데이타베이스 공유 시스템에서 B-트리 인덱스를 위한 캐쉬 일관성 제어

온 경 오[†] · 조 행 래[††]

요 약

데이타베이스 공유 시스템(Database Sharing System : DSS)은 고성능의 트랜잭션 처리를 위해 제안된 구조이다. DSS에서 고속의 통신망으로 연결된 노드들은 별도의 메모리와 운영체제를 가지며, 데이타베이스를 저장하고 있는 디스크는 모든 노드에 의해 공유된다. 빈번한 디스크 액세스를 피하기 위해 각 노드는 최근에 액세스한 데이타 페이지와 인덱스 페이지들을 자신의 메모리 버퍼에 캐싱한다. 일반적으로 B-트리 인덱스 페이지들은 데이타 페이지에 비해 빈번하게 캐싱되고, Fetch, Fetch Next, 삽입, 그리고 삭제와 같은 복잡한 연산을 수행하므로, 높은 동시성을 지원하는 효율적인 캐쉬 일관성 기법이 필요하다. 본 논문에서는 DSS에서 B-트리 인덱스 페이지의 식별자와 리프 페이지의 PageLSN을 사용한 캐쉬 일관성 기법을 제안한다. 제안한 기법은 노드간의 통신과 인덱스 재순회를 줄임으로써 단위 시간당 트랜잭션 처리량을 향상시킬 수 있다.

A Cache Consistency Control for B-Tree Indices in a Database Sharing System

kyungoh Ohn[†] · Haengrae Cho[††]

ABSTRACT

A database sharing system (DSS) refers to a system for high performance transaction processing. In the DSS, the processing nodes are coupled via a high speed network and share a common database at the disk level. Each node has a local memory and a separate copy of operating system. To reduce the number of disk accesses, the node caches data pages and index pages in its memory buffer. In general, B-tree index pages are accessed more often and thus cached at more processing nodes, than their corresponding data pages. There are also complicated operations in the B-tree such as Fetch, Fetch Next, Insertion and Deletion. Therefore, an efficient cache consistency scheme supporting high level concurrency is required. In this paper, we propose cache consistency schemes using identifiers of index pages and page_LSN of leaf page. The proposed schemes can improve the system throughput by reducing the required message traffic between nodes and index re-traversal.

키워드 : 병렬 DBMS(Parallel DBMS), 데이타베이스 공유(Database Sharing), B-트리(B-Tree), 캐쉬 일관성(Cache Consistency), 성능 평가(Performance Evaluation)

1. 서 론

데이타베이스 공유 시스템(Database Sharing System : DSS)은 고성능의 온라인 트랜잭션 처리가 필요한 응용분야를 효율적으로 지원하기 위하여 제안된 시스템으로 디스크 계층에서 전체 데이타베이스를 공유하고, 별도의 메모리와 운영체제를 가진다[3]. DSS를 구성하는 각 노드들은 물리적으로 인접한 위치에 존재하며, 고속의 통신 시스템을 이용하여 메시지 패싱 방식으로 통신한다. DSS의 경우 모든 노드가 데이타베이스를 공유하므로 임의의 노드가 고장이 나더라도 다른 노드들은 기존 작업을 계속 수행할 수 있다는 장점이 있다. 뿐만 아니라, 각 노드들은 상호 독립적으로 동작하기 때문에 분산 처리 작업이 단순해지며 노드들간의 부하 분산이 용이하고, 새로운 노드의 추가로 인한 확장성이 증가한다는 장점을 가진다.

DSS에서 각 노드는 자신의 버퍼에 최근에 액세스한 페이지들을 캐싱한다. 노드가 항상 최신의 데이타를 사용할 수 있기 위해서는 캐싱된 데이타의 일관성이 유지되어야 한다[2, 8, 10]. 그러나 DSS에서 기존에 제안된 대부분의 캐쉬 일관성 기법들은 데이타 페이지와 인덱스 페이지의 구분을 두지 않았다. 인덱스 페이지는 데이타 페이지에 비해 빈번히 액세스되고, 그 결과 보다 많은 노드에 캐싱될 가능성이 있다. 뿐만 아니라, B-트리 인덱스와 같이 인덱스가 계층적으로 구성될 경우 루트 페이지나 중간 페이지들은

† 준 회 원 : 영남대학교 대학원 컴퓨터공학과
†† 정 회 원 : 영남대학교 전자정보공학부 교수
논문접수 : 2001년 6월 28일, 심사완료 : 2001년 9월 10일

리프 페이지보다 빈번히 액세스된다. 그 결과 B-트리를 구성하는 모든 인덱스 페이지에 대해 무효화나 액세스 시 검사와 같은 일반적인 캐쉬 일관성 기법을 적용할 경우 빈번한 메시지 전송으로 인한 성능 저하가 발생할 수 있다[4, 12].

본 논문에서는 DSS에서 B-트리 인덱스를 위한 캐쉬 일관성 기법을 제안한다. 제안한 기법은 단일 시스템에서 B-트리 인덱스를 관리할 수 있는 대표적인 기법인 ARIES/IM[7]을 DSS 환경으로 확장하였다. ARIES/IM은 B-트리를 순회하는 동안 액세스할 페이지의 물리적인 일관성을 유지하기 위하여 래치(latch)의 개념을 사용한다. 분산 환경에서 래치를 구현하기 위하여 전역 로크 관리자(Global Lock Manager : GLM)를 이용한 페이지 로크(P 로크)를 지원하는 것이 일반적이다[8]. 각 노드는 캐싱된 페이지를 액세스하기 전에 공유 혹은 갱신 모드의 P 로크를 GLM에게 요청하고, GLM은 동일한 페이지에 대해 여러 노드에서 동시에 갱신 모드의 P 로크가 허용되지 않도록 P 로크 요청을 처리하여 그 결과를 다시 노드에게 전송한다. 이 과정에서 노드와 GLM간에 많은 수의 메시지 전송이 발생할 수 있으며, P 로크 요청 결과에 따라 B-트리를 재순회할 경우도 발생할 수 있다. 본 논문에서는 메시지 패싱 오버헤드와 인덱스 재순회 오버헤드를 최소화할 수 있는 캐쉬 일관성 기법을 제안하며, 제안한 기법의 성능을 분석하기 위하여 모의 실험 모형을 구축하여 다양한 데이타베이스 부하 환경과 시스템 구성에서 실험하도록 한다.

본 논문의 구성은 다음과 같다. 2절에서는 기존에 제안된 B-트리 인덱스를 위한 캐쉬 일관성 기법에 대해 살펴보고, 3절에서는 본 논문에서 제안한 캐쉬 일관성 기법을 설명한다. 4절에서는 성능 평가 모형을 설명하고, 5절에서 성능 평가 결과를 분석한다. 끝으로, 6절에서 결론 및 앞으로의 연구방향을 제시한다.

2. 관련 연구

B-트리 인덱스를 위한 기존 캐쉬 일관성 제어 기법으로는 ARIES/IM-SD[9]와 RIC[4], 그리고 복합형 캐싱(hybrid caching) 기법[12] 등을 들 수 있다. ARIES/IM-SD는 DSS 환경에서 제안되었으며, RIC와 복합형 캐싱 기법은 의뢰자-서버 환경에서 제안되었다. 복합형 캐싱 기법은 의뢰자 노드에서 B-트리의 리프 페이지들만 캐싱하며, 중간 단계의 페이지들은 자체적으로 구성한다. 이 기법은 서버 노드가 모든 갱신 연산 및 캐쉬 미스에 대한 디스크 I/O를 처리하므로 서버 노드에 부하가 집중될 수 있고, 그 결과 DSS에는 부적합하다. 따라서 본 절에서는 ARIES/IM-SD와 RIC를 중심으로 설명하도록 한다.

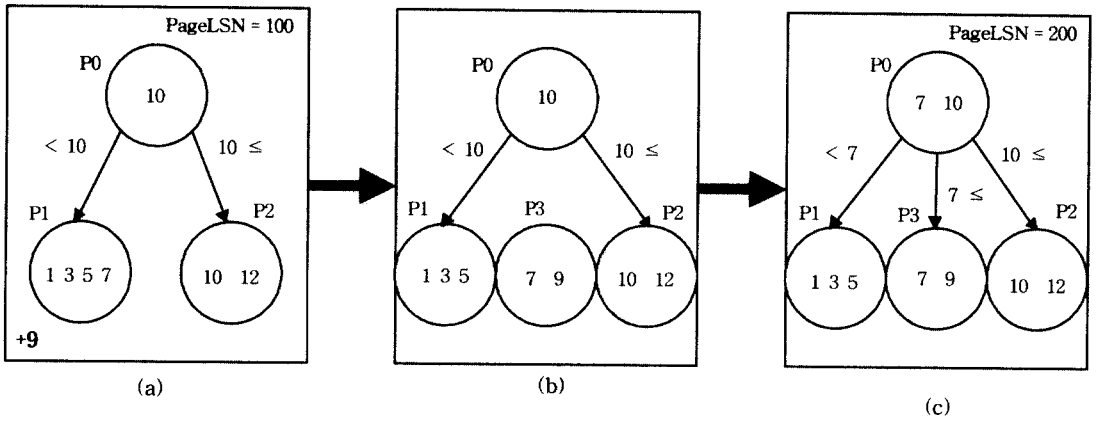
ARIES/IM-SD에서 인덱스 페이지들의 캐쉬 일관성은 두 가지 방식으로 관리된다. 중간 단계 인덱스 페이지에 대해서는 무효화 기반의 일관성 제어 기법을 사용한다. 즉, 페이지가 갱신될 때 그 페이지를 캐싱하고 있는 다른 노드들의 페이지를 무효화시킴으로써 캐싱된 페이지가 지역 노드에서 유효하다면 항상 정확한 페이지임을 보장한다. 그리고 리프 페이지에 대해서는 액세스 시 검사 기반의 일관성 기법을 사용하여 액세스하기 전에 페이지 식별자와 PageLSN[1]을 GLM에 전송하여 최신 버전의 페이지인가를 검사한다. 리프 페이지에 대한 일관성 검사 요청은 ARIES/IM의 다음 키 로킹 개념에 따라 데이타 레코드에 대한 로크 요청 메시지에 포함된다.

ARIES/IM은 로크-커플링 방식으로 인덱스 페이지에 래치를 획득하면서 루트 페이지부터 리프 페이지까지 B-트리를 순회한다[7]. 그러나, DSS의 경우 인덱스 페이지들이 여러 노드에서 동시에 액세스될 수 있고, 그 결과 액세스한 부모 페이지가 다른 노드에서 갱신되어 무효화될 수 있다. 따라서, ARIES/IM-SD에서는 트리 순회 과정에서 부모 페이지가 다른 노드에 의해 무효화되는 것을 감시하기 위해 자식 페이지에 대한 지역 래치를 획득한 후 부모 페이지의 무효화 여부를 검사한다. 부모 페이지가 무효화되었을 경우 최신 버전을 디스크나 다른 노드로부터 액세스한 후 필요한 단계부터 트리를 재순회한다.

RIC는 인덱스 연산을 "위치 단계"와 "실행 단계"로 나누어 처리한다[4]. 위치 단계는 연산이 실행될 위치를 찾는 트리 순회 단계이고, 실행 단계는 요청된 연산을 실행하는 단계이다. 위치 단계에서 RIC는 캐싱된 페이지에 대해 최신 버전인지를 검사하지 않는 낙관적 캐쉬 일관성 기법을 사용하여 B-트리를 순회한다. 즉, 이전 버전 페이지를 액세스하는 것을 허용하되 실행 단계에서 데이타에 대한 로크를 서버에 요청하는 동기화 시점에서 액세스한 인덱스 페이지가 최신의 것인지를 검사한다. 동기화 시점에서 의뢰자 노드는 로크 요청 정보와 함께 일관성 제어 정보들을 서버에게 전송한다. 일관성 제어 정보는 해당 인덱스를 포함하는 인덱스 세그먼트의 식별자와 세그먼트의 버전 번호로 구성된다. 서버는 세그먼트 버전 번호를 이용하여 세그먼트가 최신 버전인지를 판단하고, 이전 버전일 경우 서버는 인덱스 세그먼트에 포함된 모든 페이지들의 "LSN 벡터"를 로크 승인 메시지에 포함해서 전송한다. 의뢰자는 LSN 벡터를 이용하여 변경된 페이지들을 무효화하고, 만약 순회한 페이지들이 무효화되었을 경우 다시 인덱스를 순회한다.

ARIES/IM-SD는 중간 단계 인덱스 페이지의 경우 지역 버퍼에 무효화되지 않은 페이지가 있다면 정확한 페이지이므로 다른 노드와의 통신 없이 캐싱된 인덱스를 사용하여 B-트리 순회가 가능하다는 장점이 있다. 그러나, 캐쉬 무효

1) 데이타베이스 페이지에 저장되는 PageLSN 필드의 값은 그 페이지를 최종적으로 갱신한 연산에 대한 로그의 LSN(Log Sequence Number)으로 정의된다[6]. 따라서 한 페이지의 PageLSN 값은 단조 증가한다.



(그림 1) 삽입 연산에 의한 B-트리 구조 변경의 예

화를 위한 통신 오버헤드가 존재하며, 부모 페이지가 무효화될 경우 B-트리를 재순회해야 한다는 단점이 있다. 이에 대해 RIC는 B-트리 순회 단계에서의 통신은 최소화할 수 있지만, 인덱스가 자주 갱신될 경우 LSN 벡터 전송으로 인한 통신 오버헤드가 증가하며 실제 데이타와 관계없는 인덱스의 변화에 의해 여러 번 인덱스를 순회해야 하는 오버헤드가 발생한다. 이러한 관점에서 본 논문에서는 RIC와 같이 낙관적인 일관성 검사를 수행하지만, 재순회와 무효화를 줄일 수 있는 효율적인 캐쉬 일관성 기법을 제안하도록 한다.

3. 인덱스 일관성 검사

ARIES/IM-SD와 RIC는 연산과 관련 없는 구조 변경에 의한 무효화와 그에 따른 인덱스 재순회의 오버헤드가 존재한다. 예를 들어, (그림 1)과 같이 P0와 P1의 소유자 노드[2]에서 리프 페이지 P1의 분할로 인해 P0의 PageLSN이 증가될 수 있고, 그 결과 P2에 대한 캐쉬 일관성 검사를 요청한 다른 노드들은 변경된 P0에 의해 인덱스를 재순회를 하여야 한다. 뿐만 아니라, P0의 최신 버전을 소유자 노드로부터 전송 받아야하므로 메시지 전송에 따른 지연도 발생한다. 그러나, 트리 (a)에서 트리 순회 결과 P2를 액세스한 연산들은 변경된 트리 (c)에서도 역시 P2를 액세스하므로, P2의 관점에서는 인덱스 순회 및 P0의 전송이 필요 없다.

본 논문의 기본 개념은 B-트리의 순회 결과가 틀릴 가능성이 있을 때에만 변경된 페이지를 전송 받은 후 트리를 재순회한다는 것이다. 이때 트리 순회 결과의 오류 가능성은 리프 페이지의 최신 버전 여부에 따라 판단할 수 있다. 그리고 리프 페이지가 이전 버전일 경우, 리프 페이지의 조상 페이지 중에서 변경된 페이지들만 전송 받음으로써 통신 오버헤드를 최소화하도록 한다. 이를 위해 GLM은 갱신된 모든 페이지에 대해 (페이지 식별자, PageLSN, 소유자 노드) 쌍으로 구성된 "캐쉬 테이블"을 유지한다고 가정한다.

본 절에서는 B-트리 순회 과정과 리프 노드에서의 캐쉬 일관성 검사 과정에 대해 먼저 설명하도록 한다. 이때 삭제 연산에 의해 인덱스 페이지의 underflow가 발생하더라도 인덱스 페이지들간의 병합 연산이 실행되지 않는다고 가정한다. 이러한 가정은 삭제 연산의 복잡성을 줄일 수 있으며 인덱스 페이지가 분할될 확률을 줄일 수 있다는 장점으로 인해 B-트리 구현 시 많이 채택되고 있는 가정이다[5]. 인덱스 페이지들간의 병합을 허용할 경우, 본 논문에서 제안한 캐쉬 일관성 기법의 추가 고려 사항은 3.3절에서 설명하도록 한다.

3.1 B-트리 순회

B-트리 순회는 ARIES/IM과 같이 자식 페이지에 공유 래치를 획득하고 나서 부모 페이지의 공유 래치를 해제하는 로크-커플링 방식으로 수행한다[3]. (그림 2)는 로크-커플링 방식을 사용하여 B-트리 인덱스를 순회하는 과정을 나타낸다. 위치 단계에서는 중간 단계 인덱스 페이지의 최신 여부를 검사하지 않으므로 순회 시 지역 버퍼에 캐싱되어 있다면 GLM과의 통신을 하지 않는다. 단계 3에서 페이지 Pc에 대해 캐쉬 미스가 발생했을 경우, 처리 노드는 Pc 요청 및 Pc의 조상 페이지들에 대한 캐쉬 검사 메시지를 GLM에게 전송한다. 조상 페이지들에 대한 캐쉬 검사 메시지를 전송하는 이유는 다른 노드에서 Pc가 분할될 수 있기 때문이다. 즉, 분할된 최신 Pc만 전송받고 갱신된 Pc의 조상 페이지는 전송받지 않을 경우 Pc를 통한 순회 결과는 부정확할 수 있다. 따라서, 변경된 Pc의 조상 페이지도 같이 전송받은 후 조상 페이지가 갱신되었을 경우 처음부터 트리를 재순회하여야 한다.

2) 페이지의 "소유자 노드"란 페이지에 대한 갱신 모드의 P 로크를 보유하고 있는 노드를 의미한다[8].

3) 이때 래치란 한 노드에서 여러 트랜잭션이 동시에 실행될 때 각 트랜잭션이 액세스하는 페이지의 물리적인 일관성을 지원하기 위한 "지역 래치"를 의미한다. 노드들간의 캐쉬 일관성을 위해 사용되는 "분산 래치"는 "P 로크"란 용어로 표현하도록 한다.

1. 지역 인덱스의 루트 페이지에 대한 공유 래치 획득
2. 자식 페이지 Pc가 지역 버퍼에 있는지 검사
   2.1 Pc가 지역 버퍼에 있고 리프 페이지이면 Goto 5
   2.2 Pc가 지역 버퍼에 있고 중간 단계 페이지이면 래치 요청 후 Goto 4
3. Pc가 버퍼에 캐싱되지 않을 경우, Pc와 Pc의 조상 페이지들에 대해 (페이지 식별자, PageLSN) 리스트 **L**을 전송. L에서 Pc의 PageLSN 필드 값은 0으로 설정.
   3.1 GLM은 **L**을 캐쉬 테이블의 정보와 비교하여 캐쉬 검사 결과 메시지 **C**를 작성.
   - (Pi, PageLSNi) ∈ **L**에 대해 PageLSNi < 캐쉬 테이블[Pi].PageLSN일 경우, (Pi, 캐쉬 테이블[Pi].소유자 노드)를 **C**에 포함. 캐쉬 테이블[Pi].소유자 노드에게 Pi의 전송을 요청.
   - 캐쉬 테이블[Pi]가 존재하지 않을 경우, (Pi, 0)를 **C**에 포함.
   3.2 페이지를 요청한 노드는 GLM으로부터 **C**를 전송받은 후, 소유자 노드나 디스크에서 **C**에 포함된 페이지 액세스
   3.3 만약 Pc의 조상 페이지가 갱신되었다면 Goto 1
4. Pc의 래치를 획득하면 부모 페이지의 래치 해제. Pc가 리프 페이지가 아닌 경우, Goto 2
5. 리프 페이지에 대해 판독 연산일 경우 공유 래치, 기록 연산일 경우 갱신 래치를 요청

(그림 2) 로크-커플링을 사용한 B-트리 인덱스 순회

3.2 B-트리 순회 결과의 검증

B-트리 인덱스를 순회한 후 실행 단계에서 GLM에게 데이타에 대한 로크를 요청한다. 이때 순회과정에서 액세스한 인덱스 페이지들에 대한 일관성 정보도 함께 전송한다. 제안한 캐쉬 일관성 검사 기법들은 조상 인덱스 페이지들의 식별자 리스트와 PageLSN을 모두 전송하는 비관적 알고리즘과 리프 페이지의 PageLSN만을 전송하는 낙관적 알고리즘으로 구분된다.

3.2.1 비관적 알고리즘

비관적 알고리즘의 일관성 검사는 (그림 3)과 같이 수행된다. 비관적 알고리즘의 기본 개념은 B-트리 재순회가 필요할 경우 이를 효율적으로 처리하도록 하는 것이다. 이를 위하여 처리 노드는 단계 1에 나타나는 것과 같이 B-트리 순회 과정에서 액세스한 페이지들에 대한 (페이지 식별자, PageLSN) 리스트 **L**을 GLM에게 전송한다. B-트리 재순회가 필요할 경우 GLM은 **L**을 이용하여 재순회에 필요한 최신 페이지 리스트를 처리 노드에게 전송하게 되고, 처리 노드는 최신 페이지들을 액세스한 후 B-트리를 재순회할 수 있다. 단계 2.2에서 데이타 로크를 해제하는 이유는 이전 버전의 B-트리를 순회하였으므로 잘못된 데이타 레코드를 액세스했을 수 있기 때문이다.

[예 1] 노드 N1이 (그림 1)의 (a) 트리를 캐싱하고 있고, 노드 N2가 (c) 트리를 캐싱하고 있다고 가정하자. 그리고 (a) 트리에서 P1.PageLSN은 80, (c) 트리에서 P1.PageLSN은 180이라고 가정하며, P2.PageLSN은 (a)와 (c)에서 모두 50이라고 가정하자. 만약 N1이 "key = 10"인 레코드를 액세스할 경우, 데이타 로크 요청 메시지에 {(P0, 100), (P2, 50)}을 추가하여 GLM에게 전송하고, P2의 PageLSN이 최신 버전이므로 GLM은 로크 승인 메시지를 N1에게 전송한다. N1이 "key = 7"을 액세스할 경우, 데이타 로크 요청 메시지에 {(P0, 100), (P1, 80)}을 추가하여 GLM에게 전송하고, P1이 최신 버전이 아니므로 GLM은 {(P0, N2), (P1, N2)}를 포함하는 캐쉬 검사 결과 메시지를 N1에게 전송한다. 그리고, N2에게 P0와 P1을 N1으로 전송하도록 한다. N1은 P0와 P1의 최신 버전을 전송받은 후 트리를 재순회하고, P0의 내용에 따라 P3를 액세스한다. □

1. 처리 노드 N은 B-트리 순회 과정에서 액세스 한 인덱스 페이지들에 대해 (페이지 식별자, PageLSN) 리스트 **L**을 포함한 데이타 로크 요청 메시지를 GLM으로 전송
2. 데이타 로크가 승인된 후, GLM은 리프 페이지 P_{leaf}가 최신 버전인지 검사 :
   $PageLSN_{leaf}$ ≥ 캐쉬 테이블[P_{leaf}].PageLSN일 경우 최신 버전.
   2.1 최신 버전이면 로크 승인 메시지를 N에게 전송
   2.2 이전 버전이면 데이타 로크를 해제한 후, (Pi, PageLSNi) ∈ **L**에 대해 캐쉬 검사 결과 메시지 **C**를 아래와 같이 작성한 후 N에게 전송
   - PageLSNi < 캐쉬 테이블[Pi].PageLSN일 경우, (Pi, 캐쉬 테이블[Pi].소유자 노드)를 **C**에 포함. 캐쉬 테이블[Pi].소유자 노드에게 N으로 Pi의 전송을 요청.
   - 캐쉬 테이블[Pi]가 존재하지 않을 경우, (Pi, 0)를 **C**에 포함.
3. N은 메시지를 분석
   3.1 로크 승인 메시지일 경우, 연산 실행
   3.2 캐쉬 검사 결과 메시지일 경우 **C**에 포함된 페이지에 대해 아래 과정 수행
   - **C**에 포함된 모든 페이지들에 대해 상위 단계에서부터 갱신 래치 획득
   - 소유자 노드가 0이 아닐 경우, 소유자 노드로부터 페이지 전송 받음
   - 소유자 노드가 0일 경우, 디스크에서 페이지 액세스
   - 각 페이지를 액세스한 후, 해당 페이지의 갱신 래치 해제
   - 모든 페이지를 액세스한 후, B-트리 재순회. Goto 1.

(그림 3) 비관적 알고리즘의 일관성 검사 과정

3.2.2 낙관적 알고리즘

낙관적 알고리즘의 일관성 검사는 (그림 4)와 같이 수행된다. 비관적 알고리즘과의 차이는 단계 1에서 리프 페이지의 식별자와 PageLSN만을 GLM에게 전송한다는 것이다. 따라서, 리프 페이지가 최신 페이지가 아닐 경우, 단계 3에서 나타나는 것과 같이 B-트리 재순회를 위하여 처리 노드와 GLM간에 추가적인 메시지 전송이 발생한다.

[예 2] 앞에서 설명한 **[예 1]**의 경우를 다시 고려하자. N1이 "key = 10"인 레코드를 액세스할 경우, N1은 데이타 로크 요청 메시지에 (P2, 50)만 포함하여 GLM에게 전송한다. 이때 P2는 최신 버전이므로 GLM은 로크 승인 메시지를 N1에게 전송한다. 그러나, N1이 "key = 7"을 액세스할 경우에는 캐쉬 일관성 검사를 위한 추가적인 메시지 전송

이 발생한다. 즉, N1은 (P1, 80)을 로크 요청 메시지에 포함하여 전송하고, GLM은 P1이 이전 버전임을 N1에게 통보한다. 이후 N1은 {(P0, 100), (P1, 80)}을 GLM에게 다시 전송한 후, P0와 P1의 최신 버전을 N2로부터 전송받아 트리를 재순회하여야 한다. □

1. 처리 노드 N은 B-트리 순회 과정에서 액세스 한 리프 페이지의 (페이지 식별자, PageLSN)을 포함한 데이타 로크 요청 메시지를 GLM으로 전송
2. 데이타 로크가 승인된 후, GLM은 리프 페이지의 PageLSN을 이용하여 리프 페이지가 최신 버전인지 검사
   2.1 최신 버전이면 로크 승인 메시지를 N에게 전송
   2.2 이전 버전이면 데이타 로크를 해제한 후, 트리 재 순환 메시지를 N에게 전송
3. N은 메시지를 분석
   3.1 로크 승인 메시지일 경우, 연산 실행
   3.2 트리 재 순환 메시지일 경우, 아래 과정 수행
   - N은 루트 페이지부터 리프 페이지까지 순회 경로에 포함된 페이지들의 (페이지 식별자, PageLSN) 리스트 **L**을 GLM에게 전송
   - GLM은 (Pi, PageLSNi) ∈ L에 대해 캐쉬 검사 결과 메시지 **C**를 작성하여 N에게 전송. **C**의 작성 방법은 (그림 3)의 단계 2.2와 동일함.
   - N은 **C**에 포함된 페이지들을 (그림 3)의 단계 3.2와 같이 액세스한 후, 트리 재순회. Goto 1

(그림 4) 낙관적 알고리즘의 일관성 검사 과정

3.2.3 비교

비관적 알고리즘과 낙관적 알고리즘은 상호 보완적이다. 비관적 알고리즘은 B-트리 순회 후 데이타에 대한 로크를 요청할 때 순회한 모든 인덱스 페이지들의 식별자와 PageLSN을 포함함으로써 로크 요청 메시지의 크기가 커진다. 이와는 달리 낙관적 알고리즘은 리프 페이지의 식별자와 PageLSN만 포함함으로써 로크 요청 메시지 크기를 줄일 수 있다. 그러나 리프 페이지가 최신 버전이 아닐 경우, 낙관적 알고리즘은 순회 경로에 있는 인덱스 페이지들의 식별자와 PageLSN을 GLM에게 다시 전송해야 하는 오버헤드가 필요하다. 비관적 알고리즘은 데이타 로크 요청시 전송된 페이지 식별자 리스트를 사용하여 각 페이지의 소유자 노드 정보를 처리 노드에 전송함으로써 캐쉬 일관성 검사를 위한 추가적인 메시지 전송이 발생하지 않는다.

GLM이 B-트리에 대한 페이지 맵을 유지할 경우에는 비관적 알고리즘과 낙관적 알고리즘을 하나로 통합하는 것이 가능하다. 기본 개념은 리프 페이지의 식별자와 PageLSN만을 GLM에게 전송하고, 리프 페이지가 이전 버전일 경우 GLM은 리프 페이지의 조상 페이지들에 대한 (PageLSN, 소유자 노드) 리스트를 처리 노드에게 다신 통보한다는 것이다. 처리 노드는 자신이 캐싱하고 있는 페이지들의 PageLSN과 GLM으로부터 전송받은 PageLSN을 각각 비교하여, 이전 페이지들에 대해서는 소유자 노드나 디스크로부터 최신 버전을 바로 액세스할 수 있다.

기존에 제안된 캐쉬 일관성 기법과 비교할 때, 본 논문에서 제안한 캐쉬 일관성 기법은 다음과 같은 장점이 있다. 첫째, 순회 단계에서 액세스 경로에 포함된 인덱스 페이지들이 모두 지역 버퍼에 캐싱되어 있을 경우, 실행 단계에서만 GLM으로 일관성 정보를 전송하므로 순회를 위한 GLM과의 통신을 줄일 수 있다. 둘째, 갱신 연산에 의해 인덱스 페이지에 변화가 발생하더라도 그 페이지를 캐싱하고 있는 다른 노드에서 무효화시키지 않으므로 무효화 오버헤드를 줄일 수 있다. 셋째, 리프 페이지가 최신 버전이라면 액세스된 중간 단계 페이지가 이전 버전이더라도 재순회를 하지 않으므로 인덱스 재순회 빈도를 줄일 수 있다.

3.3 인덱스 페이지의 병합에 따른 고려 사항

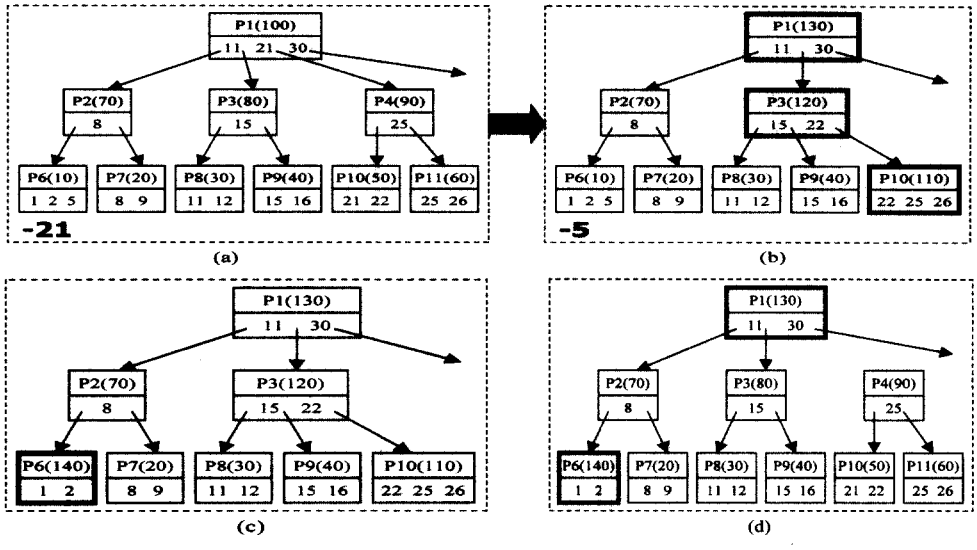
본 논문에서 제안한 인덱스 캐쉬 일관성 기법은 리프 페이지의 최신 여부에 따라 B-트리 순회 결과를 판단한다. 인덱스 페이지의 병합이 발생하지 않을 경우 B-트리의 중간 단계 페이지에 저장된 키 값은 삭제되지 않으므로, 중간 단계 페이지가 변경되더라도 기존 리프 페이지에 대한 포인터는 항상 유지된다.[4] 그러므로, 순회 결과로 액세스한 리프 페이지가 변경되지 않았을 경우에는 정확한 키 값 정보를 발견할 수 있다.

그러나, 인덱스 페이지의 병합이 발생할 경우에는 리프 페이지의 최신 버전 여부만으로는 순회 결과의 정확성을 판단할 수 없다. 예를 들어 (그림 5)와 **[예 3]**을 살펴보자.

[예 3] 노드 N1과 N2가 모두 (그림 5)의 (a) 트리를 캐싱하고 있는 상태에서, N2에서 21이 삭제되어 페이지 병합이 발생한 후 (b) 트리가 생성되었다고 가정하자. 이때 페이지병합으로 인해 P10, P3, P1의 PageLSN이 모두 증가하였다. 그리고 N2의 (b) 트리에서 다시 5가 삭제될 경우, (c) 트리가 생성되며 P6의 PageLSN이 증가하였다. 이때 (a) 트리를 캐싱하고 있는 N1이 "key = 2"인 레코드를 검색할 경우, 리프 페이지 P6가 이전 버전이므로 N2로부터 P6와 P6의 조상 페이지 중에서 갱신된 페이지 P1을 전송받으며, 그 결과 N1은 (d) 트리를 캐싱하게 된다. 이후 N1이 (d) 트리에서 "key = 22"를 검색할 경우, P9을 액세스한 후 22의 값을 갖는 데이타가 없다는 틀린 결과를 생성하지만 P9은 최신 버전이므로 GLM은 오류를 탐지할 수 없다. □

이러한 문제가 발생하는 이유는 이전 버전의 리프 페이지를 액세스할 경우, 리프 페이지의 조상 페이지에 대해서만 최신 버전으로 변경하기 때문이다. 즉, N1이 P6를 액세

4) 중간 단계 페이지가 분할될 경우에는 자식 페이지에 대한 포인터가 다른 페이지로 이동될 수 있으나, 이 경우에는 (그림 3)의 단계 3.2에 나타나듯이 갱신 래치를 이용하여 현재 변경된 인덱스 경로를 순회중인 트랜잭션이 존재하지 않을 경우에만 페이지를 갱신한다.



(그림 5) 삭제 연산에 의한 B-트리 구조 변경의 예

스한 후 P6와 P1만 최신 버전으로 대체하며 P3는 이전 버전을 계속 캐싱하고 있다. 이때 P1에는 P4에 대한 키 정보가 삭제되며 P3에는 P4의 정보가 포함되어 있지 않으므로 P1을 통한 P3로의 순회는 틀린 결과를 생성할 수 있다. 데이타 삽입으로 인한 페이지 분할 시에는 키 정보가 삭제되지 않으므로 이러한 문제가 발생하지 않는다.

(그림 5)의 문제는 ARIES/IM-SD와 같이 병합된 페이지들에 대해 다른 노드에게 무효화 메시지를 전송함으로써 해결할 수 있다. 즉, **[예 3]**에서 N2가 (b) 트리를 생성한 후, GLM에게 (P1, P3, P10)의 병합으로 인한 변경 사실을 통보하고, GLM은 (P1, P3, P10)을 포함하는 무효화 메시지를 다른 노드에게 전송한다. N1은 P1→P3→P10의 순서로 갱신 래치를 획득한 후, 각 페이지들을 무효화시킨다. 이후 N1이 P3를 액세스할 때 캐쉬 미스가 발생하여 최신 버전의 P3를 N2로부터 전송받으므로, **[예 3]**의 문제를 해결할 수 있다.

페이지 무효화 정책을 포함한 경우에도 본 논문에서 제안한 캐쉬 일관성 기법의 성능은 크게 저하되지 않을 것으로 예상된다. 그 이유는 페이지 underflow에 따른 페이지 병합 연산은 성능상의 이유로 실제 시스템에서 구현하지 않는 것이 일반적이기 때문이다[5]. 단, 삭제 연산의 결과 빈 인덱스 페이지가 생성될 경우에 한해 페이지 병합을 하는 경우는 있으나, 상대적으로 발생 빈도가 낮으므로 전체 시스템의 성능에 큰 영향을 미치지는 않는다.

3.4 인덱스에 대한 기타 연산들

본 논문에서는 B-트리 순회 연산에서 필요한 캐쉬 일관성 기법에 대해 주로 설명하였다. 그러나, B-트리에 대해서는 Fetch Next나 삽입, 삭제와 같은 인덱스 연산이 실행될 수 있으며, 이에 대해서도 캐쉬 일관성을 유지해야 한다. [9]에서 설명한 바와 같이 이러한 연산은 ARIES/IM과 같은 기존의 단일 시스템에서 사용되던 인덱스 관리 알고리즘에 대해 래치의 개념을 GLM을 이용한 P 로크로 대치함으로써 비교적 간단하게 구현할 수 있다.

Fetch Next 연산은 조건에 맞는 키를 가진 다음 레코드를 찾는 연산으로 조건에 따라 리프 페이지들간의 NextPage 링크나 PrevPage 링크를 이용하여 다음 리프 페이지를 액세스할 수 있다. 현재 캐싱된 리프 페이지를 이용하여 레코드를 발견한 다음, 레코드에 대한 로크 요청과 함께 리프 페이지에 대한 PageLSN을 전송함으로써 리프 페이지가 최신 버전임을 GLM으로부터 검사 받아야 한다. 그 이유는 해당 리프 페이지가 다른 노드에서 다시 갱신될 수 있기 때문이다. 만약 캐싱한 페이지가 이전 버전일 경우에는 최신 버전의 리프 페이지를 다른 노드나 디스크로부터 액세스한 후 다음 레코드를 다시 발견하여야 한다.

리프 페이지에 대한 삽입, 삭제 연산들은 순차적으로 실행되어야 한다. 이를 위한 처리 과정은 데이타 페이지에 대한 갱신 알고리즘[8]과 거의 동일하다. 즉, 처리 노드 N은 리프 페이지를 갱신하기 전에 리프 페이지에 대한 갱신 모드의 P 로크를 GLM에게 요청하며, GLM은 현재 리프 페이지의 소유자 노드에게 페이지 전송을 요청한다. 소유자 노드는 현재 실행 중인 리프 페이지의 갱신 연산을 완료한 후 갱신된 페이지를 N에게 전송한다. 이후 GLM은 N을 리프 페이지의 새로운 소유자 노드로 등록한다.

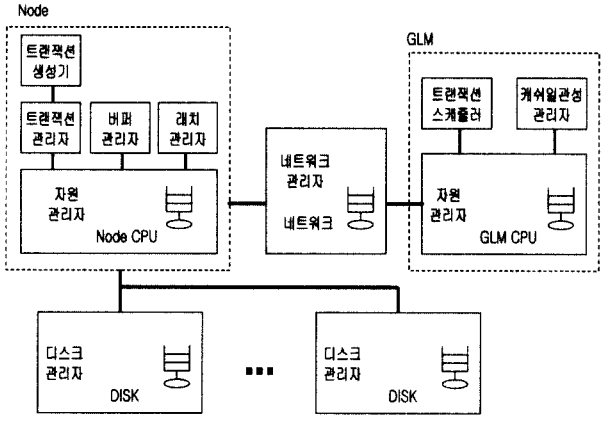
리프 페이지에 대한 삽입, 삭제 연산의 결과 인덱스 페이지가 분할되거나 병합되는 B-트리 구조 변경 연산(Structure Modification Operation : SMO)이 실행될 수 있다. 본 절에서는 페이지 분할에 대한 SMO 처리 과정을 중심으로 설명한다. 리프 페이지 P_{leaf}가 가득찬 경우 삽입 연산을 실

행할 수 없으므로 P_{leaf}를 분할하는 SMO를 실행하여야 한다. 여러 SMO들이 동시에 실행되는 것을 방지하기 위하여 트리 로크와 트리 래치를 먼저 획득한다. 트리 로크는 노드들간에 SMO 연산을 순서화하기 위해 사용되며, 트리 래치는 하나의 노드에서 하나의 SMO가 실행되도록 하기 위해서 사용된다. 트리 로크와 래치를 획득한 후, P_{leaf}의 SM_bit를 '1'로 설정한다. SM_bit는 현재 SMO가 적용 중인 페이지를 표시하기 위해 사용되며, 트리를 순회하는 다른 트랜잭션이 SM_bit가 '1'로 설정된 페이지를 액세스할 경우 그 트랜잭션은 SMO가 완료된 후 트리를 재순회하여야 한다[7]. 분할을 위한 새로운 페이지 P_{new}를 선택한 후, 역시 SM_bit를 '1'로 설정한다. 이후 P_{new}를 이용하여 P_{leaf}의 내용을 분할하고, P_{leaf}의 부모 페이지 P_{parent}에 P_{new}의 정보를 추가한다. 이때 GLM과의 통신을 통해 P_{parent}의 최신 버전을 액세스함으로써 P_{parent}의 소유자 노드로 우선 등록하여야 하며, 리프 페이지에 대한 삽입과 동일한 방식으로 삽입 연산을 실행한다. P_{leaf}의 분할 결과가 P_{leaf}의 조상 페이지에게 모두 전달된 후, 갱신된 페이지들의 SM_bit를 '0'으로 설정하고 트리 래치와 트리 로크를 해제한 후 SMO는 완료한다.

ARIES/IM-SD와는 달리 SMO에 다른 노드의 페이지를 무효화하는 과정이 생략되므로 SMO의 실행 시간이 상대적으로 단축될 수 있다는 점에 유의하여야 한다.

4. 성능 평가 모형

본 절에서는 제안한 캐쉬 일관성 기법의 성능을 평가하기 위한 모의 실험 모형을 설명한다. 본 논문에서 개발한 DSS의 성능 평가 모형은 (그림 6)에 나타나며, 미국의 MCC에서 개발한 CSIM 언어[11]를 이용하여 구현하였다.



(그림 6) DSS의 성능 평가 모형

DSS는 하나의 GLM과 네트워크를 통해 연결된 여러 개의 노드로 구성되고, 각 노드는 모든 디스크들을 공유한다. 각각의 노드는 LRU 버퍼를 관리하는 버퍼 관리자와 캐싱된 페이지에 대한 래치를 관리하는 래치 관리자를 갖는다. 트랜잭션 생성기는 일정 시간의 지연을 가지고 트랜잭션을 생성하는 역할을 담당한다.

GLM은 각 노드의 트랜잭션에서 요청한 로크 관리 및 캐쉬 일관성 기법을 수행한다. 이를 위해 GLM은 트랜잭션 스케줄러와 캐쉬 일관성 관리자를 갖는다. 트랜잭션 스케줄러는 레코드 단위의 2 단계 로킹 기법을 지원하며, 캐쉬 일관성 관리자는 평가할 캐쉬 일관성 기법들을 각각 구현한다. 본 논문에서는 GLM이 전체 시스템에 하나만 존재한다고 가정하며, 전체 시스템의 병목 현상을 방지하기 위하여 GLM은 다른 노드들에 비해 성능이 우수하다고 가정한다.

본 논문에서 사용한 입력 매개 변수는 <표 1>에서 요약한다. 각 매개 변수의 값 설정을 위해 시스템 구성 변수의 값들은 [1]에서 주로 참조하였고, 인덱스에 관련된 변수들은 [4, 12]에서 참조하였다.

〈표 1〉 입력 매개 변수

시스템 구성 변수		
LLMSpeed	노드 CPU의 속도	10 MIPS
GLMSpeed	GLM CPU의 속도	100 MIPS
NetBandwidth	네트워크의 데이타 전송 속도	10 Mbps
NDBMS	노드의 수	1, 3, 5, 10, 20
DISK_NUM	공유 디스크의 수	4
MinDiskTime	최소 디스크 액세스 시간	0.01 초
MaxDiskTime	최대 디스크 액세스 시간	0.03 초
BufSize	노드의 버퍼 크기	300 페이지
FixedMsgInst	메시지 처리를 위한 고정 명령 수	20000
PerByteMsgInst	메시지 길이당 추가되는 명령 수	10000 per page
ControlMsgSize	제어 메시지의 길이	256
PerIOInst	디스크 I/O를 위한 명령 수	5000
TRSize	트랜잭션당 평균 페이지 액세스 수	10
TRSZDV	트랜잭션 길이의 편차	0.1
P_{lookup}	판독 트랜잭션 비율(%)	20 ~ 100
P_{update}	갱신 트랜잭션 비율(%)	0 ~ 80
인덱스 관련 변수		
NdxDepth	인덱스 트리의 높이	3, 4, 5
AccessedLeaf	액세스되는 리프 페이지 수	1000
NdxSegmentSize	인덱스 세그먼트의 크기	1000, 1200, 1500
IPGSIZE	인덱스 페이지의 fan-out	128
NdxPageInst	인덱스 페이지 처리 시 명령 수	5000
P_{update_parent}	부모 페이지 갱신 비율(%)	10

노드 수는 1에서 20까지 다양하게 변경하며 실험을 수행하였고, 네트워크 관리자는 10Mbps의 대역폭을 갖는 FIFO 서버로 구현하였다. 네트워크를 통해 메시지를 전송하는 과정을 표현하기 위해 노드의 CPU 및 GLM의 CPU는 메시지마다 FixedMsgInst만큼의 고정된 명령 수와 한 페이지가

추가될 때마다 PerByteMsgInst 만큼의 추가적인 명령 수를 실행한다. 공유 디스크의 수는 4로 가정하였으며, 디스크 액세스 시간은 0.01초에서 0.03초까지의 일양 분포를 따른다. 디스크와 CPU는 FIFO 큐를 이용하여 I/O 요청 및 로크 요청 등을 들어온 순서대로 처리한다.

트랜잭션은 연산의 종류에 따라 판독 트랜잭션과 갱신 트랜잭션으로 분류된다. 판독 트랜잭션은 B-트리 순회를 실행하여 특정 레코드를 판독하며 판독하는 레코드는 무작위로 선택한다. 갱신 트랜잭션은 판독 트랜잭션과 같이 B-트리 순회 후 리프 페이지를 갱신한다. 각 트랜잭션이 액세스하는 리프 페이지 수는 TRSize ± TRSize × TRSZDV 사이의 일양 분포를 따른다. 갱신 트랜잭션의 경우 리프 페이지 변화에 의해 부모 페이지의 구조가 바뀔 확률은 P_{update_parent}에 의해 결정되고, 전체 갱신 트랜잭션의 10%가 구조 변경 연산이 필요하다고 가정한다.

인덱스 트리의 모양에 따른 성능 차이를 평가하기 위하여 인덱스 트리의 높이를 다양하게 변경하였다. 단, 인덱스 내부 노드의 fan-out을 고정할 경우 트리의 높이가 커질수록 리프 노드의 수가 증가하므로, 인덱스 높이에 따른 캐싱 효과를 분석하기가 곤란해질 수 있다. 이런 문제점을 해결하기 위해 인덱스 트리의 높이에 관계없이 실제로 액세스되는 리프 노드의 수(AccessedLeaf)는 1000으로 고정하였다. 그리고 노드의 CPU는 순회되는 각각의 인덱스 페이지에 대해 NdxPageInst만큼의 명령을 실행한다.

모의 실험에서 사용한 주요 성능 평가 지수는 평균 트랜잭션 응답 시간이다. 이 시간에는 연산 실행 시간 및 통신 시간, 그리고 디스크 입출력 시간과 트리 순회 시간 등이 포함된다. 그리고, 보조 성능 지수로 트랜잭션 실행을 위해 소요되는 통신 시간과 트리 순회 시간을 사용하도록 한다. 신뢰성 있는 모의 실험 결과를 얻기 위해 다른 seed를 이용하여 여러번 실험한 결과들을 산출하였다. 각각의 실험들은 완료된 트랜잭션 수가 1,000개가 될 때까지 수행하며, 초기 100개가 완료될 때까지의 결과들은 무시하였다.

5. 실험결과 분석

본 절에서는 캐쉬 일관성 기법들의 성능 평가 결과를 분석한다. 비교한 캐쉬 일관성 기법들은 기존의 ARIES/IM-SD(ARIES)와 RIC(RIC), 그리고 본 논문에서 제안한 비관적 캐쉬 일관성 기법(PSS)과 낙관적 기법(OPT)이다. 먼저 <표 1>의 입력 매개 변수를 이용하여 각 기법들에 대한 트랜잭션 처리 시간을 정성적으로 비교한 후, 갱신 트랜잭션의 비율과 노드의 수, 그리고 인덱스 트리의 높이 등을 변화시키면서 실제 실험을 수행한 결과를 설명하도록 한다.

5.1 정성적인 성능 비교

트랜잭션 처리를 위해 소요되는 시간(T)은 식 (1)과 같다. T_{COM}은 캐쉬 일관성 검사 및 페이지 전송에 소요되는 통신시간이며, T_{TRAV}는 인덱스를 순회하는데 소요되는 시간이다. 그리고 T_{IO}는 디스크 입출력 시간이고, T_{PROC}은 실제 연산의 처리 시간이다.

$$T = T_{COM} + T_{TRAV} + T_{IO} + T_{PROC} \quad (1)$$

T_{IO}와 T_{PROC}은 캐쉬 일관성 기법마다 거의 동일하며, T_{TRAV}는 T_{COM}에 비해 상대적으로 작은 부분을 차지하므로 본 절에서는 T_{COM}의 관점에서 각 기법들을 비교하도록 한다. 트랜잭션은 TRSize 수만큼 연산을 수행하고, 각 연산당 평균 통신 횟수(#COM)만큼의 단위 통신 시간(T_{UNIT})이 필요하므로 T_{COM}은 식 (2)로 나타낼 수 있다.

$$T_{COM} = \#COM \times T_{UNIT} \times TRSize \quad (2)$$

ARIES의 경우 캐쉬 미스가 발생한 인덱스 페이지에 대해 GLM과의 통신이 필요하며, 최종적으로 리프 페이지에 대한 로크 요청 메시지가 전송되므로 $\#COM_{ARIES}$는 식 (3)과 같이 표현된다. 이때, $P_{cache_miss}(i)$는 레벨 i의 인덱스 페이지에 대해 캐쉬 미스가 발생할 확률이다. RIC의 경우에는 인덱스 세그먼트가 최신 버전일 확률(P_{rct_seg})에 따라 여러 번의 통신이 필요하므로 $\#COM_{OPT}$는 식 (4)와 같이 표현된다. PSS와 OPT도 RIC와 유사하게 리프 페이지가 최신일 확률(P_{new_leaf})에 따라 여러 번의 통신이 필요하다. 단, OPT의 경우에는 리프 페이지가 이전 버전일 경우에는 GLM과의 통신이 한번 더 발생한다.

$$\#COM_{ARIES} = \sum_{i=1}^{NdxDepth-1} P_{cache\_miss}(i) + 1 \quad (3)$$

$$\#COM_{RIC} = P_{rct\_seg} \times \#COM_{ARIES} + \sum_{i=1}^{\infty} (1-P_{rct\_seg})^i \times (i+1) \times \#COM_{ARIES} \quad (4)$$

$$\#COM_{PSS} = P_{new\_leaf} \times \#COM_{ARIES} + \sum_{i=1}^{\infty} (1-P_{new\_leaf})^i \times (i+1) \times \#COM_{ARIES} \quad (5)$$

$$\#COM_{OPT} = P_{new\_leaf} \times \#COM_{ARIES} + \sum_{i=1}^{\infty} (1-P_{new\_leaf})^i \times (i+1) \times (\#COM_{ARIES}+1) \quad (6)$$

T_{UNIT}은 전송되는 메시지의 크기에 따라 결정된다. 먼저, 캐쉬 일관성 검사를 위한 GLM과의 통신에 소요되는 기본 시간(T_α)은 식 (7)과 같이 표현할 수 있다.

$$T_\alpha = \frac{FixedMsgInst}{LLMSpeed} + \frac{LEN_{ReqMsg}}{NetBandwidth} + \frac{FixedMsgInst}{GLMSpeed} + \frac{LEN_{RspMsg}}{NetBandwidth} \quad (7)$$

식 (7)에서 LEN_{ReqMsg}는 검사 요청 메시지의 길이를 나타

내며 LEN_{RspMsg}는 결과 메시지의 길이를 나타내는데, 실제 크기는 캐쉬 일관성 기법에 따라 상이하다. ARIES의 경우, LEN_{ReqMsg}는 리프 페이지 식별자의 길이와 리프 페이지의 PageLSN 길이를 더한 값이 되며, LEN_{RspMsg}는 해당 리프 페이지의 최신 버전에 대한 PageLSN의 길이가 된다. 단, 갱신 트랜잭션의 경우, 중간 단계 인덱스 페이지에 대한 무효화 메시지가 이전 버전을 캐싱하고 있는 다른 노드들에게 전송되므로 T_{UNIT}은 식 (8)과 같이 표현할 수 있다. 식 (8)에서 $P_{update_parent}(j)$는 레벨 j의 인덱스 페이지가 갱신될 확률을 의미하며 $\#DBMS_{cache}$는 이전 버전의 페이지를 캐싱하고 있는 노드의 수이다.

$$T_{UNIT}(ARIES) = P_{lookup} \times T_a + P_{update} \times \left(\sum_{i=1}^{NdxDepth-1} \prod_{j=i}^{NdxDepth-1} P_{update\_parent}(j) \times T_{invalidation} \right)$$

$$T_{invalidation} = \left(\frac{LEN_{PID}}{NetBandWidth} + \frac{FixedMsgInst}{GLMSpeed} \right) \times \#DBMS_{cache} \quad (8)$$

RIC는 캐쉬 일관성 검사를 위해 리프 페이지의 식별자와 인덱스 세그먼트 식별자를 전송하므로 LEN_{ReqMsg}는 ARIES와 동일하다. 인덱스 세그먼트가 최신일 경우 LEN_{RspMsg}는 검사 결과만 통보하면 되므로 ARIES와 거의 동일하지만, 최신이 아닐 경우 인덱스 세그먼트에 포함된 모든 페이지들의 "LSN 벡터"를 전송해야 하므로 LEN_{RspMsg}는 NdxSegmentSize × $LEN_{PageLSN}$ 만큼 커진다. 인덱스 세그먼트가 최신일 경우의 통신 시간을 T_{a1}으로 정의하고 그렇지 않을 경우의 통신 시간을 T_{a2}로 정의할 경우, T_{UNIT}(RIC)는 P_{rct_seg}에 따라 식 (9)와 같다.

$$T_{UNIT}(RIC) = P_{rct\_seg} \times T_{a1} + (1-P_{rct\_segt}) \times T_{a2} \quad (9)$$

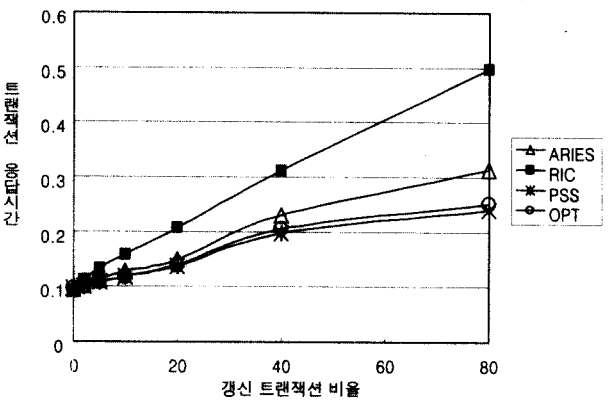
PSS에서 LEN_{ReqMsg}는 NdxDepth × (LEN_{PID} + $LEN_{PageLSN}$)이다. OPT의 경우, 첫 번째 메시지의 LEN_{ReqMsg}는 LEN_{PID} + $LEN_{PageLSN}$이며, 두 번째 메시지의 경우에는 PSS의 LEN_{ReqMsg}와 동일하다. 그리고, 두 기법 모두 LEN_{RspMsg}는 ARIES와 동일하다.

결론적으로 ARIES는 페이지 무효화에 의해 T_{unit}이 다른 기법에 비해 상대적으로 증가하며, 페이지 무효화 확률은 P_{update}와 $\#DBMS_{cache}$에 따라 결정된다. RIC는 P_{rct_seg}에 따라 #COM과 T_{unit}이 크게 변할 수 있다. PSS와 OPT에서는 P_{new_leaf}에 따라 #COM이 변하지만, $P_{new_leaf} \ll P_{rct_seg}$이므로 RIC에 비해 성능이 향상될 수 있다. 단, PSS에서는 T_{unit}이 NdxDepth에 비례하지만 OPT는 리프 페이지가 최신 버전일 경우 이에 대한 오버헤드를 줄일 수 있다.

이런 관점에서 다음 절에서는 P_{update}와 P_{rct_seg}, P_{new_leaf} 등에 따른 성능 차이를 분석하기 위해 갱신 트랜잭션의 비율을 변화시키면서 실험을 수행하였다. 그리고, 노드 수를 변화시킴으로써 앞의 확률 변수 외에 $\#DBMS_{cache}$의 영향을 분석하였다. 마지막으로 인덱스 트리의 높이를 변화시키면서 실험을 수행함으로써 NdxDepth의 영향을 분석하였다.

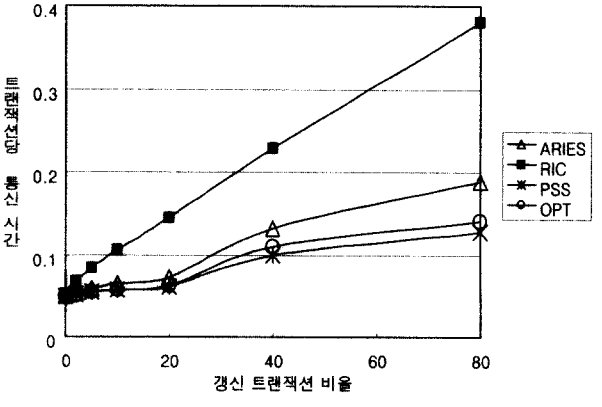
5.2 실험 1 : 갱신 트랜잭션 비율 변화

본 절에서는 갱신 트랜잭션의 비율에 따른 캐쉬 일관성 기법들간의 성능을 비교한다. DSS는 10개의 노드로 구성되고 인덱스 트리의 높이는 3으로 고정하였다. 캐쉬 일관성 기법의 평균 트랜잭션 응답시간이 (그림 7)에 나타난다.

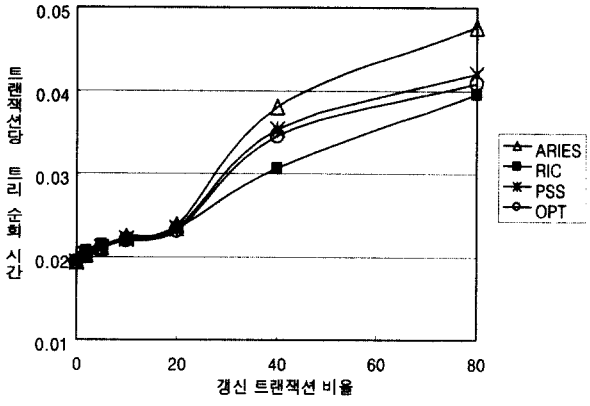


(그림 7) 갱신 트랜잭션 비율 변화에 따른 트랜잭션 응답 시간

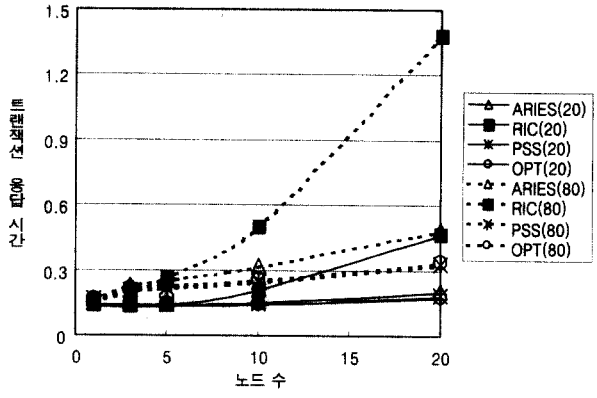
갱신 트랜잭션 비율이 증가할수록 캐싱 효과가 떨어지므로 모든 캐쉬 일관성 기법들의 성능이 저하되었다. 특히, RIC와 ARIES의 성능이 많이 저하되었는데, 이는 (그림 8)에 나타나듯이 캐쉬 일관성을 유지하기 위한 통신 오버헤드가 증가하기 때문이다. RIC의 경우 이전 버전을 캐싱하고 있는 노드에게 GLM이 LSN 벡터를 전송하며, LSN 벡터의 크기는 다른 캐쉬 일관성 기법에서 전송되는 메시지 크기보다 월등히 크므로 통신 시간이 많이 소요되었다. 한 가지 흥미로운 현상은 (그림 9)에 나타나듯이 RIC의 트리 순회 시간은 오히려 단축된다는 것이다. 그 이유는 RIC의



(그림 8) 트랜잭션당 평균 통신 시간



(그림 9) 트랜잭션당 평균 트리 순회 시간



(그림 10) 노드 수 변화에 따른 트랜잭션 응답 시간

경우 위치 단계에서는 트리 재순회를 하지 않지만, 다른 기법의 경우에는 부모 페이지의 무효화나 캐쉬 미스가 발생할 경우에 트리 재순회를 하기 때문이다. 그러나, 트리 순회에 소요되는 시간은 통신 시간에 비해 매우 작으므로 전체 응답 시간에 큰 영향을 미치지는 않았다.

ARIES의 경우에는 갱신 트랜잭션 비율이 높을 경우 페이지 무효화에 따른 통신 메시지 양의 증가로 인하여 제안한 기법에 비해 통신 시간이 높게 나타났다. 뿐만 아니라, 부모 페이지가 무효화될 때마다 트리 재순회를 하므로 트리 순회 시간도 상대적으로 높게 나타났다.

제안한 기법들은 GLM과의 통신량을 줄임으로써 ARIES에 비해 성능이 최대 20%, RIC에 비해서는 성능이 최대 100% 정도 향상되었다. 갱신 트랜잭션 비율이 작을 경우에는 제안한 기법들의 성능이 거의 비슷했으나, 갱신 트랜잭션 비율이 증가할수록 PSS의 성능이 OPT의 성능보다 우수하게 나타났으며 최대 5% 정도 성능이 향상되었다. 그 이유는 갱신 트랜잭션 비율이 증가할 경우 OPT에서 GLM과의 통신 횟수가 PSS보다 늘어나기 때문이다. PSS는 메시지 크기가 OPT보다 다소 크지만, (그림 8)에 나타나듯이 전체 통신 시간은 단위 메시지의 크기보다 메시지 전송 횟수에 더욱 큰 영향을 받았다.

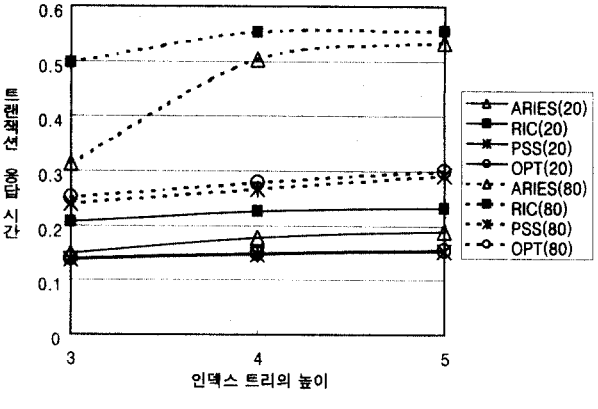
5.3 실험 2 : 노드 수 변화

본 절에서는 노드 수의 변화에 따른 캐쉬 일관성 기법들간의 성능을 비교한다. 이를 위하여 인덱스 트리의 높이는 3으로 고정하였고 갱신 트랜잭션 비율을 각각 20%와 80%로 고정한 상태에서 노드 수를 1에서 20까지 변경하면서 실험을 수행하였다. 실험 결과가 (그림 10)에 나타난다.

노드 수가 증가할수록 제안한 기법의 성능이 다른 기법들에 비해 성능이 우수한 것으로 나타났으며, 이러한 현상은 갱신 트랜잭션 비율이 80%일 때 더욱 명확하게 나타났다. 그 이유는 다음과 같다. 첫째, 노드 수가 증가할 경우 동시에 많은 수의 갱신 트랜잭션이 실행되고 그 결과 이전 버전을 캐싱할 확률이 커진다. 이전 버전을 캐싱할 경우 RIC는 대규모의 LSN 벡터를 전송하여야 하므로 통신 시간이 커지므로 트랜잭션 응답 시간이 늦어진다. 둘째, 노드 수가 클 경우 인덱스의 내부 페이지를 캐싱하고 있는 노드 수가 증가한다. 따라서 SMO 연산에 의해 내부 페이지가 변경될 경우 페이지 무효화를 위한 많은 수의 메시지가 GLM으로부터 다른 노드로 전송되어야 하고, 그 결과 ARIES의 통신 시간이 증가한다. ARIES와 같이 무효화를 이용한 캐쉬 일관성 기법에서 노드 수 증가에 의한 성능 저하 현상은 이미 [2]에서 지적된 바 있다. 이와는 달리 PSS와 OPT는 액세스 시 검사 개념에 기초한 캐쉬 일관성 기법을 채택하였으므로 노드 수에 관계없이 상대적으로 안정된 성능을 나타낼 수 있었다.

5.4 실험 3 : 인덱스 트리의 높이 변화

본 절에서는 인덱스 트리의 높이 변화에 따른 캐쉬 일관성 기법들간의 성능을 비교한다. 이를 위하여 노드 수는 10으로 고정하였고 갱신 트랜잭션 비율을 각각 20%와 80%로 고정한 상태에서 인덱스 트리의 높이를 3, 4, 5로 변경하면서 실험을 수행하였다. 실험 결과가 (그림 11)에 나타난다.



(그림 11) 인덱스 트리의 높이 변화에 따른 트랜잭션 응답 시간

인덱스 트리의 높이가 커질수록 ARIES의 성능이 급격히

나빠졌는데, 그 이유는 트리의 높이가 커질수록 내부 페이지 수가 많아지며 이에 대한 무효화 오버헤드가 증가하기 때문이다. 즉, 인덱스 트리의 상위 레벨에 위치한 페이지에 대한 SMO 연산의 경우 대부분의 노드로 무효화 메시지를 전송해야 하므로 통신 시간이 길어진다. RIC의 경우에는 인덱스 트리의 높이에 따른 성능 차이가 크지 않은 것으로 나타났다. 이는 본 논문에서 LSN 벡터의 크기가 인덱스 트리의 높이와 무관하다고 가정했기 때문이다. 즉, 인덱스 트리의 높이가 3인 경우와 5인 경우 모두 LSN 벡터의 크기를 동일하게 설정하였다. 만약 인덱스 트리의 높이가 커진 결과로 인덱스 트리에 보다 많은 페이지가 존재한다면, LSN 벡터의 크기도 커지며 RIC의 통신 시간도 증가할 것으로 예상된다.

PSS와 OPT의 경우에는 인덱스 트리의 높이에 따른 성능 차이가 크지 않았다. 그 이유는 두 기법의 경우 내부 페이지에 대한 이전 버전을 이용할 수 있으므로, ARIES와는 달리 내부 페이지가 변경되더라도 리프 페이지가 최신 버전일 경우에는 추가적인 메시지 전송이 발생하지 않기 때문이다. 그리고 예상과는 달리 인덱스 트리의 높이가 증가하더라도 PSS의 성능이 크게 떨어지지는 않았다. 즉, 인덱스 높이의 증가에 따라 로크 요청 메시지 길이가 증가하지만 이에 대한 통신 시간 지연은 크지 않았으며, OPT에서 발생하는 메시지 전송 횟수의 증가에 따른 통신 시간 지연 효과가 더욱 큰 것으로 나타났다.

6. 결 론

본 논문에서는 데이타베이스 공유 시스템(DSS)에서 B-트리 인덱스를 위한 캐쉬 일관성 기법을 제안하였다. 제안한 캐쉬 일관성 기법의 장점은 다음과 같다. 첫째, 리프 페이지까지 지역 버퍼에 캐싱되어 있을 경우 실행 단계에서만 GLM으로 일관성 정보를 전송하므로 순회 단계에서 GLM과의 통신을 줄일 수 있다. 둘째, 갱신 연산에 의해 인덱스 페이지에 변화가 발생하더라도 그 페이지를 캐싱하고 있는 다른 노드에서 무효화시키지 않으므로 무효화 오버헤드를 줄일 수 있다. 셋째, 리프 페이지가 최신 버전이라면 액세스된 중간 단계 페이지가 이전 버전이더라도 재순회를 하지 않으므로 인덱스 재순회를 줄일 수 있다.

제안한 기법의 성능을 분석하기 위하여 DSS를 위한 모의 실험 환경을 개발하였고, 다양한 데이타베이스 부하 환경과 시스템 구성 하에서 실험을 하였다. 실험 결과, 제안한 기법은 기존의 RIC와 ARIES/IM-SD 등의 기법에 비해 갱신 트랜잭션의 비율이 클 경우나 인덱스 트리의 높이가 클 경우, 그리고 DSS를 구성하는 노드 수가 증가할 경우 우수한 성능을 나타내었다. DSS의 주요 응용 분야가 대규모 데이타베이스에 대한 빠른 질의와 트랜잭션 처리라는 관점에서 인덱스 트리의 높이가 클 경우에 좋은 성능을 나타내는 것은 중요하다. 뿐만 아니라, 하드웨어의 가격 하락과 네트워크 속도의 향상 등을 가정할 때 DSS를 구성하는 노드 수는 점차 증가할 것으로 예상되며, 이러한 추세에 비추어 볼 때 제안한 캐쉬 일관성 기법의 특징은 매우 바람직하다고 할 수 있다.

참 고 문 헌

[1] M. Carey, M. Franklin and M. Zaharioudakis, "Adaptive, Fine-Grained Sharing in a Client-Server OODBMS : A Callback-Based Approach," ACM Trans. on Database Syst., Vol.22, No.4, pp.570-627, 1997.

[2] A. Dan and P. Yu, "Performance Analysis of Buffer Coherency Policies in a Multisystem DataSharing Environment," IEEE Trans. on Parallel and Distributed Syst., Vol. 4, No.3, pp.289-305, 1993.

[3] D. DeWitt and J. Gray, "Parallel Database Systems : The Future of High Performance Database Systems," Comm. ACM, Vol.35, No.6, pp.85-98, 1992.

[4] V. Gottemukkala, E. Omiecinski and U. Ramachandran, "Relaxed Index Consistency for a Client-Server Database," in : Proc. 12th Int. Conf. on Data Eng. pp.352-361, 1996.

[5] J. Gray and A. Reuter, 'Transaction Processing : Concepts and Techniques,' Morgan Kaufmann Publishers, 1993.

[6] C. Mohan et al., "ARIES : A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging," ACM Trans. on Database Syst., Vol.17, No.1, pp.94-162, 1992.

[7] C. Mohan and F. Levine, "ARIES/IM : An Efficient and High Concurrency Index Management Method Using Write-Ahead Logging," in : Proc. ACM SIGMOD, pp.371-380, 1992.

[8] C. Mohan and I. Narang, "Recovery and Coherency Control Protocols for Fast Intersystem Page Transfer and Fine-Granularity Locking in a Shared Disks Transaction Environment," in : Proc. 17th Int. Conf. VLDB, pp.193-207, 1991.

[9] C. Mohan and I. Narang, "Locking and Latching Techniques for Transaction Processing Systems Supporting the Shared Disks Architecture," IBM Research Report, 1995.

[10] E. Rahm, "Empirical Performance Evaluation of Concurrency and Coherency Control Protocols for Database Sharing Systems," ACM Trans. on Database Syst., Vol.18, No. 2, pp.333-377, 1993.

[11] H. Schwetman, 'CSIM User's Guide for use with CSIM Revision 16,' MCC, 1992.

[12] M. Zaharioudakis and M. Carey, "Highly Concurrent Cache Consistency for Indices in Client-Server Database Systems," in : Proc. ACM SIGMOD, pp.50-61, 1997.

온 경 오

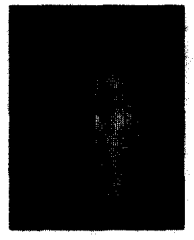
e-mail : ondal@cse.yu.ac.kr

1999년 영남대학교 컴퓨터공학과 학사

2001년 영남대학교 컴퓨터공학과 석사

2001년~현재 영남대학교 컴퓨터공학과 박사과정

관심분야 : 분산/병렬 데이타베이스, 트랜잭션 처리, DBMS 개발 등



조 행 래

e-mail : hrcho@yu.ac.kr

1988년 서울대학교 컴퓨터공학과 학사

1990년 한국과학기술원 전산학과 석사

1995년 한국과학기술원 전산학과 박사

1995년~현재 영남대학교 전자정보공학부 부교수

관심분야 : 분산/병렬 데이타베이스, 트랜잭션 처리, DBMS 개발 등