어느 시점에서나 빠른 회복이 가능하다

참고문헌

[1] Hearder, T and Reuter, A, "Principle of Transaction-Oriented Database Recovery", *ACM Computing Surveys*, Vol. 15, No. 4, 1983

[2] Salem, K. and Garcia-Molina, H., "Checkpointing memory-resident databases", *In Proceedings of the Fifth IEEE International Conference on Data Engineering*, 1989

[3] Lin, J. L. and Margaret, H. D., "Segmented Fuzzy Checkpointing for Main Memory Databases", *ACM Symposium on Applied Computing*, 1996

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