産業技術硏究(江原大學校 産業技術硏究所 論文集) 第 15輯 (1995)
Research Report, The Institute of Industrial Technology, Kangwon Nat'l Univ., Korea, Vol. 15, 1995

B^+ 트리를 위한 벌크 로드

김 상 욱 · 황 환 규*

On Bulk-Loading B^+-trees

Sang-Wook Kim · Whan-Kyu Whang*

ABSTRACT

In this paper, we propose a bulk-load algorithm for B'-trees, the most widely used index structures in database systems. The main characteristic of our algorithm is to simultaneously process all the keys to be placed on each B'-tree page when accessing the page. This avoids the overhead for accessing the same page multiple times, which results from applying the B'-tree insertion algorithm repeatedly. For performance evaluation, we analyze our algorithm in terms of the number of disk accesses. The results show that the number of disk accesses excluding those in the redistribution process is identical to the number of B'-tree pages. Considering that the redistribution process is an unavoidable preprocessing step for bulk-loading, our algorithm requires just one disk access per B'-tree page, and therefore turns out to be optimal. We also present performance tendancy according to the changes of parameter values via simulation.

I. 서 론

B'트리(B'-trees)[Bay72] [Com79] [Knu 73]는 데이타베이스내의 객체를 직접 액세스하기 위한 메카니즘으로서 가장 널리 사용되는 인덱스 구조이다. 본 논문에서는 B' 트리를 위한 벌크 로드(bulk-load) 기법에 관하여 논의하고자 한다. B' 트리를 위한 벌크 로드란 다수의 객체들의 키값(key value)들을 대상으로 B' 트리를 일괄처리방식으로 구성하는 것을 의미한다. B' 트리를 위한 연산 알고리즘들이 현재까지 많이 제안되어 왔으나[Bay72][Com79] [Knu73], 이들은 한 객체의 키값을 대상으로 하는 검색, 삽입, 삭제 알고리즘에 국한되어 왔으며, 이러한 벌크 로드에 관해서는 논의된 바 없다.

물론, 별도의 벌크 로드 알고리즘의 개

강원대학교 정보통신공학과 전임강사
* 강원대학교 정보통신공학과 조교수

발 노력 없이 하나의 객체의 키값을 B+ 트리에 삽입하는 기존의 삽입 알고리즘을 반복적으로 적용함으로써 전체 B+ 트리를 구성할 수도 있다. 그러나 이러한 경우, 객체의 키값들간의 인접성이 전혀 고려되지 않은 상태에서 키값들이 무작위 순으로 B+ 트리내에 삽입된다. 이 결과, 같은 페이지를 디스크로부터 여러번 액세스하게 되므로 B+ 트리 구성을 위한 오버헤드가 크다. 반면, 본 벌크 로드 알고리즘을 적용하는 경우, 각 페이지를 디스크로부터 한번 액세스할 때, 이 페이지에 저장될 여러 키값들을 함께 처리하므로 삽입 알고리즘을 반복적으로 적용하는 방식에서 발생하는 오버헤드를 해결할 수 있다. 특히, 데이타베이스 구축시에는 수백만에서 수천만에 이르는 방대한 양의 객체들을 대상으로 하므로 효율적인 벌크 로드 알고리즘의 개발은 반드시 필요하다.

본 논문에서는 B+ 트리의 벌크 로드를 위한 최적의 알고리즘을 제안한다. 먼저, 개념적인 설명을 위한 기본 형태의 알고리즘을 제안하고, 이 알고리즘의 실제 구현을 위한 성능 개선 방안을 제시한다. 또한, 벌크 로드의 수행시 발생되는 디스크 액세스 수를 분석함으로써 제안된 알고리즘의 최적성을 보이며, 시뮬레이션을 통하여 각 인자들의 값의 변화에 따른 성능상의 경향을 제시한다.

본 논문의 구성은 다음과 같다. 제2장에서는 B+ 트리의 특성에 관하여 간략히 소개한다. 제 3장에서는 벌크 로드를 위한 기본 알고리즘과 구현시의 디스크 액세스 최소화 방안에 관하여 자세히 논의 한다. 제 4장에서는 성능 분석 및 시뮬레이션 결과를 제시함으로써 본 알고리즘의 우수성을 보인다. 제 5장에서는 결론을 내리고, 앞으로의 연구 방향을 제시한다.

2. B+ 트리

B+ 트리[Bay72][Com79][Knu73]는 균형 탐색 트리(balanced search tree)로서 객체의 키값을 사용한 트리 순회(tree traverse)를 통하여 원하는 객체의 위치를 직접 찾을 수 있는 색인 구조의 하나이다. 본 장에서는 B+ 트리의 특성에 관하여 간략히 소개하고, 논의 전개를 위한 용어를 정의한다.

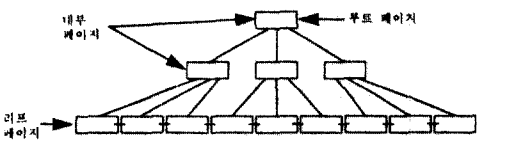
B+ 트리는 내부 노드(internal node)와 리프 노드(leaf node)로 구성된다. 데이타베이스 환경에서는 각 노드가 디스크 페이지내에 저장되므로 본 논문에서는 논의 전개상 각각을 내부 페이지(internal page) 및 리프 페이지(leaf page)라 부른다(그림 2.1(a) 참조). 리프 페이지는 <Object IDi, Keyi> (1<=i<=n)로 표현되는 리프 엔트리(leaf entry)들의 집합과 인접한 다음 리프 페이지를 가리키는 포인터 NxtPageID로 구성된다(그림 2.1(b) 참조). 여기서 Keyi는 한 객체의 키값을 나타내고[1], ObjectIDi는 이 객체가 저장된 위치를 가리키는 객체 식별자(object identifier)를 나타낸다. 리프 페이지내에서의 키값은 Key1<Key2<…<Keyn의 순이다.

내부 페이지는 <PageIDi, Keyi> (1<=i<=n) 의 쌍으로 구성되는 내부 엔트리들의 집합과 PageIDn+1로 구성된다(그림 2.1(c) 참조). 여기서 PageIDi는 하위 서브 트리의 루트 페이지를 가리키는 페이지 식별자(page identifier)이다. 내부 페이지내에서의 키값은 Key1<Key2<…<Keyn의 순이며, PageIDi가 가리키는 페이지내의

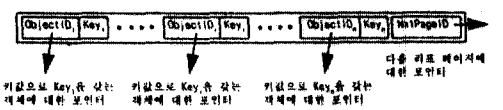
1 하나 이상의 객체들이 같은 키값을 가질 수도 있으나, 본 논문에서는 모든 객체가 유일한 키값을 갖는다는 가정하에 논의를 전개한다. 그러나 이 가정이 제안된 벌크 로드 알고리즘의 적용을 위한 제한 조건은 아니며, 단지 논의 전개를 단순화하기 위한 것이다.

임의의 엔트리의 키값 X는 1<i<=n인 경우에는 Keyi-1<X<=Keyi 이고, i=1인 경우에는 X<=Keyi이며, i=n+1인 경우에는 Keyi-1<X이다.

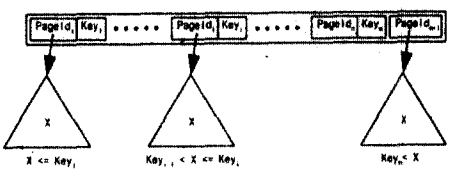
하나의 내부 페이지내에 최대로 들어갈 수 있는 엔트리의 수를 m이라 하면, 루트를 제외한 B+ 트리의 모든 내부 페이지는 항상 ⌈(m+1)/2⌉개 이상의 페이지 식별자들을 갖는다. 루트인 내부 페이지는 두개 이상의 페이지 식별자를 갖는다. 반면, 하나의 리프 페이지내에 들어갈 수 있는 엔트리의 수를 m'이라 하면, 루트를 제외한 B+ 트리의 모든 내부 페이지는 항상 ⌊(m'/2)⌋개 이상의 엔트리들을 갖는다. 루트인 리프 페이지는 한개 이상의 엔트리를 갖는다. 이러한 특성은 B+ 트리의 삽입, 삭제 알고리즘에 의하여 자동적으로 유지된다.



(a) Overall Structure.



(b) Leaf Page Structure.



(c) Internal Page Structure.

Fig. 2. 1. B+-Tree Structure.

3. 벌크 로드 알고리즘

본 장에서는 B+ 트리의 벌크 로드를 위한 알고리즘을 제안한다. 먼저, 보다 쉬운 개념의 이해를 위하여 기본 형태의 알고리즘을 기술하고, 이 알고리즘의 실제 구현을 위한 성능 개선 방안에 대하여 논의 한다.

3.1 기본 알고리즘

벌크 로드 알고리즘의 핵심 아이디어는 하나의 페이지를 디스크로부터 한번 액세스할 때, 이곳에 저장되어야 할 유사한 키값의 엔트리들을 함께 처리하는 것이다. 일반적으로 데이타베이스 구축시 객체들은 키값의 유사성을 고려하지 않은 상태에서 데이타 화일내에 무작위로 저장되어 있으므로 이 순서 그대로 벌크 로드 할 수는 없다. 따라서 키값이 서로 인접한 객체들과 대응되는 리프 엔트리들을 물리적으로도 인접하도록 사전 처리(preprocessing)로서 재배치 작업(relocation process)을 수행해야 한다.

재배치 작업 후 B+ 트리의 구성은 상향식(bottom-up)으로 진행된다. 먼저, 정렬된 리프 엔트리들을 참조하여 B+ 트리의 리프 단계를 구성하고, 이를 참조하여 그 상위의 내부 단계들을 구성한다. 이렇게 하위 단계를 참조하여 상위 단계를 구성하는 작업은 최종 단계가 하나의 페이지로 구성되는 루트 페이지로 될 때까지 반복된다. 이렇게 상향식으로 진행할 수 밖에 없는 이유는 B+ 트리의 각 단계가 하위 단계를 참조하여야만 구성이 가능하기 때문이다.

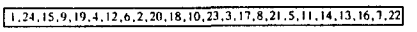
그림 3.1은 B+ 트리의 벌크 로드를 위한 알고리즘을 개략적으로 나타낸 것이다. 본 절에서는 알고리즘의 세부 작업에 관하여 자세히 논의하고, 그림 3.2를 이용하여 이 알고리즘의 수행의 예를 보인다.

1.재배치 작업을 수행한다

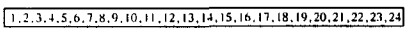
2.리프 단계를 위한 페이지화 작업을 수행한다.

3.내부 단계가 단 하나의 페이지만으로 구성될 때까지 해당 단계를 위한 내부 단계의 페이지화 작업을 수행한다.

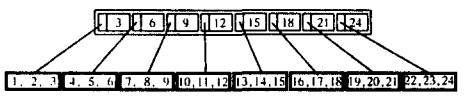
Fig. 3. 1. Basic Algorithm for Bulk-Loading.



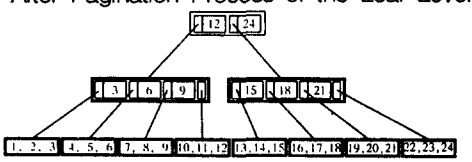
(a) Objects in a Raw Data File.



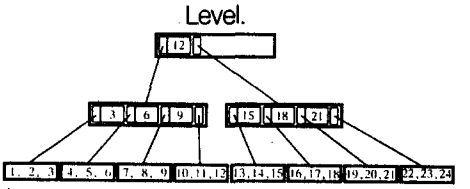
(b) After Redistribution Process.



(c) After Pagination Process of the Leaf Level.



(d) After Pagination Process of the First Internal Level.



(e) After Pagination of the Second Internal Level.

Fig. 3. 2. An Example of Bulk-Loading a B^+-Tree.

(1) 재배치 작업

재배치 작업(relocation process)을 위하여 외부 정렬(external sorting)[Hor93]을 이용한다. 즉, 객체를 참조하여 이와 대응되는 리프 엔트리를 생성하고, 이들에 대하여 버퍼 페이지 단위의 내부 정렬(internal sorting)을 먼저 수행한다. 이러한 내부 정렬이 완료되면, 그 결과에 대하여 K+1개의 버퍼 페이지를 이용하는 K 원 병합 작업(K-way merging)[Hor93]을 반복적으로 수행함으로써 전체 리프 엔트리들을 정렬시킨다. 그림 3.2(a)는 데이타 화일 내에 존재하는 객체들의 키값을 저장된 순서대로 나타낸 것이며, 그림 3.2(b)는 재배치 작업의 결과로서 정렬된 리프 엔트리들의 키값을 저장된 순서대로 나타낸 것이다.

(2) 리프 단계의 페이지화 작업

재배치 작업 후에는 정렬된 리프 엔트리들을 리프 페이지로 그룹화 하는 리프 단계의 페이지화 작업(pagination process)이 필요하다. 이 작업에서는 정렬된 리프 엔트리들을 차례로 액세스하며, 각 리프 페이지에 미리 지정된 수 만큼의 엔트리들을 저장한다. 이 수는 리프 페이지내에 저장 가능한 엔트리의 수를 m'이라 할 때, $\lfloor m'/2 \rfloor$와 m' 사이에서 응용의 특성에 따라 미리 지정할 수 있다[2]. 이 작업의 결과 B^+ 트리의 리프 단계가 완성된다.

---

2 벌크 로드 후 객체가 삽입될 가능성이 없는 응용에서는, 한 페이지내에 저장하는 엔트리의 수를 m'으로 지정함으로써 높은 저장 공간 이용율을 얻을 수 있다. 반면, 벌크 로드 후 추가로 삽입될 객체가 많은 응용에서는, 한 페이지내에 저장하는 엔트리의 수를 m'보다 작은 값으로 지정함으로써 각 페이지내에 여분의 저장 공간을 남겨둘 수 있다. 이 결과 새로운 객체의 키값을 삽입할 때, 이 여분의 저장 공간을 활용할 수 있으므로 분할(splitting) 과정으로 인한 오버헤드를 줄일 수 있다. 본 논문에서는 특별한 언급이 없는 한 저장 공간 이용율을 최대로 하는 m'을 가정하여 논의를 전개한다.

한편, 이와 병행하여 리프 단계 바로 상위의 내부 단계를 위한 정렬된 엔트리들을 생성한다. 즉, 하나의 리프 페이지가 완성될 때마다 이와 대응되는 상위의 내부 엔트리를 생성시킨다. 이 엔트리의 키값은 대응되는 리프 페이지내의 엔트리들 중 키값이 가장 큰 것이며, 페이지 식별자 값은 이 리프 페이지의 주소이다. 그림 3.2(c)는 그림3.2(b)를 이용하여 리프 단계의 페이지화 작업을 수행한 결과 완성된 B' 트리 리프 페이지들과 그 상위 단계의 정렬된 내부 엔트리들의 집합을 나타낸 것이다. 굵은 선으로 그려진 사각형은 페이지화가 완료된 페이지를 의미한다.

(3) 내부 단계의 페이지화 작업

이 작업에서는 직전의 리프 단계의 페이지화 작업의 결과로 얻은 정렬된 내부 엔트리들을 차례로 액세스하며, 리프 단계의 경우와 마찬가지로 각 내부 페이지내에 미리 지정된 수 만큼의 엔트리들을 저장한다. 단, 각 내부 페이지의 마지막 페이지 식별자는 정렬된 내부 엔트리들 중에서 그 내부 페이지내에 저장된 마지막 엔트리 직후에 나타나는 엔트리의 페이지 식별자이다. 또한, 이 엔트리의 키값은 이 내부 페이지와 그 직후에 나타날 내부 페이지가 갖는 엔트리들의 키값들을 분리시키는 역할을 하므로 이 내부 페이지 식별자와 함께 쌍을 이룸으로써 이 내부 페이지와 대응되는 상위의 내부 엔트리로서 생성된다. 내부 단계의 페이지화 작업의 결과, B+ 트리의 리프 단계 바로 상위의 내부 단계가 완성되며, 또한 그 상위의 정렬된 내부 엔트리들도 생성된다.

그림 3.2(d)는 그림 3.2(c)를 이용하여 내부 단계의 페이지화 작업을 수행한 결과 완성된 B' 트리의 리프 단계 및 바로 상위의 내부 단계와 또 그 상위의 정렬된 내부 엔트리들을 나타낸 것이다. 그림 3.2(c)에서 키값이 12인 내부 엔트리의 페이지 식별자값이 그림 3.2(d) 첫 내부 페이지의 마지막 페이지 식별자로서 사용되었으며, 이 키값 12와 첫 내부 페이지의 주소가 상위의 내부 단계 엔트리로 생성되었음을 볼 수 있다.

이러한 내부 단계의 페이지화 작업은 상위 단계로 계속 올라가면서 전체 단계를 하나의 페이지내에 유지할 수 있을 때까지 반복된다. 그림 3.2(e)는 그림 3.2(d)를 이용하여 내부 단계의 페이지화 작업을 수행한 결과를 나타낸 것이다. 이 단계에서는 모든 엔트리들을 하나의 페이지내에 유지시킬 수 있으므로 모든 작업이 종료된다. 그림 3.2(e)가 벌크 로드에 의하여 최종적으로 완성된 B' 트리이다

3.2 성능 개선 방안

본 절에서는 제 3.1절에서 제안한 알고리즘을 실제로 구현할 때, 보다 디스크 액세스 수를 줄일 수 있는 성능 개선 방안에 대하여 논의한다.

그림 3.3은 제안된 기본 알고리즘을 이용한 벌크 로드 과정을 나타낸 것이다. 왼쪽은 각 작업에서 생성되는 B' 트리의 단계를 리프 단계로부터 그린 것이며, 오른쪽은 각 작업과 화일간의 연관 관계를 나타낸 것이다. 특히, 검정색의 직사각형은 한 작업에서 생성되어 다음 작업에서만 사용될 뿐, 최종적으로는 제거해야 하는 임시 화일을 의미한다

각 화살표 옆에 나타난 READ와 WRITE는 화일을 읽거나 쓰는 경우에 발생하는 디스크 액세스를 나타낸다. 이들 중 데이타 화일을 위한 디스크 읽기와 B'

트리 페이지들을 위한 디스크 쓰기는 B' 트리 구성을 위하여 반드시 요구되는 최소한의 디스크 액세스이다. 반면, 나머지 READ, WRITE는 임시 화일과 관련된 디스크 액세스이므로 가능하다면 이들의 발생을 제거하는 것이 전체 성능 향상에 도움이 된다.

이와 같은 성능 개선을 위하여 본 논문에서는 각 단계의 페이지화 작업이 완료된 후에야 비로소 상위 단계의 페이지화 작업

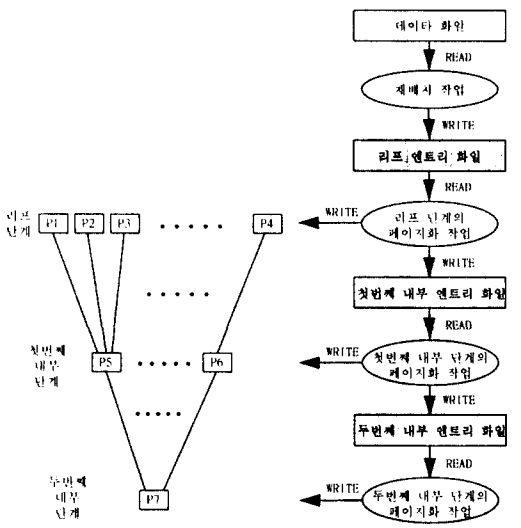


Fig. 3. 3. Processing Steps in Bulk-Loading.

을 시작하는 기본 알고리즘 대신, 하위단계의 페이지화 작업과 상위 단계의 페이지화 작업을 유기적으로 연관시켜 모든 단계의 페이지화 작업을 병행하는 방안을 사용한다. 그림 3.3의 왼쪽에 나타나는 B' 트리를 예로 살펴보자. 리프 단계내의 P1이 페이지화 되면, P1과 대응되는 엔트리를 생성하여 상위 단계인 첫번째 내부 단계에 넘겨준다. 유사한 방식으로 P2와 P3이 차례로 페이지화 되면 첫 내부 단계의 페이지 P5도 페이지화가 확정되므로 이때 P5와 대응되는 엔트리를 생성하여 그 상위 단계인 두번째 내부 단계에 넘겨준다. 이러한 작업은 리프 단계의 마지막 페이지인 P4의 페이지화가 확정될 때까지 반복된다.

이와 같이 하위 단계의 페이지화 작업과 상위 단계의 페이지화 작업이 병행될 수 있다. 이 결과 그림 3.3에서와 같은 임시 화일을 읽고 쓰는 과정이 불필요해진다. 따라서 재배치 작업의 완료 후에는 B' 트리에 속하는 페이지 수 만큼만의 디스크 쓰기가 발생한다.

4. 성능 분석

본 장에서는 제 3장에서 제안한 B' 트리 벌크 로드 알고리즘에 대한 성능을 분석한다. 먼저, 본 알고리즘을 적용하여 B' 트리를 구성할 때 발생되는 디스크 액세스 수를 추정한다. 또한 시뮬레이션을 통하여 각 인자들의 변화에 따르는 성능상의 경향을 제시한다.

4.1 디스크 액세스 수

먼저, 디스크 액세스 수의 분석을 위하여 필요한 인자를 정의하고, 본 알고리즘의 각 작업에서 발생되는 디스크 액세스 수를 추정한다.

N은 B' 트리 구성의 대상이 되는 객체의 수를 나타낸다. B는 B' 트리의 각 페이지의 블러킹 인수(blocking factor) [Wie83]로서, 페이지내에 몇개의 엔트리까지 저장이 가능한가를 나타낸다. 일반적으로 내부 페이지와 리프 페이지의 블러킹 인수는 다를 수 있으나, 여기서는 공식 유도를 단순화하기 의하여 두 수를 같도록 놓았다. 또한, K는 본 알고리즘의 재배치 작업에서 사용되는 외부 정렬내의 K 원

병합을 위하여 사용되는 K값을 나타낸다.

리프 단계는 $\lceil(N/B)\rceil$개의 페이지로 구성된다. 페이지 단위의 내부 정렬과 K 원 병합 작업을 이용하여 전체 리프 엔트리들을 정렬하는데 필요한 디스크 읽기 및 쓰기 단계 수는 $\lceil\log_K\lceil(N/B)\rceil\rceil$이다[Hor93]. 여기서 재배치 작업시 데이타 화일에서 객체들을 참조하기 위하여 발생하는 디스크 액세스는 B^+ 트리 구성을 위한 어떤 알고리즘에서도 발생하는 것이므로 이를 비용에서 제외한다. 또한, K 원 병합의 마지막 단계의 결과로 나타나는 정렬된 리프 엔트리들에 대해서는 쓰기 작업을 하지 않고, 그대로 페이지화 작업의 입력으로 사용할 수 있으므로 이 비용 역시 제외한다. 따라서 실질적인 디스크 읽기 및 쓰기 단계수는 각각 $\lceil\log_K\lceil(N/B)\rceil\rceil-1$ 이다. 각 단계에서 디스크 읽기와 쓰기의 두번의 디스크 액세스가 발생하므로 재배치 작업을 위한 총 디스크 액세스 수는 다음과 같다.

$$relocation\ cost = \lceil(N/B)\rceil \times (\lceil\log_K\lceil(N/B)\rceil\rceil - 1) \times 2 \quad (식\ 4.1)$$

리프 페이지와 중간 페이지의 페이지화 작업에서는 제 3.2절에서 설명한 성능 개선 방안을 적용하면, 완성된 B^+ 트리를 구성하는 페이지 수만큼의 디스크 쓰기만이 발생된다. N이 1에서 B까지는 B^+ 트리를 구성하는 페이지 수가 $\lceil(N/B)\rceil$, N이 B+1에서 B^2까지는 $\lceil(N/B)\rceil+\lceil(N/B^2)\rceil$, N이 B^2+1에서 B^3까지는 $\lceil(N/B)\rceil+\lceil(N/B^2)\rceil+\lceil(N/B^3)\rceil$이므로 이를 일반화하면 다음과 같다.

$$pagination\ cost = \sum_{x=1}^{\lceil\log_B N\rceil} \lceil(N/B^x)\rceil \quad (식\ 4.2)$$

따라서, 본 알고리즘을 적용하여 B^+ 트리를 구성하는 경우의 전체 디스크 액세스 수는 (식 4.1)과 (식 4.2)의 합이므로 다음과 같다.

$$total\ cost = \lceil(N/B)\rceil \times (\lceil\log_K\lceil(N/B)\rceil\rceil - 1) \times 2 + \sum_{x=1}^{\lceil\log_B N\rceil} \lceil(N/B^x)\rceil \quad (식\ 4.3)$$

본 알고리즘을 적용하여 B^+ 트리를 구성하는 경우, 재배치 작업을 제외하면 B^+ 트리에 속하는 페이지 수 만큼만의 디스크 쓰기가 발생한다. 재배치 작업이 벌크 로드를 위한 필수 불가결한 전처리 단계인 것을 감안하면, 제안된 알고리즘은 각 페이지에 대하여 단 한번의 디스크 액세스만을 요구하므로 벌크 로드를 위한 최적의 알고리즘이라 할 수 있다.

4.2 시뮬레이션

본 절에서는 제 4.1절에서 구한 세가지 공식을 이용한 시뮬레이션을 통하여 공식 내의 각 인자들의 변화에 따르는 제안된 알고리즘의 성능상의 경향에 대하여 논의한다. 먼저, 시뮬레이션에 사용된 인자들의 값에 대하여 설명하고, 시뮬레이션 결과를 제시한다.

블러킹 인수(blocking factor) B는 200으로 지정하였다. 이는 실제적으로 일반 상용 데이타베이스 관리 시스템에서 주로 사용하는 페이지의 크기가 4K 바이트라는 사실과 B^+ 트리 엔트리의 크기가 대략 20 바이트 내외라는 무리없는 가정을 기반으로 한 것이다. 또한 K 원 병합 작업을 위한 K 값은 10, 100, 200, 1,000 등의 네가지를 사용하였다. 이것은 시스템에서 제공하는 버퍼 페이지 수가 본 알고리즘에 미치는 성능상의 영향을 관찰하기 위한 것이다. 제 4.1절에서 언급한 바와 같이 K 원 병합을 수행할 때, K+1개의 버퍼 페이지를 사용하게 되는데, 대용량의 데이타를 관리하는 저장 시스템 혹은 데이타베이스 관리 시스템에서 실제로 천개 정도까지의 버퍼 페이지를 제공하는데는 큰 무리가 없다는

사실[Exo92]에 기반을 둔 것이다. 끝으로 대상이 되는 객체의 수 N은 1에서 10,000,000까지 25만개씩 증가시키며 B' 트리를 구성하도록 하였다. 이는 각 객체 수 증가에 따르는 본 알고리즘의 디스크 액세스 수 증가의 경향을 분석하기 위한 것이다.

그림 4.1은 K 값이 100인 경우와 200인 경우의 시뮬레이션 결과를 나타낸 것이다. 네개의 증가 곡선은 두가지 경우에 대하여 재배치 작업시와 페이지화 작업시에 각각 발생하는 디스크 액세스 수를 대상이 되는 객체 수의 변화에 따라 나타낸 것이다. 가로 축은 객체 수를 나타내고, 세로 축은 각 작업에서 발생하는 디스크 액세스 수를 나타낸다.

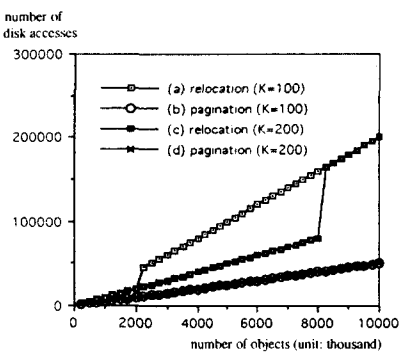


Fig. 4. 1. Disk Accesses in Redistribution and Pagination Processes.

먼저, 페이지화 작업시의 디스크 액세스 발생 수를 보면, K 값에 영향을 받지 않고 두 증가 곡선이 같은 특성을 보임을 알 수 있다. 이것은 (공식 4.2) 내의 인자에 K 값이 포함되지 않기 때문이다. 또한 이 증가 곡선은 선형 함수로 나타남을 볼 수 있다. 이것은 B(=200) >> 1이므로 (식 4.2)의 첫째 항 이후에 나타나는 항의 값들이 첫째 항이 갖는 값에 비하여 무시할 정도로 작기 때문이다. 따라서 첫째 항 이후의 항들을 무시하면, 페이지화 작업시 발생하는 디스크 액세스 수의 증가 함수는 객체 수에 대한 1/B(=1/200)의 기울기를 갖는 선형 함수임을 알 수 있다.

다음은 재배치 작업시의 디스크 액세스 발생 수를 살펴본다. 먼저, K 값이 200인 경우를 보면, 객체 수가 8,000,000인 지점까지는 디스크 액세스 수가 객체 수에 대한 2/B(=1/100)의 기울기를 갖는 선형 함수로 나타나다가 이 지점에서 디스크 액세스 수가 급격히 증가되며, 이후에는 4/B(=1/100)의 기울기를 갖게 됨을 볼 수 있다. 이것은 (식 4.1)내의 ($\lceil \log_K \lceil (N/B) \rceil \rceil$ -1) 값이 객체 수가 8000,000까지는 1이었다가 8,000,000이 넘으면서 2로 되므로 이 이후부터는 증가 곡선의 기울기가 두배로 되기 때문이다. 이것은 재배치 작업시의 외부 정렬내의 병합 단계 수가 이 지점이후 하나 더 늘어나는데서 비롯된 것이다. 한편, K 값이 100인 경우에는 ($\lceil \log_K \lceil (N/B) \rceil \rceil$-1) 값이 1에서 2로 되는 시점이 2,000,000이므로 이 지점에서 기울기가 변하게 된다. 객체 수가 2,000,000 이하인 구간과 8,000,000을 초과하는 구간에서는 K 값에 관계없이 같은 수의 병합 단계가 요구되므로 두 증가 곡선이 같은 특성을 보였다.

그림 4.2는 K 값이 10, 100, 1,000인 경우의 시뮬레이션 결과를 나타낸 것이다. 이것은 벌크 로드시에 발생하는 전체 디스크 액세스 수가 K 값의 변화에 대하여 어떻게 영향을 받는가를 보이기 위한 것이다. K 값이 커짐에 따라 각 증가 곡선의 평균 기울기가 크게 줄어듦을 볼 수 있다. 이는 전술한 대로 재배치 작업시의 병합 단계 수의 차이에서 비롯된 것이다.

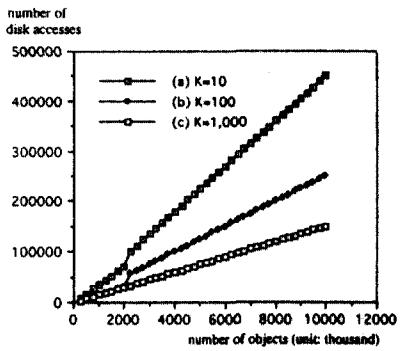


Fig. 4. 2. Disk Accesses in Bulk-Loading.

특히, K 값이 1,000인 경우에는 객체 수가 10,000,000인 시점까지 (식 4.1)내의 ($\lceil \log_K \lceil (N/B) \rceil \rceil - 1$) 값이 1을 유지하므로 3/1,000을 기울기로 갖는 선형 함수로 나타났다. 이 기울기는 재배치 작업을 위한 증가 함수의 기울기(=2/1,000)와 페이지화 작업을 위한 증가 함수의 기울기(1/1,000)를 더한 결과이다. 실제 대용량의 데이타베이스를 관리하는 시스템에서 1,000개 정도의 버퍼 페이지들을 제공하는 것은 일반적인 일이므로 본 시뮬레이션 결과를 통하여 10,000,000개까지의 객체들을 대상으로 벌크 로드를 수행할 때 발생하는 디스크 액세스 수를 예측할 수 있다.

5. 결 론

데이타베이스를 구축하는 경우나 데이타베이스내의 인덱스를 새로 구성하는 경우에는 방대한 양의 객체들을 대상으로 하므로 효율적인 인덱스 벌크 로드 기법이 요구된다. 본 논문에서는 데이타베이스 인덱스 구조로 가장 널리 사용되는 B' 트리를 위한 최적의 벌크 로드 알고리즘을 제안하였다.

먼저, 개념적인 설명을 위하여 기본 형태의 알고리즘을 제안하고, 구현시 디스크 액세스 수를 최소화할 수 있도록 전체 단계의 페이지화 작업을 병행할 수 있는 성능 개선 방안을 제시하였다. 또한 성능 분석을 위하여 본 알고리즘을 적용하여 B' 트리를 구성할 때 발생되는 디스크 액세스 수를 분석하였다. 분석 결과에 의하면, 벌크 로드를 위하여 반드시 요구되는 재배치 작업을 제외하면, B' 트리 구성에 사용되는 페이지 수 만큼만의 디스크 쓰기가 발생하는 것으로 나타났다. 따라서 각 페이지에 대하여 단 한번의 디스크 액세스만을 요구하므로 제안된 알고리즘은 벌크 로드를 위한 최적의 알고리즘이라 할 수 있다. 또한 시뮬레이션을 통하여 각 인자들의 변화에 따라 벌크 로드시 필요한 시간을 예측할 수 있도록 성능상의 경향을 분석하였다.

향후 연구 방향은 기존의 삽입 알고리즘을 반복적으로 적용하는 기존의 경우와 실험적인 비교를 통하여 제안된 알고리즘을 적용하는 경우 얼마만큼의 성능이 향상되는가를 규명하는 것이다.

후 기

본 논문은 인공지능연구 센터(CAIR) 95 연도 위탁연구과제 연구비 지원의 결과임.

참 고 문 헌

[Bay72] R. Bayer and C. McCreight, "Organization and Maintenance of Large Ordered Indexes," Acta Informatica 1, pp. 173-189, 1972.

[Com79] D. Comer, "The Ubiquitous B-Trees," ACM Computing Surveys, Vol. 11, No. 2, pp. 121-137, 1979.

[Exo92] Exodus Project Implementation Team, Using the Exodus Storage Manager, Exodus Project Document, University of Wisconsin, 1992.

[Hor93] E. Horowitz, S. Sahni, and S. Anderson-Freed, Fundamentals of Data Structures in C, Computer Science Press, 1993.

[Knu73] D. Knuth, The Art of Computer Programming, Vol. 3, Sorting and Searching, Addison-Wesley Publishing Co. Inc., 1973.

[Wie83] G. Wiederhold, Database Design, 2nd Ed., McGraw-Hill Book Company, 1983.