실시간 시스템에서 퍼지 검사점을 이용한 주기억 데이타베이스 프로토타입 시스템의 설계

박 용 문[†] · 이 찬 섭[††] · 최 의 인[†††]

요 약

최근 들어 실시간 시스템의 사용이 늘어나고 있다. 하지만 실시간 시스템은 시스템의 구성요소가 디스크 데이타베이스 시스템으로 이루어지기 때문에 트랜잭션 처리 시 디스크 입출력으로 인해 오버헤드가 증가되고 시스템 파손 시 회복이 늦어진다는 문제점이 존재한다. 이 논문에서는 이런 문제점의 개선을 위해 시스템 환경을 주기억 데이타베이스 프로토타입 시스템으로 설계하였다. 이 환경에서 제안된 회복 기법은 크게 로그 처리의 검사점으로 나누어진다. 회복 기법은 그룹 종료 방법을 이용한 재수행 로그 기법과 기존의 퍼지 검사점 기법이 가지고 있는 문제점을 개선한 퍼지 검사점 기법을 제안하였다. 마지막으로 제안된 기법과 기존의 기법을 두 가지 측면으로 나누어 SLAM II를 사용하여 성능 평가와 비교 분석을 하였다.

Design of Main-Memory Database Prototype System using Fuzzy Checkpoint Technique in Real-Time Environment

Yong-Mun Park[†] · Chan-Seob Lee[††] · Eui-In Choi[†††]

ABSTRACT

As the areas of computer application are expanded, real-time application environments that must process as many transactions as possible within their deadlines, such as a stock transaction systems, ATM switching systems etc, have been increased recently. The reason why the conventional database systems can't process soft real-time applications is the lack of prediction and poor performance on processing transaction's deadline. If transactions want to access data stored at the secondary storage, they can not satisfy requirements of real-time applications because of the disk delay time. This paper designs a main-memory database prototype system to be suitable to real-time applications and then this system can produce rapid results without disk i/o as all of the information are loaded in main memory database. In thesis proposed the improved techniques with respect to logging, checkpointing, and recovering in our environment. In order to improve the performance of the system, a) the frequency of log analysis and redo processing is reduced by the proposed redo technique at system failure, b) database consistency is maintained by improved fuzzy checkpointing. The performance model is proposed which consists of two parts. The first part evaluates log processing time for recovery and compares with other research activities. The second part examines checkpointing behavior.

※ 이 논문은 1997년도 한국 학술진흥재단의 학술 연구비(학신 1997-002-e00216)에 의하여 기원되었음.
† 정 회 원 : 한국전자통신연구원 선임연구원
†† 정 회 원 : 한남대학교 대학원 컴퓨터공학과
††† 종신회원 : 한남대학교 컴퓨터공학과 교수
논문접수 : 1999년 3월 23일, 심사완료 : 2000년 5월 22일

1. 서 론

컴퓨터의 응용 영역이 확대됨에 따라 단순한 트랜잭션 수행뿐만 아니라 이동 통신 등의 실시간 처리를 요구하는 응용 분야가 증가되므로 인하여 일정 시간 이내에 트랜잭션 수행을 완료해야 하는 실시간 데이타베이스 시스템의 필요성이 대두되었다. 그러나, 기존의 디스크 기반 데이타베이스 시스템에서는 사용자의 질의 수행 시 데이타 입출력을 위한 계속적인 디스크 접근과 예측성(predictability) 부족 때문에 실시간 응용에 적합하지 않다. 즉, 기존의 데이타베이스 시스템이 실시간 응용을 지원하지 못하는 첫 번째 문제점은 예측성 부족이다. 각 트랜잭션이 접근하려는 데이타의 일관성을 보존해야하는 일관성 제약 조건은 트랜잭션으로 하여금 어느 시점에서 블록되거나 교착상태가 발생되도록 유발한다. 이런 이유로 트랜잭션들의 예상 수행 시간을 예측하기가 어려워 마감시간을 만족하기 어려운 문제점이 발생한다. 두 번째 문제점은 성능 저하이다. 즉, 트랜잭션 처리는 보조 기억 장치에 저장되어 있는 데이타베이스 접근을 요구하므로 트랜잭션의 응답 시간이 ms 단위의 디스크 접근 지연에 의해서 제한된다. 따라서, 기존의 트랜잭션 처리는 수초의 응답 시간이 받아들일 수 있는 보통의 응용들에는 충분히 적절하나 초고속을 요구하는 ATM 교환기에서의 호 처리 등 실시간 응용에는 부적당하다[1, 14, 15].

최근 들어 대용량 주기억 장치의 가격이 하락함에 따라 실시간 응용의 처리를 위해 데이타베이스를 주기억 장치에 적재시킨 상태에서 연산을 수행하는 주기억 데이타베이스 시스템에 대한 연구가 많이 진행되고 있다. 따라서 본 연구에서는 실시간 응용에 가장 큰 문제점으로 제기되는 디스크 접근으로 인한 지연 시간을 제거하기 위해 모든 데이타를 주기억 장치에 적재시켜 데이타베이스 연산을 수행하는 주기억 데이타베이스 프로토타입 시스템을 설계하였다.

그러나 주기억 데이타베이스 시스템은 저장 장치가 휘발성인 관계로 시스템 파손 시 전체 데이타를 잃게 된다는 문제점이 발생한다[4, 5, 12]. 이 논문에서는 이런 문제점의 해결을 위해 로그 관리와 검사점 실행 시 측면으로 나누어 개선된 회복 기법을 제안하였다. 로그 관리 측면에서는 그룹 종료 방식을 이용한 로그 관리 기법을 제안하였다. 이 기법은 기존의 기법에 비해 효율적인 로그 관리가 이루어지도록 하였다. 검사점(*checkpoint*)은 퍼지(*fuzzy*) 검사점 기법이 데이타베이스의 불일치가 발생할 수 있다는 단점이 있기 때문에 데이타베이스의 일치성을 보장할 수 있도록 본 연구에서는 퍼지 검사점 기법을 개선하였다.

본 논문의 구성은 제2장에서 본 연구와 관련된 기존의 연구를 설명하고, 제3장에서는 본 연구에서 제안한 주기억 데이타베이스 시스템에서의 회복 기법에 대해 설명한다. 제4장에서는 본 연구를 통해 설계된 주기억 데이타베이스 프로토타입 시스템에 대해 설명한다. 제5장에서는 제안된 회복 기법과 기존의 기법과 비교 분석한다. 마지막으로 제6장에서는 결론과 추후 연구 방향에 대하여 기술한다.

2. 관련 연구

2.1 트랜잭션 종료방식

로그의 저장을 위한 별도의 안정된 메모리가 존재하지 않을 때에는 주로 디스크 로그를 사용하게 된다. 이때 트랜잭션의 종료 방식은 즉시 종료방식(immediate commit), 그룹 종료방식(group commit), 부분 종료방식(precommit) 등으로 구분할 수 있다. 즉시 종료방식은 트랜잭션의 수행이 완료되면 디스크 로그에 즉시 기록하고 트랜잭션을 종료시키는 방식이다. 그룹 종료 방식은 트랜잭션 수행 완료 시 로그를 로그 버퍼에 임시적으로 저장하였다가 버퍼가 다 채워지면 버퍼의 내용을 디스크 로그에 기록하고 트랜잭션들을 함께 종료시키는 방식이며 로그 버퍼에 기록 후 곧바로 록을 해제하는 부분 종료방식이 사용될 수 있다[3, 6, 8, 10].

2.2 검사점 기법

검사점 기법은 크게 트랜잭션 일치, 액션 일치, 퍼지 검사점 기법 등으로 나눌 수 있다. 트랜잭션 일치와 액션 일치는 트랜잭션 수행과 검사점 수행이 동기화되는 방식으로 백업 데이타베이스의 일관성을 유지할 수 있다는 장점이 있지만 검사점 동작의 수행 동안에는 트랜잭션의 처리가 일시 정지되고, 데이타의 일관성 유지로 오버헤드가 증가되는 단점이 있다. 퍼지 기법은 트랜잭션 처리와 검사점 수행이 비동기적으로 수행되므로 시스템의 트랜잭션 처리량을 증가시킬 수 있지만 데이타베이스의 일관성을 유지하지 못하는 문제점이 있다[5, 8-10, 13].

3. 로그와 검사점을 이용한 회복 기법

3.1 로그 구조

본 회복 기법에서는 로그 레코드를 기록하기 위한 자료 구조로서 개인 로그와 재수행 로그, 그리고 디스크 로그를 사용한다. 개인 로그는 각 트랜잭션마다 주기억장치에 할당되는 임시 로그로써 수행중인 트랜잭션의 로그 레코드를 기록하기 위해서 사용된다(<표 1>). 디스크 로그는 수행이 완료된 트랜잭션의 로그를 안정적으로 저장하기 위해 사용된다(<표 2>). 즉, 트랜잭션은 연산의 내용을 디스크 로그에 기록한 후에야 비로소 종료를 할 수 있다. 이와 같이 디스크 로그에 기록된 로그 레코드들은 시스템에 파손이 발생했을 때 재수행 작업을 실시하기 위해서 사용된다.

또한, 트랜잭션 종료 방법으로는 그룹 종료 방식을 사용한다. 즉, 한 트랜잭션의 수행이 끝나면 개인 로그의 내용을 디스크 로그에 즉시 기록하는 것이 아니라 임시 버퍼인 재수행 로그에 기록하고 버퍼가 다 채워지기를 기다린다. 그리고 다른 트랜잭션들이 수행을 완료하여 재수행 로그 버퍼가 다 채워지면 디스크 로그에 기록을 하고 이때에 모든 트랜잭션들이 함께 종료하는 방식이다. 재수행 로그에는 이미 수행을 마친 트랜잭션들의 로그가 기록된다 따라서 개인 로그에서와는 달리 트랜잭션의 새로운 값(new_value)만을 기록한다(<표 2>) 그리고 효율적인 디스크 입출력을 위해서 재수행 로그 버퍼의 크기를 디스크 입출력의 단위인 블록 크기와 동일하게 설정하였다. 이와 같이 그룹 종료 기법을 사용함으로써 트랜잭션의 종료 시마다 매번 디스크를 접근하지 않고 여러 트랜잭션들의 로그를 모아서 한 번에 디스크에 기록하므로 디스크 입출력이 줄어들고 트랜잭션의 수행 시간도 단축시킬 수 있다.

〈표 1〉 개인 로그 레코드 구조

Tid	Type	Rel_name	Page_no	Offset	Old_value	New_value
트랜잭션 ID	로그 레코드의 종류	릴레이션 이름	페이지 번호	페이지 내의 오프셋	데이타 아이템의 이전 값	데이타 아이템의 새로운 값

〈표 2〉 재수행 로그와 디스크 로그 레코드 구조

Tid	Type	Rel_name	Page_no	Offset	New_value
트랜잭션 ID	로그 레코드의 종류	릴레이션 이름	페이지 번호	페이지 내의 오프셋	데이타 아이템의 새로운 값

3.2 로깅 과정

트랜잭션 수행 시 발생하는 로그 레코드를 기록하는 로깅은 다음과 같다

1) 시스템에서는 각각의 트랜잭션이 수행을 요청하면 개인 로그를 할당하여 트랜잭션 수행 중 발생하는 로그 레코드를 기록한다. 이러한 개인 로그 레코드의 구조는 <표 1>과 같다. 개인 로그에는 트랜잭션의 수행으로 인해 변경된 데이타 항목의 이전 값(Old_valuc)과 새로운 값(New_value)을 모두 기록한다. 따라서 트랜잭션의 수행 도중에 사용자의 요청이나 에러 발생에 의해 비정상적으로 종료하게 될 때에는 개인 로그에 기록되어 있는 데이타 항목의 이전 값을 사용하여 트랜잭션의 수행 결과를 취소시킬 수 있다.
2) 트랜잭션의 수행이 완료되면 개인 로그에 기록된 로그 레코드들을 재수행 로그에 기록한다 재수행 로그에 기록하는 시점은 트랜잭션의 수행이 이미 완료된 상태이고 트랜잭션의 비정상적인 종료는 더 이상 발생하지 않으므로 <표 2>에서와 같이 데이타 항목의 이전 값은 기록하지 않고 새로운 값만 기록하여 로그의 크기를 줄이도록 하였다
3) 재수행 로그에 기록한 후 재수행 로그 버퍼가 다 찼다면 디스크 로그에 기록하고 트랜잭션은 그룹 종료를 하게 된다. 그러나 재수행 로그 버퍼가 다 차지 않았다면 디스크 로그로의 기록은 보류되고 트랜잭션은 부분 종료를 하게 된다. 그러나 트랜잭션이 부분 종료하면 데이타 항목에 대해 소유하고 있는 록을 모두 해제하여 이를 다른 트랜잭션에서 사용할 수 있도록 하였다. 부분 종료한 트랜잭션들의 Tid를 부분 종료 리스트(precommit_list)에 기록함으로써 추후에 재수행 로그 버퍼가 모두 차게 되면 이들을 그룹 종료시킬 수 있도록 하였다. 트랜잭션이 부분 종료하고 재수행 로그 버퍼가 모두 차기를 기다리고 있을 경우 만약 더 이상의 수행되는 트랜잭션이 없다면 이 트랜잭션은 종료하지 못하고 무한히 대기하게 되는 문제점이 발생된다. 본 연구에서는 이와 같이 트랜잭션이 무한히 대기하는 경우를 방지하기 위하여 일정 시간이 경과해도 더 이상의 다른 트랜잭션이 수행되고 있지 않다면 재수행 로그 버퍼가 다

차지 않았더라도 재수행 로그를 디스크 로그에 기록하고 트랜잭션을 정상 종료시키는 방법도 함께 고려하였다.

3.3 로깅 알고리즘

트랜잭션이 수행될 때 생성되는 로그 레코드는 다음과 같은 로깅 알고리즘에 의해서 기록된다. 로깅 알고리즘은 트랜잭션의 갱신 동작이 수행중일 때, 트랜잭션의 갱신이 완료되었을 때, 그리고 트랜잭션이 비정상적으로 종료되어 취소될 때의 3가지 경우로 나누어 기술하였다.

```
/******          트랜잭션 갱신 동작중 .          ******/
로그_레코드에 트랜잭션의 갱신을 기록.
if (dirty_tag == 00) {
   해당 페이지를 DPT에 기록, }
   dirty_tag = 10,

/******          트랜잭션 갱신 완료후          ******/
로그_레코드에 트랜잭션의 완료를 기록.
재수행_로그_버퍼의 내용을 로그_레코드에 복사,
dirty_tag = 11.
if (재수행_로그_버퍼가 다 차거나 or 다른 실행중인 트랜잭션이 없으면) {
  재수행_로그_버퍼의 내용을 디스크에 반영;
  카운트를 증가시킴.
  그룹완료()' }
else {
   부분완료().
   재수행_로그_버퍼의 내용이 디스크에 반영이 된 후 완료되면, }

/******          트랜잭션 취소 시          ******/
마지막 로그_레코드를 탐색,
while (첫 번째 로그_레코드를 만날 때까지) {
   if (로그_type이 갱신중이면) {
      이전값을 Data_Item에 복사;
      dirty_tag = 00 (or 11), }
    역 방향 탐색 }
모든 로그_레코드를 제거,
완료'
```

3.4 검사점 수행

트랜잭션의 수행 시 데이타베이스의 갱신은 주기억 데이타베이스에만 반영되고 디스크 데이타베이스에는 반영되지 않는다. 따라서 오랜 시간 동안 트랜잭션들이 수행되면 주기억 데이타베이스와 디스크 데이타베이스 사이에는 많은 부분의 불일치가 발생하게 된다. 또한, 주기억 데이타베이스의 갱신에 대한 정보를 저장하고 있는 디스크 로그의 크기는 매우 방대해 질 것이다. 만약 이러한 때에 시스템에 파손이 발생하여 주기억 데이타베이스의 내용이 손실되면 재수행 동작을 필요로 하는 로그 레코드가 매우 커져 주기억 데이타베이스의 회복 시간은 상당히 길어진다. 이와 같이 문제를 해결하기 위해서는 검사점의 실행이 필요하다.

본 회복 시스템에서는 데이타베이스 페이지마다 갱신 태그를 가지고 있다. 따라서 백업 시에는 이를 참조하여 갱신이 발생한 페이지만 디스크로 백업을 실시하도록 하였다[12]

〈표 3〉 페이지의 갱신 태그

갱신 태그	갱신 태그의 의미
00	페이지가 갱신 안됨
10	페이지의 갱신이 진행중
11	페이지의 갱신이 완료됨

3.5 검사점 수행 기법

본 회복 기법의 검사점 기법으로는 트랜잭션 일치 기법, 액션 일치 기법, 퍼지 기법 중 주기억 데이타베이스 시스템 환경에서 가장 적합한 퍼지 검사점 기법을 기반으로 하였다. 퍼지 검사점 기법은 검사점 수행 중에도 트랜잭션의 처리가 계속 된다는 장점이 있다. 그러나 퍼지 검사점 기법은 주기억 데이타베이스를 백업하는 동안에도 트랜잭션의 수행이 계속되므로 백업되고 있는 주기억 데이타베이스가 일관성 있는 상태임을 보장할 수 없다는 문제점이 있다. 본 연구에서는 데이타베이스의 일관성을 유지할 수 있도록 퍼지 검사점 기법을 수정한 검사점 기법을 제시하였다. 데이타베이스의 일관성에 위배되는 페이지는 백업이 진행중일 때 트랜잭션에 의해 갱신되고 있는 페이지이다. 따라서 갱신이 진행중인 페이지는 갱신이 완료되어 일관된 상태가 될 때까지 디스크로의 백업이 보류되도록 하였다. 이를 위해서 각 데이타 페이지마다 <표 3>과 같은 2비트 갱신 태그를 사용하였다. 이와 같은 갱신 태그를 사용함으로써 페이지의 갱신 여부뿐만 아니라 갱신 동작의 완료 여부까지도 나타낼 수 있다. 즉, 갱신이 전혀 일어나지 않은 페이지는 00값을 가지며, 갱신이 완료된 페이지는 11값을 가진다. 그리고 현재 갱신이 진행 중인 페이지는 10값을 가지도록 하였다. 따라서 주기억 데이타베이스를 디스크로 백업할 때에 11의 갱신 태그 값을 가지는 페이지에 대해서는 즉시 백업을 실시하며, 갱신이 진행 중이어서 10인 값을 가지는 페이지는 갱신이 완료되어 갱신 태그가 11이 되면 비로소 백업을 실시하도록 하였다. 본 회복 기법에서는 위와 같은 검사점 동작이 발생하는 조건으로 다음과 같은 두 가지를 설정하였다. 두 가지 조건 중 하나가 만족되면 검사점 신호가 발생되고 검사점 프로세스

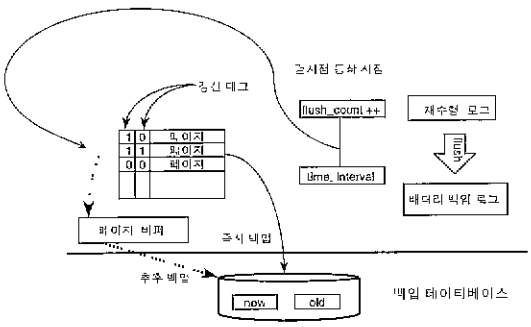
가 호출되어 트랜잭션 처리 프로세스와 독립적인 검사점 동작을 수행하게 된다.

1) 재수행 로그의 디스크 기록 횟수에 의해
2) 이전 검사점 수행 시작부터 경과된 시간에 의해

재수행 로그가 디스크에 기록된 횟수가 많을수록 주기억 데이타베이스에 많은 갱신이 일어났음을 의미하고 이때 백업 동작이 수행되도록 하였다 이를 위해서 재수행 로그가 디스크에 기록될 때마다 flush_count라는 변수의 값을 1씩 증가시킨다. 이 값이 CHECKPOINT_FLUSH_COUNT라는 시스템에서 미리 정의한 상수 값에 도달하면 검사점 신호가 발생하도록 하였다. 또한 트랜잭션의 수행이 많이 발생하지 않아서 너무 오랫동안 검사점이 수행되지 않는 경우를 방지하기 위해서 이전 검사점 수행 시작부터 경과한 시간에 의해서도 검사점 신호가 발생하도록 하였다. 이를 위해서 검사점 동작이 시작될 때마다 시각을 기억해 놓는다 검사점 동작이 완료되면 트랜잭션 처리 동작중 주기적으로 현재 시각을 검사하여 시간 간격이 CHECKPOINT_TIME_INTERVAL이라는 상수 값 이상이 되면 역시 검사점 신호가 발생하도록 하였다.

3.6 검사점 수행 과정

검사점 수행은 (그림 1)과 같이 동작한다.



(그림 1) 검사점 수행

1) 검사점 조건 두 가지 중 하나가 만족되면 검사점 신호가 발생한다. 이때 트랜잭션 관리기는 검사점 동작을 수행할 자식 프로세스를 생성한다. 이와 같이 생성된 검사점 관리기는 먼저 현재까지 기록된 재수행 로그를 디스크로 기록하며 검사점 동작의 시작을 나타내는 <CHECKPOINT_START> 로그 레코드를 추가한다.
2) 검사점 관리기는 갱신된 페이지 정보를 저장하고 있는 DPT(Dirty Page Table)의 내용을 Temp_DPT로 복사한다 검사점 관리기는 백업 동작 시 Temp_DPT를 참조한다. DPT에 기록된 내용들을 전부 삭제하며 트랜잭션 처리 시 새롭게 갱신되는 페이지 정보를 기록하기 위해서 트랜잭션 관리기에 의해 사용된다.
3) Temp_DPT가 가리키는 모든 데이타베이스 페이지에 대하여 갱신 태그가 11이면 디스크로 즉시 백업을 하고 갱신 태그가 10이면 갱신이 원료될 때까지 기다렸다가 백업을 수행한다
4) 백업이 완료되면 <CHECKPOINT_END> 로그 레코드를 디스크에 추가하여 검사점 동작이 완료되었음을 기록한다. 그리고 갱신된 페이지의 백업으로 인해 더 이상 불필요한 디스크 로그 레코드를 삭제한다 즉, 검사점 동작이 시작될 때 기록했던 <CHECKPOINT_START> 로그 레코드 보다 먼저 기록된 모든 로그 레코드들을 모두 삭제시킨다. 불필요한 로그 레코드의 삭제가 끝나면 모든 검사점 동작이 완료하게 된다.

본 연구에서는 위의 4)에서와 같이 모든 페이지의 백업이 완료된 후에 디스크 로그를 삭제하므로 만약 일부 페이지만 백업되었을 때 시스템에 파손이 발생한다면 이미 백업된 페이지에 대해서도 재수행 작업을 하게 된다.

3.7 검사점 수행 알고리즘

검사점 신호가 발생하여 주기억 데이타베이스의 갱신된 내용을 디스크로 백업하고 더 이상 필요 없는 디스크 로그를 삭제하는 알고리즘은 다음과 같다

```
/* 알고리즘                검사점 수행과 백업 알고리즘          */
/* 반영 횟수가  CHECKPOINT_FLUSH_COUNT보다 크거나
  time 주기가 CHECKPOINT_TIME_INTERVAL보다 클 때 */
재수행_로그_버퍼의 내용을 반영,
반영 횟수를 0으로 초기화,
검사점 시간을 기록,
<CHECKPOINT_START>를 로그_레코드에 기록,
move(Temp_DPT, DPT),
/* DPT는 트랜잭션 처리 프로세스가 트랜잭션 처리를 위해 사용하고,
  Temp_DPT는 검사점 프로세스가 백업을 위해 사용함 */
while (만약 Temp_DPT가 존재하면) {
        /* Temp_DPT 엔트리가 가리키는 페이지에 대해서 */
  if (dirty_tag == 11) {
```

```
    페이지에 록을 걸고'
    페이지 내용을 버퍼에 기록,
    페이지의 록을 해제,
    버퍼의 내용을 디스크에 기록,
    dirty_tag = 00;
    Temp_DPT를 삭제,
   )
)
<CHECKPOINT_END>를 로그_레코드에 기록'
<CHECKPOINT_START> 이전에 발생된 모든 로그_레코드의 기록을 제거
```

4. 주기억 데이타베이스 프로토타입 시스템 설계

주기억 데이타베이스 시스템은 기존의 데이타베이스 시스템과는 다르게 주기억장치에 모든 데이타를 적재시킨 상태에서 각종 연산을 수행한다. 이와 같은 근본적인 차이점을 고려하여 본 연구에서는 주기억 데이타베이스 프로토타입 시스템을 UNIX Workstation으로 구성된 클라이언트/서버 환경에서 C 언어를 이용하여 설계하였으며, 사용자 인터페이스를 구축하기 위하여 SQL subset을 질의 모델로 채택하여 질의 처리기를 설계하였다

4.1 주기억 DBMS의 기본 구성

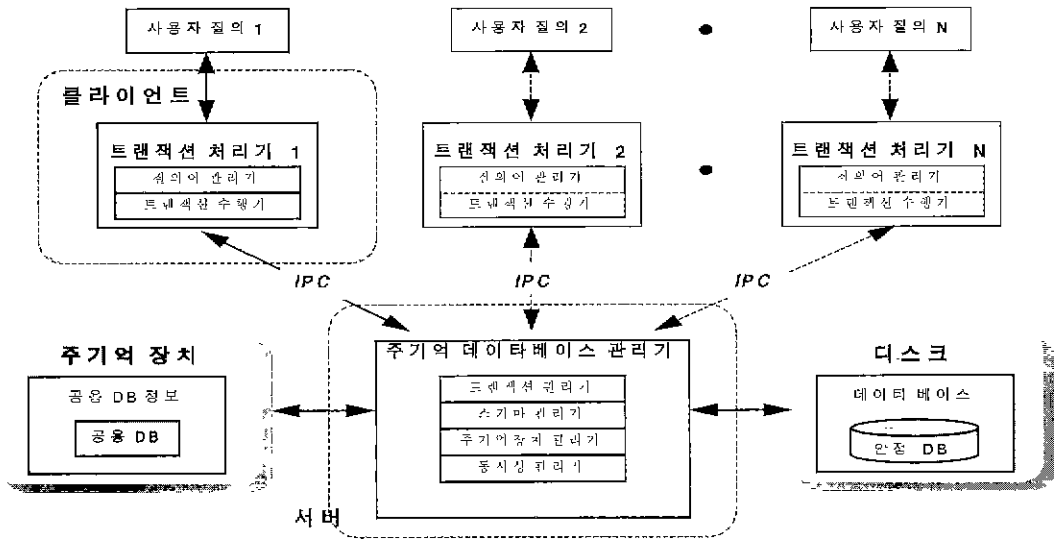
클라이언트인 트랜잭션 처리기와 서버인 주기억 데이타베이스 관리기로 구성되는 주기억 데이터베이스 프로토타입 시스템의 전반적인 구조는 (그림 2)와 같다. 클라이언트인 트랜잭션 처리기는 사용자의 질의를 입력받아 문법적 오류를 검사하고, 주기억 데이타베이스 관리기에게 트랜잭션 수행을 요청한다. 질의의 수행의 대부분인 질의 처리를 클라이언트인 트랜잭션 처리기가 수행하여 클라이언트의 처리율을 높이고, 서버인 주기억 데이타베이스 관리기의 작업 부하를 감소시킨다. 서버인 주기억 데이타베이스 관리기는 트랜잭션 관리기, 스키마 관리기, 주기억장치 관리기, 그리고 동시성 관리기로 구성된다 주기억 데이타베이스 관리기는 시스템 구동 시, 디스크의 모든 데이타베이스를 주기억장치에 적재하여 주기억 데이타베이스를 구성 및 관리하고, 트랜잭션 처리기와 정보 이동을 위한 트랜잭션 큐를 초기화한다.

4.1.1 트랜잭션 처리기

질의어 관리기와 트랜잭션 수행기로 구성되는 트랜잭션 처리기는 사용자 질의를 입력받아 문법의 정확성 여부를 검사하고, IPC를 통하여 주기억 데이타베이스 관리기에게 트랜잭션 수행을 요청한다. 또한, 주기억 데이타베이스 관리기가 수행한 결과를 트랜잭션 큐에서 진달받아 사용자에게 전달한다. 본 논문에서는 트랜잭션 수행기에 대해서만 설명한다.

- 트랜잭션 수행기

트랜잭션 수행기는 트랜잭션 분류기, 트랜잭션 우선



(그림 2) 주기억 데이타베이스 프로토타입 시스템의 구조

순위 할당기, 그리고 트랜잭션 수행 요청기로 구성된다. 트랜잭션 수행기는 스케쥴링하려는 트랜잭션의 마감시간 이내 수행 여부를 검사한다. 그리고, 마감시간 이내에 수행 가능한 트랜잭션과 마감시간 이내에 수행 불가능한 트랜잭션 각각에 적절한 우선순위를 할당하고, 각 트랜잭션을 트랜잭션 큐에 등록하여 필요한 데이타를 주기억 데이타베이스 관리기에게 요청한다. 트랜잭션 분류기는 발생된 트랜잭션의 예측 및 실행 시간 등의 정보를 이용하여 발생 트랜잭션이 마감시간 이내에 수행할 수 있는지 여부를 검사한다. 즉, 발생 트랜잭션을 마감시간 이내에 수행 가능한 트랜잭션과 마감시간 이내에 수행 불가능한 트랜잭션으로 분류한다 트랜잭션 수행 요청기는 트랜잭션 우선순위 할당기에 의해 할당된 트랜잭션의 우선순위에 따라 해당하는 트랜잭션 큐에 수행 요청을 등록한다. 그리고, 주기억 데이타베이스 관리기의 트랜잭션 관리기는 트랜잭션 큐에 등록한 요청 트랜잭션의 수행 결과를 트랜잭션 처리기에 전달한다.

4.1.2 주기억 데이타베이스 관리기

주기억 데이타베이스 관리기는 트랜잭션 관리기, 스키마 관리기, 주기억장치 관리기, 그리고 동시성 관리기로 구성된다 주기억 데이타베이스 관리기의 기능은 다음과 같다. 첫째, 시스템 구동 시 사용자의 질의 수행에 필요한 트랜잭션 큐를 생성 및 관리한다 둘째, 본 시스템의 주기억 데이타베이스 적재 구조에 맞게 디스크의 모든 데이타베이스 화일을 주기억장치에 적재한다. 셋째, 트랜잭션 처리기가 데이타베이스를 접근할 때 제안한 우선순위를 기반으로 한 이중 록킹 기법으로 데이타베이스의 일관성을 유지하면서 트랜잭션의 요청을 수행한다. 끝으로, 시스템 종료시 주기억장치에 있는 모든 데이타베이스를 디스크로 저장한다.

4.1.2.1 트랜잭션 관리기

트랜잭션 관리기는 시스템 구동 시에 트랜잭션 수행 요청과 트랜잭션 수행 결과를 등록하는 트랜잭션 큐를 생성 및 관리한다. 트랜잭션 큐는 마감시간 이내에 처리가 가능한 트랜잭션들이 대기하는 적중(hit) 트랜잭션 큐와 마감시간 이내에 처리가 불가능한 트랜잭션들이 대기하는 초과(miss) 트랜잭션 큐로 구성된다. 가장 높은 우선순위를 지닌 적중 큐에서 트랜잭션 수행 요청 유무를 검사한다 트랜잭션 수행 요청이 있는 경우 요청 트랜잭션을 수행한다. 다음으로 가장 높은 우선순위를 지닌 트랜잭션 큐에서 점차 낮은 우선순위를 지닌 트랜잭션 큐로 트랜잭션 수행 요청 유무를 검사한다. 모든 적중 트랜잭션 큐에 더 이상 트랜잭션 수행 요청이 없는 경우, 초과 큐에 요청한 트랜잭션을 수행한다. 이 때, 적중 큐에서와 마찬가지로 높은 우선순위를 지닌 초과 큐에서 낮은 우선순위를 지닌 초과 큐의 순서로 트랜잭션 수행 요청을 인지한다. 트랜잭션 관리기가 사용하는 정보 테이블은 <표 4>와 같다.

〈표 4〉 트랜잭션 정보 테이블

변 수	내 용
TR_ID	트랜잭션 식별자
A(T)	트랜잭션의 도착 시간
D(T)	트랜잭션의 마감시간
SR(T)	트랜잭션이 서비스 받은 시간
E(T)	트랜잭션의 예측 실행 시간
Now(T)	현재 시간
Tr_Tag	트랜잭션 분류 태그
Priority(T)	트랜잭션의 우선 순위

4.1.2.2 스키마 관리기

스키마 관리기는 시스템 카탈로그와 사용자 릴레이션을 생성 및 관리한다. 시스템 카탈로그는 릴레이션 카탈로그, 애트리뷰트 카탈로그로 구성된다. 릴레이션 카탈로그는 시스템이 정의한 릴레이션과 사용자가 생성한 릴레이션에 대한 모든 정보를 유지한다. 애트리뷰트 카탈로그는 릴레이션 카탈로그 내에 등록된 릴레이션에 속한 애트리뷰트에 대한 모든 정보를 저장한다 시스템 카탈로그, 애트리뷰트 카탈로그의 구성은 각각 <표 5>, <표 6>과 같다.

〈표 5〉 릴레이션 카탈로그의 구성

Rel_Name	Attri_Name	Type	Length	Index
Char(20)	Char(10)	Char(10)	Int	Int

〈표 6〉 애트리뷰트 카탈로그의 구성

Rel_Name	Creator	Permission	Deriv	Attri #
Char(20)	Char(10)	Char(3)	Char(20)	int

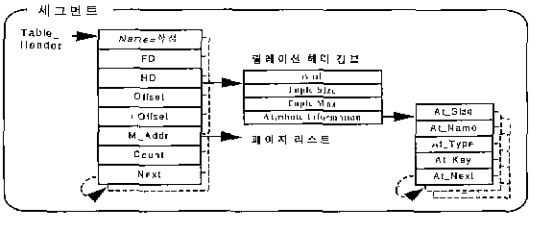
릴레이션 카탈로그에는 릴레이션의 이름을 나타내는 Rel_Name, 릴레이션 생성자의 ID를 나타내는 Creator, 접근 허가에 대한 정보를 나타내는 Permission, 릴레이션의 유형, 즉, 기본(base) 릴레이션과 뷰(View) 릴레이션을 구분하는 Deriv, 그리고 릴레이션에 포함된 애

트리뷰트의 수를 나타내는 Attri #로 구성된다.

애트리뷰트 카탈로그는 해당 애트리뷰트를 포함하고 있는 릴레이션의 이름을 나타내는 Rel_Name, 어떤 릴레이션에 포함된 특정한 애트리뷰트의 이름을 나타내는 Attri_Name, Integer, Real, 그리고 Character String를 나타내 Type, 애트리뷰트의 길이를 나타내는 Length로 구성된다.

4.1.2.3 주기억장치 관리기

주기억장치 관리기가 사용자 테이블을 주기억장치로 적재하여 관리하는 저장 공간의 단위로는 세그먼트와 페이지가 있다 하나의 사용자 릴레이션은 하나의 세그먼트에 저장되고, 각 세그먼트는 세그먼트 정보 블록, 그리고 페이지 리스트로 구성된다 세그먼트 정보 블록 구조는 (그림 3)과 같다.

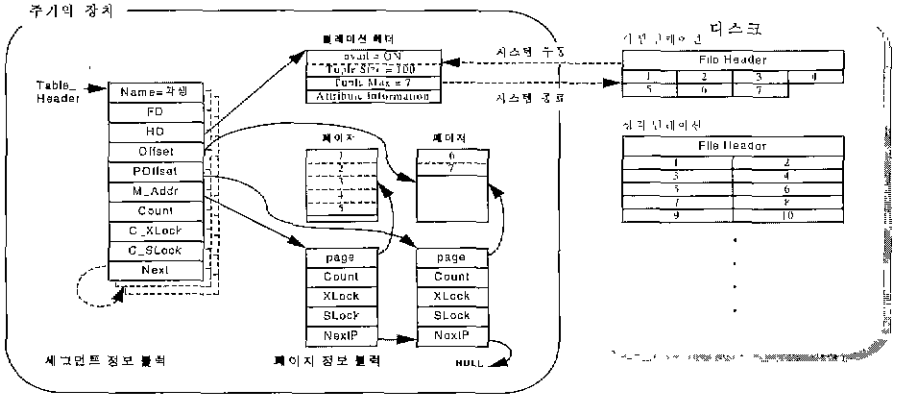


(그림 3) 세그먼트 정보 블록 구조

디스크에 있는 사용자 릴레이션은 튜플 길이 단위로 주기억장치의 해당 세그먼트 내 페이지 리스트의 여러 페이지로 나누어 순서대로 적재된다. 페이지 정보 블록에는 해당 페이지의 영역을 가리키는 포인터(page), 페이지내의 튜플 수를 나타내는 값(Count) 그리고 다음 페이지를 가리키는 포인터(NextP)등 각 페이지에 대한 정보가 저장된다 각 페이지는 512Byte 크기의 공간으로써 사용자 릴레이션을 주기억장치로 적재 시 동적으로 할당된다. 사용자가 트랜잭션 처리기를 통하여 튜플을 삽입할 때 페이지가 동적으로 할당되고, 사용자가 트랜잭션 처리기를 통하여 튜플을 삭제할 때 페이지가 동적으로 반환된다

주기억장치 관리기의 주요 기능은 다음과 같다. 첫째, 시스템 구동 시 모든 시스템 카탈로그와 사용자 릴레이션을 디스크로부터 주기억장치로 적재하여 주기억 데이타베이스를 구성한다. 둘째, 시스템 구동 시 릴레이션을 디스크에서 주기억장치로 적재하기 위해 세그먼트 정보 테이블을 할당하여 릴레이션에 대한 정보를 구성하고, 필요에 따라 페이지를 할당하여 릴레이션의 튜플을 주기억장치로 적재한다. 사용자가 튜플 삽입의 질의를 요청한 경우, 세그먼트내의 삽입 위치를 검사하여 페이지의 유효 공간이 없거나 페이지내의 유효 공간이 삽입 튜플의 길이보다 짧을 경우 동적으로 페이지를 할당하여 페이지 리스트에 연결한 후 새로운 페이지에 튜플을 삽입한다. 셋째, 시스템 종료시 주기억 데이타베이스의 모든 릴레이션을 디스크에 저장한다. 먼저 세그먼트 정보 블록의 릴레이션 헤더 정보를 디스크에 저장한 후 페이지 리스트에서 순서대로 페이지내의 튜플을 디스크에 저장한다 릴레이션의 저장이 끝나면 세그먼트 정보 블록과 세그먼트내의 모든 페이지 영역 그리고 페이지 정보 블록을 반환한다. 이와 같은 방법으로 주기억 데이타베이스내의 모든 릴레이션을 디스크에 저장한다.

주기억장치 관리기가 디스크의 릴레이션을 주기억장치로 적재하는 구조는 (그림 4)와 같다.



(그림 4) 릴레이션의 주기억장치 적재 구조

4.2 질의 수행 예

4.2.1 검색 질의

사용자가 "select all from STUDENT;"의 질의 수행을 요청한 경우, 트랜잭션 처리기가 어휘 분석기, 구문 분석기, 그리고 질의어 관리기를 통해 사용자의 입력 질의에 대해 문법에 맞는지 오류 유무를 검사하고, 구문 트리를 구성한다. 문법적인 오류와 시스템 카탈로그에 위반 사항이 없는 경우, 트랜잭션 처리기의 트랜잭션 분류기가 발생 트랜잭션이 마감시간 이내에 수행 가능한지를 검사한다.

위의 예제 질의는 마감시간 이내에 수행 가능하고 발생 트랜잭션의 우선 순위를 2라 가정한다. 그러므로 트랜잭션 수행 요청기는 적중 큐2에 발생 트랜잭션의 수행 요청을 등록한다. 트랜잭션 처리기가 트랜잭션 수행기를 통해 튜플의 검색 수행 요청을 적중 큐2에 등록하면 트랜잭션 관리기는 주기억장치 관리기를 호출하여 튜플을 검색하고, 실행 결과를 적중 큐2를 통해 트랜잭션 처리기에게 전달한다. 트랜잭션 처리기는 트랜잭션 관리기로부터 받은 튜플이 사용자가 입력한 조건에 맞는지 여부를 검사하여 유효한 경우 결과 버퍼에 저장한다. 계속 반복하여 릴레이션 끝까지 검색한 후 결과 버퍼에 저장된 튜플을 트랜잭션 수행 결과로써 사용자에게 전달한다.

4.2.2 삽입 질의

사용자가 "insert STUDENT((ID=95410001), (NAME = "BAEK"));"의 질의 수행을 요청한 경우, 트랜잭션 처리기가 어휘 분석기, 구문 분석기, 그리고 질의어 관리기를 통해 사용자의 입력 질의에 대해 문법에 맞는지 오류 유무를 검사하고, 구문 트리를 구성한다. 트랜잭션 처리기가 트랜잭션 수행기를 통해 튜플의 삽입 수행 요청을 적중 큐2에 등록하면 트랜잭션 관리기는 주기억장치 관리기를 호출하여 튜플 삽입 위치에 튜플을 삽입하고 해당 페이지 정보 테이블의 튜플 수를 증가시킨다. 이 때, 동시성 관리가 호출되어 트랜잭션이 실행을 완료할 때까지 튜플을 삽입하려는 페이지에 독점 록을 소유하도록 하여 다른 트랜잭션이 같은 페이지 내에서 충돌이 발생하는 연산을 수행하지 못하도록 한다. 트랜잭션의 수행이 완료되면 트랜잭션 관리기가 트랜잭션 수행 결과를 적중 트랜잭션 큐2에 등록한다. 트랜잭션 처리기는 트랜잭션 수행 결과를 질의어 관리기에게 전달한다.

4.2.3 삭제 질의

사용자가 "delete STUDENT where NAME="BAEK";" 질의를 요청했다고 가정한다. 또한, 이 예제 트랜잭션은 마감시간 이내에 수행이 가능하고, 발생 트랜잭션의 우선순위를 2라 가정한다. 트랜잭션 처리기가 트랜잭션 수행기를 통해 사용자 튜플의 삭제 수행 요청을 적중 큐2에 등록하면, 트랜잭션 관리기는 주기억장치 관리기를 호출하여 튜플을 삭제하고, 해당 페이지 정보 테이블의 튜플 수를 감소시킨다. 튜플의 삭제는 튜플의 유효성을 나타내는 유효 필드를 "On"에서 "Off"로 바꿈으로 튜플의 삭제가 이루어진다. 이 때 동시성 관리가 호출되어 트랜잭션이 실행을 완료할 때까지 튜플을 삭제하려는 페이지에 독점 록을 소유하도록 하여 다른 트랜잭션이 같은 페이지 내에서 연산을 수행하지 못하도록 한다. 트랜잭션의 수행이 완료되면, 트랜잭션 관리기가 트랜잭션 수행 결과를 적중 큐2에 등록한다. 이때, 트랜잭션 처리기는 트랜잭션 수행 결과를 사용자에게 전달한다.

5. 실험 평가

모의 실험 평가의 목적은 본 연구에서 제안된 기법과 기존의 기법을 비교 분석하는 것이다. 제안된 기법과 시스템 파손 시 회복에 중점을 둔 전략을 실험하기 위하여 실험 모델을 설계하고, 이를 SLAM II 언어를 사용하여 시뮬레이션 하였다. 그리고 실험 모델에 대하여 취할 기법과 기본 매개 변수를 설정하고, 설정된 매개 변수를 사용하여 다음과 같은 순서로 진행한다. 먼저 기존의 기법인 Lehman, Eich, Salem등의 기법들을 본 연구 환경에 맞도록 변화시켜 로깅 기법과 검사점 실행 시기에 따른 로드 및 검사점 실행 시기에 따른 회복 시간의 변화 등으로 나누어 분석 평가하였다.

5.1 실험 환경

본 연구에서 제안한 회복 기법 알고리즘은 SLAM II를 이용하여 구현하였다[2]. 제안된 회복 모델은 SLAM II의 모델로 표현하였다. 실험은 Pentium MMX 400MHz WINDOW 98의 환경에서 이루어 졌다. SLAM II는 자체의 처리기와 클럭을 가지고 있기 때문에 실험 시스템의 속도에는 아무런 영향을 받지 않는다. 성능 분석 모델은

이 논문에서 제안한 시스템 환경을 기초로 하여 각 시뮬레이션 단계마다 약간의 변형과 매개 변수에 변화를 주어 실행하였다.

5.2 실험 가정

5.2.1 실험 전략

본 논문에서 실험 모델에 채택하는 기법 및 환경은 다음과 같다. 로깅은 개인 로그와 로그 버퍼(재수행, 철회수행), 디스크 로그, 배터리 백업 로그를 가지고 있는 경우로 나누어 적용한다. 로그 실행도 재수행과 철회수행으로 나누고, 지연 갱신 기법과 즉시 갱신 기법으로도 나누어 비교 분석하였다. 또한 트랜잭션의 종료도 그룹 종료나 부분 완료 등으로 나누어 분석하였다.

검사점 수행도 트랜잭션, 액션, 퍼지 검사점의 경우와 본 연구에서 제안하는 개선된 검사점 기법으로 나누어 분석하였다

회복 시 사용하는 프로세스의 경우도 회복을 위해 전담 프로세서를 사용하는 경우와 정상 처리와 회복 처리에 같은 프로세서를 사용하는 경우로 볼 수 있다. 그러나 본 비교 분석에서는 하나의 프로세서만을 사용한다고 가정하였다

먼저 로깅 기법의 경우도 기존의 연구 기법중 대표적으로 Lehmann, Eich와 본 기법을 비교 분석하였다. Lehmann의 경우는 로그의 구성이 SLB라는 로그 버퍼와 SLT로 구성되고 디스크 로그를 가지고 있고, 그룹 종료 기법을 사용하며 즉시 갱신 기법을 사용하여 로깅을 하였다. Eich의 경우는 로그를 사전 이미지와 사후 이미지로 나누어 관리하면서 디스크 로그를 두고 로그 버퍼를 사용하지 않았으며, 즉시 갱신 기법을 사용하는 것은 Lehmann과 같으나 체도우 기법을 로깅에 함께 적용하여 사용하였다. 본 연구는 개인 로그와 재수행 로그를 두었다. 그리고 로깅 기법에서는 지연 갱신 기법을 사용하였다.

디스크 데이타베이스 사본도 본 연구는 지연 갱신 기법을 사용하며 하나의 사본 유지를, Lehmann의 경우는 즉시 갱신 기법을 사용하며 하나의 사본을 사용하는데 비하여, Salem의 경우는 즉시 갱신 기법과 핑퐁 기법을 사용하였고, Eich는 즉시 갱신 기법과 체도우 기법을 이용하였다.

검사점 기법의 경우도 트랜잭션 일치 기법을 사용하는 Lehmann, 퍼지 검사점 기법을 사용하는 Salem과 Eich로 구분 할 수 있다. 본 연구는 퍼지 검사점 기법을 개선한 퍼지 검사점 기법을 제안하여 사용하였다.

Salem과 본 연구는 하나의 프로세서를 사용하나 Lehmann과 Eich는 두 개의 프로세서를 사용하였다. 그러나 본 비교 분석 환경에서는 하나의 프로세서만 사용한다고 가정하였다.

트랜잭션 처리 및 회복에 대한 일반적인 동작은 트랜잭션의 실행이 시작되면서 각각의 트랜잭션이 생성한 로그 레코드를 주기억 장치내의 개인 로그에 기록하면서 시작된다. 개인 로그는 트랜잭션 처리 과정에서 철회수행과 재수행으로 나누어 기록된다. 개인 로그는 트랜잭션이 정상 완료될 경우 로그 버퍼나 배터리 백업 로그에 저장되고, 일정 시간이 경과하거나 배터리 백업 로그 또는 로그 버퍼가 모두 차면 검사점을 실행하여 디스크 데이타베이스 사본으로 데이타베이스 백업을 실시하며 기존의 기법에서는 로그도 디스크 로그로 백업을 실시한다. 그리고 시스템 파손이 발생하면 디스크 데이타베이스 내용을 재적재 기법에 따라 주기억장치로 재적재 한 뒤 배터리 백업 로그를 이용하여 데이타베이스의 내용을 파손 이전의 일치된 상태로 복구한다.

5.2.2 실험 매개 변수

실험에 사용되는 매개 변수는 [7, 9, 11]에서 제안된 값을 기초로 하였다.

대표적인 매개 변수는 <표 7>, <표 8>과 같다. SLAM II는 동적 데이타 구조를 지원하지 않고 배열

<표 7> 매개 변수

매개변수	값
로그 페이지 크기	1024 words
전송 시간	7.46ms
주 기억 장치 버퍼 복사 시간	1.1738ms
페이지 접근 시간	21.76ms
입출력 오버헤드	1000 instructions
입출력 지연 시간	0.03 seconds
데이타베이스 크기	256Mwords
레코드 크기	32 words
세그먼트 크기	8192 words
트랜잭션 로그 오버헤드	32 words
갱신 연산 수	5 rec/TRs
워드당 바이트	4 bytes
트랜잭션 실패율	0.05
페이지 크기	1800

로 모든 데이타 구조를 표현한다. <표 8>에 표시된 1차원 배열 xx는 전역 데이터로 사용된다.

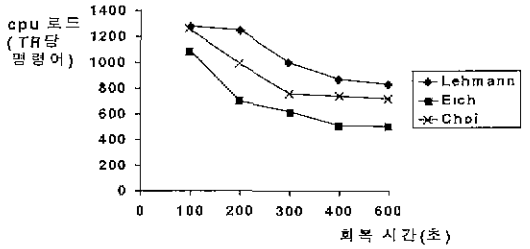
그리고 다음과 같이 기본 가정을 하였다. 주기억 장치에 있는 베터리 백업 로그나 로그 버퍼의 크기는 200Kb로 가정하였다. 트랜잭션 도착율은 초당 450개의 트랜잭션이 연속적으로 도착한다고 가정하며, 이 가정을 균등 분포(uniform distribution)로 나타내었다. 또한 하나의 트랜잭션이 하나의 로그 레코드를 생성하며 한 번에 하나의 트랜잭션이 실행된다고 가정한다. 그리고 각각의 시뮬레이션마다 제한적으로 더 필요한 가정은 각 절에서 추가하여 설명한다.

〈표 8〉 전역 매개 변수

변 수	의 미	값
xx(1)	트랜잭션처리시간 + 로그생성시간	2 ms
xx(2)	주기억 상치 로그 버퍼의 full 여부 체크(로그 버퍼 flush 시기 결정)	0(초기값)
xx(3)	디스크 로그 화일의 full 여부 체크 (백업 시기 결정)	0(초기값)
xx(31)	하나의 로그 버퍼의 로그 레코드 갯수	500 개
xx(32)	로그 디스크에 쓰는 시간	87.8 ms xx(35)+xx(36)-xx(37)
xx(34)	로그 버퍼의 크기	200*500bytes
xx(35)	평균 탐색 시간	16 ms
xx(36)	평균 지연 시간	8.3 ms
xx(37)	전송 시간	63.5 ms
xx(40)	디스크의 로그 버퍼를 주기억 장치로 읽어들이는데 걸리는 시간	87.8 ms
xx(41)	로그 분석 + 재수행 + 철회수행	2 ms
xx(42)	재수행	1 ms
xx(43)	트랜잭션 도착 비율	450 트랜잭션/초
xx(45)	계직제 시간	6380ms

5.3 로깅 기법을 이용한 회복 시간 로드 비교

여기서는 본 연구의 환경에 Lehmann과 Eich 로깅 기법을 적용하여, 회복 시간에 따른 CPU 로드를 비교 분석하였다. Lehmann과 Eich의 기법은 안정된 로그를 사용하고 디스크 로그를 사용하나 본 기법에 적용하면서 디스크 로그는 사용하지 않고, 단지 로깅 기법의 차이만 적용하였다. 본 연구는 로깅을 사용시 지연 갱신 기법을 사용하나, Lehmann의 경우는 즉시 갱신 기법을 사용하였다 또한 Eich의 경우는 세도우를 사용하면서 즉시 갱신 기법을 사용하였다 (그림 5)는 회복 시간(초 단위)이 0에서 600까지 증가됨에 따른 CPU 로드를 나타낸 것이다 본 비교에서는 데이타베이스의 백업에 대해서는 고려하지 않고, 로깅에 따른 로드만을 고려하였다.



(그림 5) 회복 시간 변화에 따른 CPU 로드(로깅 기법)

이 실험 결과 전체적인 기법들이 회복 시간이 늦어짐에 따라 CPU의 로드가 감소됨을 알 수 있다. 그리고 로깅 기법에 의한 로드의 분석은 로깅 기법에 크게 의존하지 않는 Eich가 제안한 주기억 데이타베이스 로깅 기법이 Lehmann이나 본 기법에 비해 뛰어난 것으로 분석되었다 본 기법은 Lehmann의 기법에 비해서는 약간 우세 한 것으로 분석되었다. Eich 기법이 본 기법이나 Lehmann 기법에 비해 로깅 성능이 훨씬 뛰어난 이유는 Eich의 경우는 세도우 기법를 사용하기 때문에 로그의 철회수행 처리가 필요하지 않기 때문이다 본 기법이 Eich의 기법에 비해 성능은 떨어지나 Lehmann에 비해 성능이 좋게 평가된 것은 철회수행로 그는 처리하지 않지만 정상완료 시까지 기다리는 로드가 포함되기 때문이다.

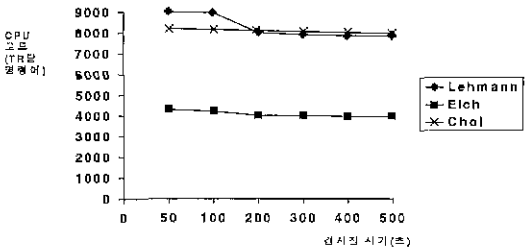
그러나 만약 본 기법이 뒤에 제안하는 주기억 데이타베이스 시스템으로 시스템을 구성하여 디스크 로그를 사용하지 않고 베터리 백업 로그를 사용한다면 디스크 입출력의 감소로 Eich의 기법과 거의 같은 효과를 가질 수 있다(한 로그 레코드의 디스크 기록 시간과 백업시 발생되는 디스크 로그의 재적제 시간이 <표 7>, <표 8>에서 보면 각각 87.8 ms로 같기 때문이다). 즉, 본 연구는 개인 로그와 재수행 로그를 두어 재수행 로그에는 정상 완료된 트랜잭션들만 기록한다 또한 로그 분석과 철회수행. 재수행 단계로 나누어진 로깅 단계가 로그 분석과 재수행 단계로 이루어지기 때문에 빠른 회복이 가능하다.

위 시뮬레이션을 기초로 하여 로그 버퍼 크기가 같고, 트랜잭션 발생율이 같다고 가정할 경우 처리할 수 있는 트랜잭션의 양을 고려해보면, 본 연구 기법이 다른 기법에 비해 재수행 로그 레코드만이 기록되기 때

문에 철회 수행 로그와 재수행 로그 둘 다 저장하는 기존 기법보다 처리 시간이 단축된다. Eich의 경우는 체도우를 사용하기 때문에 로드가 많이 감소되는 것도 볼 수 있다.

5.4 검사점 실행 시기에 따른 로드 비교

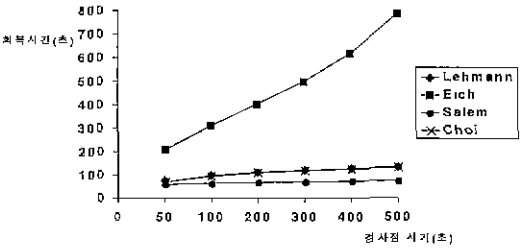
(그림 6)은 검사점을 실행하는 시기의 변화에 따른 CPU 로드는 검사점의 시기가 늦어짐에 따라 모든 기법이 거의 비슷하게 감소되는 것을 볼 수 있다. 그 이유는 검사점 발생 시기가 길어짐에 따라 CPU의 로드가 적어지기 때문이다(검사점이 자주 일어나면 CPU의 로드는 상대적으로 커진다). 본 기법의 경우 개선된 검사점 기법이 페이지 버퍼에 일단 갱신된 데이타 페이지를 저장하였다가 디스크에 기록을 하기 때문에 다른 기법에 비해 약간의 로드 증가를 볼 수 있다. Eich의 경우는 체도우 기법을 사용하면서 피지 검사점 기법을 사용하기 때문에 다른 기법에 비해 로드가 상당히 감소하는 것을 볼 수 있다. 그러나 이 기법은 데이타의 일치성을 보장할 수 없기 때문에 회복시 문제가 발생된다. Lehmann 기법은 데이타의 일치성을 보장하기 위해 트랜잭션 일치 검사점을 사용하므로 비용의 증가 및 이를 위한 기억 공간의 낭비로 많은 로드가 부가되는 것을 볼 수 있다.



(그림 6) 검사점 실행 시기에 따른 CPU 로드

5.5 검사점 실행 시기 결정

다음의 (그림 7)은 검사점 시기 변화에 따른 회복 시간을 실험 평가하였다. Salem의 경우는 핑퐁 기법을 사용하면서 피지 검사점 기법을 이용하여 회복 시간이 일정하나 로그 버퍼를 사용하지 않고 체도우를 사용한 Eich의 경우는 검사점이 늦어짐에 따라 오버헤드(체도우의 포인터 관리, 로그 처리)가 증가되어 좋지 않은 결과가 나왔음을 알 수 있다.



(그림 7) 검사점 시기 변화에 따른 회복 시간

6. 결 론

실시간 시스템에서 디스크 베이스 시스템으로 운용될 경우 디스크 입출력으로 인해 빠른 응답을 할 수 없는 문제가 있다. 본 논문에서는 이를 보완하기 위해 시스템이 환경을 주기억 데이타베이스 시스템으로 구성하여 이에 따른 프로타입을 설계해 보았다. 그리고 주기억 데이타베이스 시스템이 가지고 있는 문제점인 회복에 대해서도 로그 측면과 검사점 기법으로 나누어 로그 관리 측면에서는 그룹 종료와 재수행 만을 수행함으로써 빠른 로그 분석을 통한 회복이 이루어지도록 하였다. 또한 검사점 수행에서도 기존의 피지 검사점이 가지고 있는 데이타의 비일관성 문제를 갱신 태그를 사용하여 처리함으로써 일관성을 유지하도록 하였다. 이를 검증하기 위해 SLAM II를 이용하여 기존의 연구와 비교 분석하였다. 하지만 설계된 주기억 데이타베이스 프로토타입 시스템의 구현하기 위해서는 질의 분석 및 처리에 따른 여러 변수들에 대한 설계가 이루어져 실제적인 테스트 및 평가 분석이 필요하다고 본다

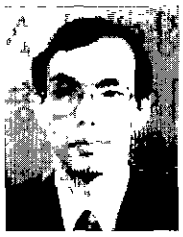
참 고 문 헌

[1] A. Agrawal, A. EL Abadi and R. Jeffers, "Using Delayed Commitment in Locking Protocols for Real-Time Databases." ACM SIGMOD, Vol.21, No.1, pp.104-113, 1992.

[2] A.Alan, B.Pritsker, "Introduction to Simulation and SLAM II." A Halsted Press Book, John Wilcy & Sons, Third Edition, 1986.

[3] Bernstein. P A., V Hadzilacos, and N. Goodman, "Concurrent Control and Recovery in Database

Systems," Addison-Wesley. 1987.
[4] M.H. Eich, "Foreword Main Memory Databases : Current and Future Research Issues," IEEE TKDE, Vol.4, No.6, pp.507-508, 1992.
[5] Xi Li & Margaret H. Eich, "Post-crash Log Processing for Fuzzy Checkpointing Main Memory Databases," IEEE Data Eng., 1993.
[6] J. Gray, & Reuter. A., Transaction Processing : Concepts and techniques, Morgan Kaufmann, San Maeteo. 1993.
[7] Le Gruenwald & M. H. Eich, "Simulation of Main Memory Database Recovery," SIMULATION, January 1993.
[8] T. Haerder & A. Reuter, "Principles of Transaction-Oriented Database Recovery," ACM Computing Surveys 15, 4, December 1983.
[9] Tobin J. Lehman & Michael J. Carey, "A Recovery Algorithm for High-performance Memory-Resident Database System," Proc. ACM SIGMOD, 1987
[10] C. Mohan & Don Haderle, et al, "ARIES : A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging," ACM TODS, Vol.17, No.1, March 1992.
[11] Kenneth Salem & Hector Garcia-Molina, "Checkpointing Memory-Resident Databases," IEEE Data Eng., 1989
[12] Kenneth Salem & Hector Garcia-Molina, "Main Memory Database Systems : An Overview," IEEE TKDE, Vol.4, No.6, 1992.
[13] E. I, Choi, H.C, Lim, "Recovery Technique Based on Fuzzy Checkpoint in a Client/Server DataBase System," IEEE Compsac'96, pp.542-547, 1996.
[14] Mukesh Singhal, "Issues and Approaches to Design of Real-Time Database Systems" ACM SIGMOD RECORD, Vol.17, No.1, 1988.
[15] Sang H.Son, "Real-Time Database Systems : Issues and Approaches" ACM SIGMOD RECORD, Vol.17, No 1, 1988.



박 용 문

e-mail : ympark@etri.re.kr
1983년 한남대학교 계산통계학과 졸업(학사)
1985년 중앙대학교 대학원 전자계산학과(이학석사)
2000년 한남대학교 대학원 컴퓨터공학과(박사과정)
1985년~2000년 한국전자통신연구원 선임연구원
관심분야 : 이동컴퓨팅, 실시간 데이타베이스

이 찬 섭

e-mail : cslee@dblab.hannam.ac.kr
1990년 한남대학교 컴퓨터공학과 졸업(학사)
2000년 한남대학교 대학원 컴퓨터공학과(공학석사)
2000년 한남대학교 대학원 컴퓨터공학과(박사과정)
관심분야 : Data mining, Web DB, XML

최 의 인

e-mail : eichoi@dblab.hannam.ac.kr
1982년 숭전대학교 계산통계학과 졸업(학사)
1984년 홍익대학교 전자계산학과 졸업(이학석사)
1995년 홍익대학교 전자계산학과 졸업(이학박사)
1985년~1988년 공군 교육사 전산실장
1992년~1996년 명지전문대학 전자계산과 조교수
1996년~현재 한남대학교 컴퓨터공학과 조교수
관심분야 : 실시간데이타베이스, 주기억데이타베이스, 클라이언트/서버 데이타베이스