「정보시스템연구」 제 10권 제 1호
한국정보시스템 학회
2001년 06월, pp. 257-278

인덱스 그래프 : 동적 문서 데이터베이스를 위한 IR 인덱스 구조

박 병 권 동아대학교 경영정보과학부 bpark@daunet.donga.ac.kr

<목 차>

Ⅰ. 서 론

문서 데이타베이스를 위한 IR 인덱스란 정보검색을 위하여 문서 데이터베이스로부터 주어진 키워드가 발생한 문서들을 찾아 주는 인덱스로서(Frakes, 1992; Salton, 1988) 가장 널리 사용되고 있는 것은 역 인덱스(inverted index)이다(Faloutsos, 1995). 역 인덱스는 문서 데이타베이스에 저장된 문서로부터 추출된 각 키워드 별로 포스팅 리스트(posting list)를 유지하고 있다. 포스팅 리스트는 포스팅(posting)의 리스트를 말하며 포스팅은 그 키워드가 발생한 어떤 한 문서의 식별자(document identifier: docID)와 그 문서 내에서의 발생위치 정보를 담고 있다(Frakes, 1992; Salton, 1988).

IR 인덱스를 통한 정보검색은 주어진 키워드들의 포스팅 리스트를 검색함으로써 이루어진다. 예를 들어 "멀티미디어에 관한 문서를 찾아라"는 정보검색 질의는 '멀티미디어'라는 키워드의 포스팅 리스트를 검색함으로써 처리된다. 따라서, IR 인덱스는 정보검색을 위해 주어진 키워드의 포스팅 리스트를 찾을 수 있는 데이타 구조가 필요하다.

포스팅 리스트들이 문서 식별자 순으로 정렬되어 있으면 두 개 이상의 키워드로 이루어진 정보검색 질의를 효율적으로 처리할 수 있다. 왜냐하면 각 키워드에 대한 포스팅 리스트를 한 번만 순차적으로 읽어서 병합(merge)하면 되기 때문이다. 예를 들어, "멀티미디어와 데이타베이스에 관한 문서를 검색하라" 라는 질의는 '멀티미디어'와 '데이타베이스'라는 두 개의 키워드가 포함된 정보검색 질의이고 이를 처리하기 위해서는 두 키워드 '멀티미디어'와 '데이타베이스'의 포스팅 리스트를 각각 순차적으로 읽어서 병합하면 된다.

동적 문서 데이타베이스에서 문서의 동적 추가나 수정이 일어날 때 포스팅 리스트가 문서 식별자 순의 정렬을 유지하도록 IR 인덱스를 갱신하기 위해서는 문서의 추가나 수정에 따른 새로운 포스팅을 문서 식별자 순으로 포스팅 리스트에 삽입하여야 한다. 만약 새로운 문서의 식별자는 항상 증가되는 값을 부여받는다는 규칙이 있다면 새로운 문서의 포스팅은 항상 포스팅 리스트의 마지막에 추가되어야 한다. 그러나, 문서가 수정되면 문서의 내용만 새로운 내용으로 대체될 뿐 문서의 식별자 값은 변하지 않으므로 수정된 문서의 포스팅을 추가할 때에는 기존 문서 식별자의 위치에 따라 포스팅 리스트에 삽입되어야 한다. 따라서 동적 데이타베이스를 위한 IR 인덱스는 문서의 추가 또는 수정에 따라 동적으로 갱신되더라도 포스팅 리스트가 문서 식별자 순 정렬을 유지할 수 있는 데이터 구조를 가지고 있어야 문서의 추가 및 수정 성능을 높일 수 있다.

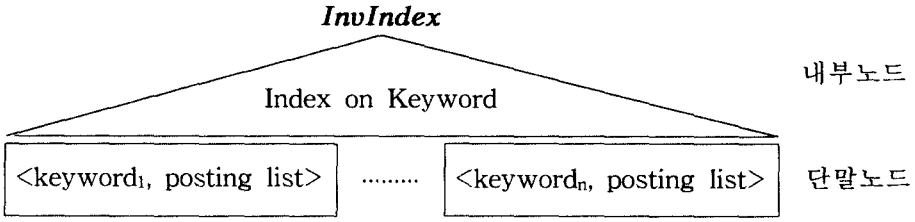
동적 문서 데이타베이스에서 문서의 삭제가 일어날 때 삭제된 문서의 모든 포스팅들이 포스팅 리스트에서 제거되도록 IR 인덱스를 갱신하여야 한다. 이 때 IR 인덱스가 다음과 같은 두 데이타 구조를 가지고 있으면 문서의 삭제 성능을 높일 수 있다. 첫째, 삭제된 문서의 포스팅이 포함된 포스팅 리스트들을 찾을 수 있는 데이타 구조가 필요하다. 이러한 포스팅 리스트는 삭제된 문서에 포함된 키워드에 대한 포스팅 리스트로서 그 개수는 삭제된 문서의 키워드 개수와 같다. 둘째, 특정 포스팅 리스트에서 삭제할 포스팅을 빨리 찾아 삭제할 수 있는 데이타 구조가 필요하다. 대부분의 문서에는 길이가 긴 포스팅 리스트를 가진 키워드가 다수 포함되어 있으므로 긴 포스팅 리스트에서의 포스팅 삭제 성능은 문서 삭제 성능의 중요한 요소이다.

본 논문에서는 동적 문서 데이타베이스를 위한 새로운 IR 인덱스 구조를 제안한다. 기존의 IR 인덱스 구조는 문서의 추가, 삭제, 수정이 없는 정적 문서 데이타베이스에서의 정보검색 만을 고려하였다. 본 논문에서는 분석 및 실험을 통하여 기존의 IR 인덱스 구조는 정보검색은 잘 지원하지만 문서의 추가, 삭제, 수정에 따른 인덱스의 동적 갱신은 잘 지원할 수 없음을 보이고 동적 문서 데이타베이스를 위한 새로운 IR 인덱스 구조로서 인덱스 그래프(Index Graph)를 제안한다. 인덱스 그래프는 여러 개의 인덱스들이 그래프처럼 연결되어 연동될 수 있는 구조로서 본 논문에서는 다음과 같은 세 가지 종류의 인덱스들이 연동되어 동적 문서 데이타베이스를 위한 IR 인덱스를 이룬다. 첫째, 키워드를 키로 하여 해당 포스팅 리스트를 찾아주는 인덱스이다. 이는 정보검색 질의에서 주어진 키워드의 포스팅 리스트를 빨리 찾기 위하여 필

요하다. 둘째, 각 포스팅 리스트마다 문서 식별자를 키로 하여 해당 포스팅을 찾아주는 인덱스이다. 이는 문서의 추가 또는 수정 시 포스팅 리스트에 포스팅을 문서 식별자 순 정렬을 유지하도록 삽입하거나, 문서 삭제 시 포스팅 리스트에서 삭제할 포스팅을 빨리 찾아 삭제하기 위하여 필요하다. 세째, 문서 식별자를 키로 하여 해당 포스팅 리스트들을 찾아주는 인덱스이다. 이는 문서 삭제 시에 삭제된 문서의 포스팅이 포함된 포스팅 리스트들을 빨리 찾기 위하여 필요하다. 분석 및 실험을 통하여 인덱스 그래프 구조의 IR 인덱스가 기존의 IR 인덱스 구조와 같은 정도의 정보검색 성능을 가지면서도 문서의 동적 추가, 삭제, 수정에 따른 인덱스 갱신 성능이 기존의 IR 인덱스 구조보다 우수함을 보인다.

II. 기존 IR 인덱스 구조의 문제점

IR 인덱스 구조로서 가장 간단한 것은 데이타베이스 관리 시스템을 이용하여 <키워드, 포스팅> 구조의 테이블을 데이터베이스에 저장하는 것이다. 그러나, 이 구조는 동일한 키워드가 발생 빈도 만큼 중복 저장되므로 질의 성능이 떨어지는 것으로 알려져 있다(DeFazio, 1995; Stone, 1983). IR 인덱스 구조로서 데이터베이스 테이블을 이용하지 않고 트리(tree)와 같은 데이타 구조를 이용하는 연구들이 많이 이루어져 왔다. 즉, <그림 2-1>과 같이 키워드에 대한 인덱스를 구축하고 단말 노드의 포인터 필드에 포스팅 리스트를 저장하는 구조이다. <그림 2-1>에서 내부 노드의 인덱스 엔트리 구조는 <키워드, 부트리 포인터>이고, 단말 노드의 인덱스 엔트리 구조는 <키워드, 포스팅 리스트>이다.



<그림 2-1> 기존 IR 인덱스 구조

Samuel DeFazio등은(DeFazio, 1995) 이러한 IR 인덱스 구조가 앞에서 언급한 데이터베이스 테이블을 이용한 구조보다 저장 공간 효율과 질의 성능 면에서 더 우수함을 실험을 통하여 보였다. 그리고 Doug Cutting(Cutting, 1990), Christos Faloutsos(Faloutsos, 1992), Anthony Tomasic(Tomasic, 1994) 등은 이러한 IR 인덱스 구조에서 문서의 동적 추가로 인하여 단말 노드의 포스팅 리스트 길이가 계속하여

늘어날 경우에 대한 저장 공간 할당 정책과 그 성능에 대하여 연구를 하였고, Ricardo A. Baeza-Yates등(Baeza, 1989)은 여러 가지 저장 공간 관리 정책에 대하여 삽입 비용과 저장 효율을 수학적으로 분석하였으며, Ribeiro-Neto등(Ribeiro-Neto, 1999)은 분산 컴퓨팅을 이용한 IR 인덱스 구축 방법을 제안하였다.

그러나, 기존의 IR 인덱스 구조에서는 문서의 동적 추가 또는 수정에 따라 새로운 포스팅을 포스팅 리스트에 삽입할 때 포스팅 리스트가 문서 식별자 순 정렬을 유지하도록 하는 방법에 대해서는 고려하지 않았다. 즉, < 그림 2-1>과 같은 인덱스 구조에서는 새로운 포스팅을 포스팅 리스트에 삽입할 때 문서 식별자 순으로 삽입하기 위해서는 포스팅 리스트 전체를 읽어서 삽입할 포스팅의 문서 식별자 순 위치를 찾은 다음에 포스팅을 삽입하는 방법을 사용하여야 한다. 그 결과 포스팅 리스트의 길이가 길 경우 문서의 동적 추가 또는 수정 성능이 크게 떨어질 수 있다.

또한, 기존의 IR 인덱스 구조에서는 문서의 동적 삭제에 따라 삭제된 문서의 포스팅을 포스팅 리스트에서 삭제하는 방법에 대해서도 고려하지 않았다. 즉, <그림 2-1>과 같은 인덱스 구조에서는 삭제된 문서의 포스팅이 포함된 포스팅 리스트를 찾을 수 있는 방법과 찾은 포스팅 리스트에서 삭제할 포스팅을 탐색할 수 있는 방법이 없다. 따라서 삭제된 문서에 포함된 모든 키워드의 포스팅 리스트들을 찾기 위하여 키워드 개수 만큼 키워드 인덱스를 탐색하여야 하고, 찾은 포스팅 리스트에서 삭제된 문서의 포스팅을 삭제하기 위하여 포스팅 리스트 전체를 읽어 보아야 한다. 그 결과 문서의 동적 삭제 성능이 크게 떨어질 수 있다.

Ⅲ. 동적 DB를 위한 IR 인덱스 구조

본 장에서는 <그림 2-1>과 같은 기존 IR 인덱스 구조를 개선하여 동적 문서 데이타베이스를 위한 새로운 IR 인덱스 구조를 제안한다. 먼저, 필요한 용어들을 정의하고 이를 이용하여 제안된 IR 인덱스 구조를 설명한다. 그리고, 제안된 IR 인덱스 구조는 정보검색을 잘 지원할 수 있을 뿐만 아니라 문서의 추가 또는 수정에 따라 새로운 포스팅을 포스팅 리스트에 추가할 때 문서 식별자 순의 정렬을 유지할 수 있도록 해 주며, 문서의 삭제에 따라 삭제된 문서의 포스팅을 포스팅 리스트에서 삭제할 때에도 이를 효율적으로 지원할 수 있음을 보인다.

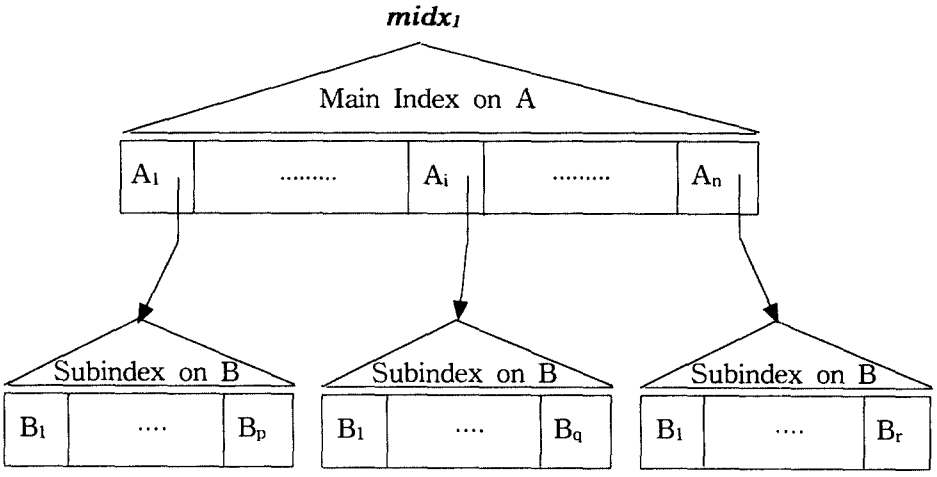
3.1 용어 정의

정의 1 : 두 인덱스 간의 **계단식 관계**(Cascading Relationship)란 두 개의 인덱스가 있을 때, 한 인덱스의 한 단말 레벨 인덱스 엔트리에서 다른 인덱스의 루트 노드(root node)를 가리키는 것으로 정의한다. 이 경우, 전자의 인덱스를 **주인덱스**(Main

Index)라 부르고 후자의 인덱스를 **부인덱스**(Subindex)라 부른다.

정의 2 : 두 인덱스 간의 **다중 계단식 관계**(Multiple-cascading Relationship)란 주인덱스의 한 단말 레벨 인덱스 엔트리에서 두 개 이상의 부인덱스를 가리키는 것으로 정의한다.

<그림 3-1>은 키 'A'에 대한 인덱스와 키 'B'에 대한 인덱스 사이에 계단식 관계가 있음을 예로 보여주고 있다. midx1은 키 'A'에 대한 인덱스로서 주인덱스가 되고 sidxi는 키 'B'에 대한 인덱스로서 부인덱스가 된다. midx1의 단말 레벨 인덱스 엔트리에서 sidxi를 가리키고 있다.

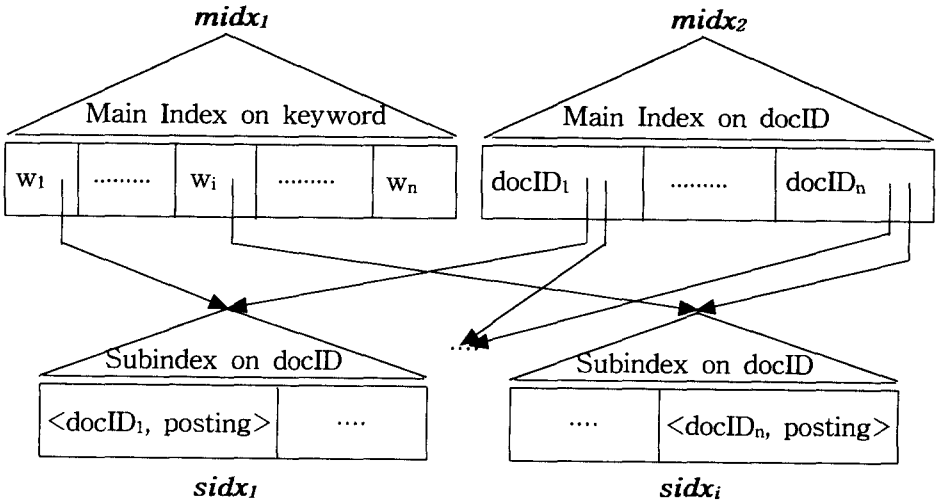


<그림 3-1> 인덱스간의 계단식 관계

계단식 관계는 주로 한 인덱스의 한 단말 레벨 인덱스 엔트리가 독립적인 인덱스로 확장(expand)되고 이를 전자의 인덱스가 가리킴으로써 많이 성립되며 여러 개의 인덱스들을 계층적으로 연결된 형태를 가지도록 하므로 다음과 같은 장점들을 가진다. 첫째, 서로 다른 인덱스 구조를 가진 인덱스들 사이에도 성립될 수 있다. 예를 들면, <그림 3-1>에서 midx1은 인덱스 구조로 트라이(trie) 구조를 가지고 sidxi는 인덱스 구조로 B+ 트리 구조를 가질 수 있다. 둘째, 여러 개의 인덱스가 연결되어 있으므로 여러 개의 키를 통한 접근이 가능하므로 전체적으로는 하나의 다차원 인덱스 구조처럼 생각할 수 있다. 그러나, 하나의 인덱스 구조를 가지는 기존의 다차원 인덱스 구조와는 달리 각 인덱스 별로 서로 다른 인덱스 구조를 가질 수 있고 새로운 인덱스를 추가하거나 일부 인덱스를 삭제하는 것이 용이하다.

3.2 인덱스 그래프

본 절에서는 동적 문서 데이타베이스를 위한 새로운 IR 인덱스 구조를 제안한다. 제안하는 IR 인덱스 구조는 <그림 3-2>와 같이 두 개의 주인덱스 midx1, midx2와 여러 개의 부인덱스들 sidxi (i=1,...,n)로 구성되어 있다. 여기서, midx1과 sidxi는 계단식 관계를 이루고, midx2와 sidxi는 다중 계단식 관계를 이루며, midx1과 midx2는 부인덱스를 공유한다. <그림 3-2>와 같은 구조는 각각의 인덱스를 노드(node)로 하고 인덱스 간의 관계를 모서리(edge)로 하면 전체가 하나의 그래프를 형성하므로 본 논문에서는 이를 **인덱스 그래프**라고 부른다.



<그림 3-2> 인덱스 그래프

인덱스 그래프를 구성하는 각 인덱스의 역할은 다음과 같다. sidxi는 문서 식별자를 키로 하여 키워드 wi의 포스팅 리스트에 구축된 인덱스로서 내부 노드의 인덱스 엔트리 구조는 <문서식별자, 서브트리 포인터>이고, 단말 노드의 인덱스 엔트리 구조는 <문서식별자, 발생위치 리스트>이다. 따라서 sidxi의 한 단말 레벨 인덱스 엔트리가 키워드 wi의 한 포스팅을 이룬다. 또한, sidxi를 통하여 특정 포스팅을 탐색할 수 있고 새로운 포스팅을 문서 식별자 순으로 삽입함으로써 키워드 wi의 포스팅 리스트가 문서 식별자 순의 정렬을 유지할 수 있도록 해 준다.

midx1은 키워드를 키로 하여 구축된 인덱스로서 주어진 키워드의 포스팅 리스트를 가진 부인덱스 sidxi를 찾아 주는 인덱스로서 내부 노드의 인덱스 엔트리 구조는 <키워드, 서브트리 포인터>이고, 단말 노드의 인덱스 엔트리 구조는 <키워드,

부인덱스 식별자>이다. midx2는 문서 식별자를 키로 하여 구축된 인덱스로서 문서 삭제 시 삭제된 문서의 포스팅이 포함된 포스팅 리스트들을 가진 부인덱스들의 집합을 찾아 주는 인덱스로서 내부 노드의 인덱스 엔트리 구조는 <문서식별자, 서브트리 포인터>이고, 단말 노드의 인덱스 엔트리 구조는 <문서식별자, 부인덱스 식별자 리스트>이다.

인덱스 그래프에서 정보검색은 다음과 같이 이루어 진다. 먼저, 주어진 키워드를 키 값으로 하여 midx1을 탐색함으로써 주어진 키워드의 포스팅 리스트를 가진 부인덱스 sidxi를 찾는다. sidxi의 단말 노드에는 주어진 키워드의 포스팅 리스트가 문서 식별자 순으로 저장되어 있으므로 sidxi의 단말 노드들을 순차적으로 읽어 반환하면 된다. 이 때, 부인덱스의 루트 노드에는 자신의 첫번째 단말 노드를 가리키는 포인터가 있고 부인덱스의 단말 노드들은 서로 연결되어 있으므로 인덱스를 탐색하지 않고 직접 포스팅 리스트에 접근한다.

인덱스 그래프에서 문서의 동적 추가 또는 수정은 다음과 같이 이루어 진다. 첫째, 수정의 경우에는 먼저 기존 문서를 삭제한다. 둘째, 추가된 문서 또는 수정된 문서로부터 키워드를 추출하고 각 키워드에 대한 포스팅을 만든다. 세째, 추출한 키워드를 키 값으로 하여 midx1을 탐색함으로써 주어진 키워드의 포스팅 리스트를 가진 부인덱스 sidxi를 찾는다. 네째, 찾은 부인덱스 sidxi를 통하여 해당 포스팅을 삽입한다. 부인덱스의 키가 문서 식별자이므로 포스팅은 문서 식별자 순으로 삽입된다. 다섯째, 추출한 키워드 모두에 대해 위의 세째와 네째 단계를 반복한다. 여섯째, 포스팅을 추가한 부인덱스들을 midx2의 해당 단말 노드 인덱스 엔트리에 연결 시킨다. 해당 단말 노드 인덱스 엔트리란 추가 또는 수정된 문서의 식별자를 키 값으로 하는 단말 노드 인덱스 엔트리를 말한다.

인덱스 그래프에서 문서의 동적 삭제는 다음과 같이 이루어진다. 첫째, 삭제된 문서의 식별자가 포함된 포스팅 리스트들을 구하기 위하여 삭제된 문서의 식별자를 키 값으로 하여 midx2를 탐색함으로써 삭제된 문서의 식별자가 포함된 포스팅 리스트를 가진 부인덱스들을 찾는다. 둘째, 찾은 부인덱스를 이용하여 삭제된 문서의 식별자가 포함된 포스팅을 삭제한다. 부인덱스는 문서 식별자가 키이므로 삭제된 문서의 식별자가 포함된 포스팅을 빠르게 찾아 삭제할 수 있다.

Ⅳ. 분석 및 실험

본 장에서는 기존 IR 인덱스 구조(이후 역 인덱스라 부른다)와 본 논문에서 제안한 인덱스 그래프의 정보검색 비용, 문서 추가 비용, 그리고 문서 삭제 비용 등을 비교해 본다. 문서의 수정은 기존 문서를 삭제하고 수정된 문서를 추가하는 것으로 간주한다. 비용은 디스크 페이지 접근 횟수로 계산하며 비용의 비교는 먼저 비용 모

델을 세우고 이를 사용하여 비교 분석을 한 다음 실제 실험을 통하여 이를 검증하는 방법을 택한다.

분석 및 실험에 사용된 조건은 다음과 같다. 첫째, 주인덱스와 부인덱스 그리고 역 인덱스의 데이타 구조로 B+ 트리를 사용한다. 둘째, 포스팅 리스트의 크기가 단말 노드 크기를 초과하는 경우 긴 포스팅 리스트라 부르고 그 이하의 경우 짧은 포스팅 리스트라 부르며 각각의 저장 방법을 달리 한다. 긴 포스팅 리스트는 역 인덱스의 경우에 별도의 오버플로우 페이지에 저장되고 역 인덱스의 단말 노드에는 이에 대한 포인터가 저장되며, 인덱스 그래프의 경우에 별도의 부인덱스를 이루고 주인덱스 midx1의 단말 노드에는 부인덱스의 식별자가 저장된다. 짧은 포스팅 리스트는 역 인덱스의 경우에 역 인덱스의 단말 노드 내에 키와 함께 저장되고, 인덱스 그래프의 경우에 주인덱스 midx1의 단말 노드 내에 키와 함께 저장된다. 세째, 125,000건의 실제 신문기사 데이타베이스를 사용하여 분석 및 실험을 한다. <표 4-1>은 분석에 사용된 파라미터들이다.

<표 4-1> 분석에 사용된 파라미터

기호	의미
D	총 문서 개수
N_w	한 문서당 평균 키워드 개수
N_{long}	긴 포스팅 리스트의 개수
N_{short}	짧은 포스팅 리스트의 개수
P_{short}	짧은 포스팅 리스트의 포스팅 평균 개수
$L_{keyword}$	키워드의 평균 크기
L_{docID}	문서식별자의 크기
$L_{posting}$	포스팅의 평균 크기
L_{ptr}	노드 포인터의 크기
L_{node}	노드의 크기
$b(idx)$	인덱스 idx의 내부 노드 블록킹 계수(blocking factor)
$N_{leaf}(idx)$	인덱스 idx의 단말 노드 개수
$P(w)$	키워드 w의 포스팅 개수

4.1 정보검색 비용

정보검색 비용은 주어진 키워드에 대한 포스팅 리스트를 찾기 위하여 키워드 인덱스를 탐색하는 비용(이하 인덱스-탐색 비용)과 찾은 포스팅 리스트를 순차적으로 읽는 비용(이하 포스팅리스트-스캔 비용)의 합이다.

4.1.1 역 인덱스 비용 모델

(1) 인덱스-탐색 비용

역 인덱스의 인덱스-탐색 비용 *IA(InvIndex)*은 <그림 2-1>의 역 인덱스의 루트 노드부터 단말 노드까지 참조하는 노드의 개수 즉, 인덱스 높이와 같다.

$$IA(InvIndex) = \lceil \log_{b(InvIndex)} N_{\text{leaf}}(InvIndex) \rceil + 1$$

b(InvIndex)는 역 인덱스의 내부 노드 인덱스 엔트리 구조가 <키워드, 서브트리 포인터>이므로 다음과 같다.

$$b(InvIndex) = \frac{L_{node}}{L_{keyword} + L_{ptr}}$$

Nleaf(InvIndex)는 역 인덱스의 모든 단말 노드 인덱스 엔트리들의 크기의 합을 Lnode로 나눈 것과 같다. 단말 노드 인덱스 엔트리의 구조는 두가지로 분류된다. 첫째는 <키워드, 포스팅 리스트>의 구조로서 짧은 포스팅 리스트를 저장하고 있는 경우이고, 둘째는 <키워드, 페이지 포인터>의 구조로서 긴 포스팅 리스트를 저장하고 있는 오버플로우 페이지를 가리키는 경우이다.

전자의 단말 노드 인덱스 엔트리의 크기는 $L_{keyword} + L_{posting} \times P_{short}$ 이고, 후자의 단말 노드 인덱스 엔트리의 크기는 $L_{keyword} + L_{ptr}$ 이다. 따라서,

$$N_{\text{leaf}}(InvIndex) = \left\lceil \frac{N_{short} \times (L_{keyword} + L_{posting} \times P_{short}) + N_{long} \times (L_{keyword} + L_{ptr})}{L_{node}} \right\rceil$$

(2) 포스팅리스트-스캔 비용

주어진 키워드 w에 대한 역 인덱스의 포스팅리스트-스캔 비용 PS(InvIndex, w)는 키워드 w의 포스팅 리스트가 긴 것이냐 또는 짧은 것이냐에 따라 다르다. 긴 것일 경우에는 포스팅 리스트가 오버플로우 페이지에 별도로 저장되어 있으므로 이를 읽는 비용이 들고, 짧은 것일 경우에는 인덱스-탐색에서 이미 읽은 단말 노드에 포스팅 리스트가 저장되어 있으므로 별도의 비용이 들지 않는다. 주어진 키워드 w의 포스팅 리스트가 저장된 오버플로우 페이지의 개수는 포스팅 리스트의 크기가 $L_{posting} \times P(w)$ 이므로 이를 Lnode로 나누면 얻을 수 있다. 따라서 PS(InvIndex, w)를 수식으로 표현하면 다음과 같다.

$$PS(InvIndex, w) = \begin{cases} \lceil \frac{L_{posting} \times P(w)}{L_{node}} \rceil, & w\text{가 긴 포스팅리스트를 가질 경우} \\ 0, & w\text{가 짧은 포스팅리스트를 가질 경우} \end{cases}$$

4.1.2 인덱스 그래프 비용 모델

(1) 인덱스-탐색 비용

인덱스 그래프의 인덱스-탐색 비용은 <그림 3-2>에서 주인덱스 midx1의 인덱스-탐색 비용 IA(midx1)으로서 다음과 같다.

$$IA(midx_1) = \lceil \log_{b(midx_1)} N_{leaf}(midx_1) \rceil + 1$$

주인덱스 midx1의 내부 노드 인덱스 엔트리 구조가 <키워드, 서브트리 포인터>이므로 b(midx1)은 다음과 같다.

$$b(midx_1) = \frac{L_{node}}{L_{keyword} + L_{ptr}}$$

Nleaf(midx1)은 주인덱스 midx1의 모든 단말 노드 인덱스 엔트리들의 크기의 합을 Lnode로 나눈 것과 같다. 단말 노드 인덱스 엔트리의 구조는 두 가지로 분류된다. 첫째는 <키워드, 포스팅 리스트>의 구조로서 짧은 포스팅 리스트를 저장하고 있는 경우이고, 둘째는 <키워드, 부인덱스 식별자>의 구조로서 긴 포스팅 리스트를 저장하고 있는 부인덱스를 가리키는 경우이다.

전자의 단말 노드 인덱스 엔트리의 크기는 $L_{keyword} + L_{posting} \times P_{short}$ 이고 후자의 단말 노드 인덱스 엔트리의 크기는 $L_{keyword} + L_{ptr}$이다. 따라서,

$$N_{leaf}(midx_1) = \lceil \frac{N_{short} \times (L_{keyword} + L_{posting} \times P_{short}) + N_{long} \times (L_{keyword} + L_{ptr})}{L_{node}} \rceil$$

(2) 포스팅리스트-스캔 비용

주어진 키워드 w에 대한 인덱스 그래프의 포스팅리스트-스캔 비용 PS(IdxGraph,w)도 키워드 w의 포스팅 리스트가 긴 것이냐 또는 짧은 것이냐에 따라 비용이 다르다. 긴 것일 경우에는 포스팅 리스트가 부인덱스를 이루고 있으므로 부인덱스의 단말 노드들을 순차 검색하는 비용이 들고, 짧은 것일 경우에는 이미 주인덱스 midx1의 단말 노드에 포스팅 리스트가 포함되어 있으므로 별도의 비용이 들지 않는다. 부인덱스의 단말 노드들을 순차 검색하는 비용은 부인덱스의 단말 노드 개수와 같고 이는 부인덱스의 모든 단말 레벨 인덱스 엔트리들의 크기의 합을 Lnode로 나눈

것과 같다. 부인덱스에서는 하나의 포스팅이 하나의 단말 레벨 인덱스 엔트리를 이루기 때문에 단말 레벨 인덱스 엔트리의 크기는 포스팅 한 개의 크기와 같고, 단말 레벨 인덱스 엔트리의 개수는 키워드 w의 포스팅 개수와 같다. 따라서 PS(IdxGraph,w)를 수식으로 표현하면 다음과 같다.

$$PS(IdxGraph, w) = \begin{cases} \lceil \frac{L_{posting} \times P(w)}{L_{node}} \rceil + 1, & w\text{가 긴 포스팅리스트를 가질 경우} \\ 0, & w\text{가 짧은 포스팅리스트를 가질 경우} \end{cases}$$

w가 긴 포스팅 리스트를 가지는 경우에 1 을 더해 주는 이유는 부인덱스의 첫 번째 단말 노드 위치를 알기 위해 부인덱스의 루트 노드를 읽어야 하기 때문이다*

4.1.3 비교 분석 및 실험

역 인덱스와 인덱스 그래프에 대하여 비용 모델에서 사용된 파라미터들의 통계값을 구해 보면 <표 4-2>와 같다. <표 4-2>에서 인덱스 그래프의 Lposting이 역 인덱스의 것보다 4 만큼 더 큰 이유는 각 포스팅마다 포스팅의 크기를 나타내는 4 바이트 크기의 필드가 저장되어 있기 때문이다. 이것은 현재 구현된 부인덱스가 포스팅의 데이타 구조를 모르기 때문에 이를 통하여 특정 포스팅을 찾아 읽을 때에 그 포스팅의 크기를 알아야 하므로 필요하다. 이에 따라 긴 포스팅 리스트의 개수도 역 인덱스보다 인덱스 그래프에서 더 많다.

<표 4-2> 파라미터에 대한 통계치

파라미터	역 인덱스	인덱스 그래프
N_{long}	10,000	11,432
N_{short}	1,905,854	1,904,422
P_{short}	3.3	3.2
$L_{keyword}$	10.6	10.6
$L_{posting}$	27	31
L_{ptr}	4	4
L_{node}	4096	4096

<그림 4-1>의 (a)는 주어진 키워드의 포스팅 개수에 따라 역 인덱스와 인덱스 그래프의 비용 모델을 통한 정보검색 비용을 나타낸 것이고, <그림 4-1>의 (b)는 실험을 통한 정보검색 비용을 나타낸 것이다.

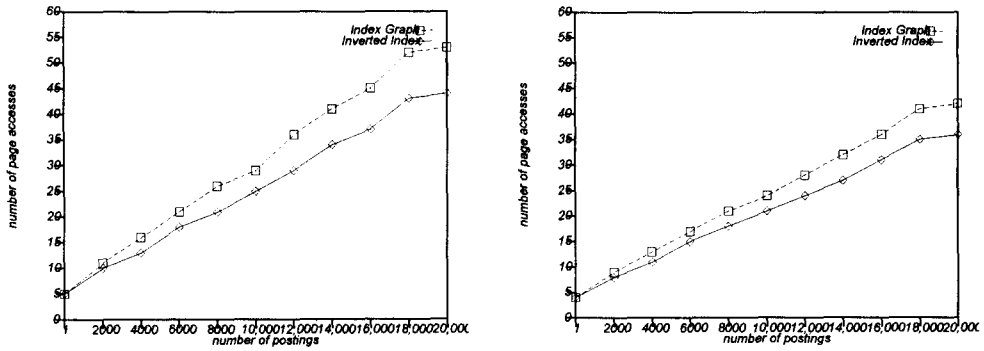
역 인덱스와 인덱스 그래프의 정보검색 비용 모델은 같지만 분석 및 실험 결과

* 부인덱스의 루트 노드에는 첫번째 단말 노드를 가리키는 포인터를 가지고 있다.

인덱스 그래프의 정보검색 비용이 역 인덱스보다 높게 나온 이유는 위에서 언급한 포스팅의 크기 차이로 인하여 인덱스 그래프의 포스팅 리스트 크기가 더 커졌기 때문이다. 따라서 인덱스 그래프의 부인덱스를 포스팅 구조를 이해하도록 구현한다면 포스팅 리스트에서 포스팅 크기 필드가 없어지므로 역 인덱스와 인덱스 그래프의 정보검색 비용이 같아질 것으로 예상된다.

<그림 4-1>의 (a)와 (b)에서 보듯이 분석과 실험은 거의 같은 결과를 보여주고 있지만 실험치가 분석치보다 높은 이유는 다음과 같다. 첫째, 긴 포스팅 리스트의 경우 실제 실험에서는 대용량 객체에 저장되고 대용량 객체는 일정 크기(실험에서는 16K바이트) 단위로 나뉘어 트리 구조를 통해 관리된다. 이에 따라 대용량 객체를 순차적으로 스캔하기 위해서는 트리 탐색이 발생하므로 이의 비용이 더 추가되었다. 둘째, 대용량 객체에 대한 스캔을 위해서는 카탈로그와 같은 시스템 테이블의 접근을 위한 비용이 더 추가되었다.

(a) (b)



<그림 4-1> 역 인덱스와 인덱스 그래프의 정보검색 비용 비교

4.2 문서추가 비용

문서가 새로이 추가되면 추가된 문서로부터 추출된 모든 키워드에 대하여 각 키워드의 포스팅 리스트를 찾아서, 추가된 문서의 포스팅을 문서 식별자 순으로 삽입하여야 한다. 따라서 N개의 키워드를 가진 문서의 추가 비용은 N개의 키워드 각각에 대한 포스팅 리스트를 찾기 위하여 키워드 인덱스를 N번 탐색하는 비용과 찾은 N개의 포스팅 리스트에 포스팅을 문서 식별자 순으로 삽입하는 비용의 합이다. 키워드 인덱스를 탐색하는 비용은 제 4.1절의 인덱스-탐색 비용 IA(Idx)과 동일하고, 주어진 키워드 w의 포스팅 리스트에 새로운 포스팅을 문서 식별자 순으로 삽입하는 비용은 키워드 w의 포스팅 리스트에서 특정 문서 식별자를 가진 포스팅을 찾는 비용(이하 포스팅-검색 비용) PS(Idx,w)과 동일하므로 N개의 키워드를 가진 문서의 추가 비용은

$$\sum_{i=1}^{N}(IA(Idx)+PS(Idx,w_i))$$

와 같다. 다음은 역 인덱스와 인덱스 그래프의 포스팅-검색 비용 모델이다.

4.2.1 역 인덱스 비용 모델

주어진 키워드 w에 대한 역 인덱스의 포스팅-검색 비용 PS(InvIndex,w)는 주어진 키워드 w의 포스팅 리스트가 긴 것이냐 또는 짧은 것이냐에 따라 다르다. 짧은 포스팅 리스트는 <그림 2-1>의 인덱스 idx1의 단말 노드에 저장되어 있으므로 별도의 비용이 들지 않지만, 긴 포스팅 리스트는 오버플로우 페이지에 별도로 저장되어 있으므로 이를 처음부터 순차적으로 읽는 비용이 든다. 따라서 평균적으로는 포스팅-검색 비용이 포스팅리스트-스캔 비용의 $\frac{1}{2}$과 같다. 즉,

$$PS(InvIndex,w)=\begin{cases}\frac{1}{2}\times\lceil\frac{L_{posting}\times P(w)}{L_{node}}\rceil, & w\text{가 긴 포스팅리스트를 가질 경우}\\ 0, & w\text{가 짧은 포스팅리스트를 가질 경우}\end{cases}$$

4.2.2 인덱스 그래프 비용 모델

주어진 키워드 w에 대한 인덱스 그래프의 포스팅-검색 비용 PS(IdxGraph,w)도 주어진 키워드 w의 포스팅 리스트가 긴 것이냐 또는 짧은 것이냐에 따라 다르다. 짧은 포스팅 리스트는 <그림 3-2>의 인덱스 midx1의 단말 노드에 저장되어 있으므로 별도의 비용이 들지 않지만, 긴 포스팅 리스트는 부인덱스 sidxi에 저장되어 있으므로 부인덱스에 대한 탐색 비용이 든다. 즉,

$$PS(IdxGraph,w)=\begin{cases}\lceil\log_{b(sidx_i)}N_{\text{leaf}}(sidx_i)\rceil+1, & w\text{의 포스팅리스트가 길 경우}\\ 0, & w\text{의 포스팅리스트가 짧을 경우}\end{cases}$$

b(sidxi)는 부인덱스 sidxi의 내부 노드 인덱스 엔트리 구조가 <문서식별자, 서브트리 포인터>이므로 다음과 같다.

$$b(sidx_i)=\frac{L_{node}}{L_{docID}+L_{ptr}}$$

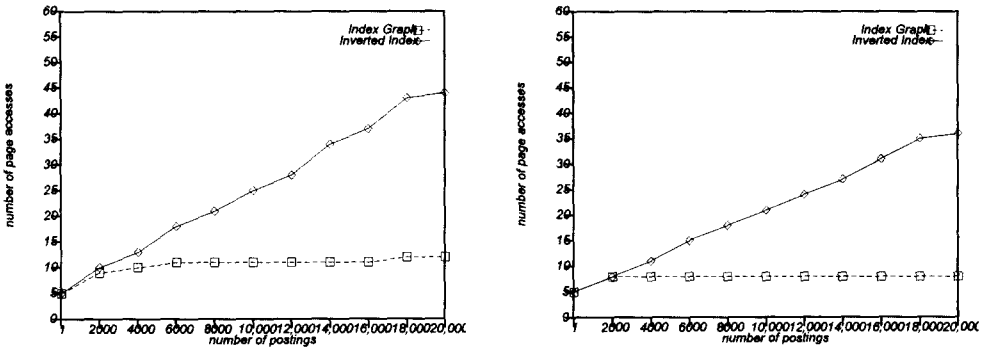
Nleaf(sidxi)는 부인덱스 sidxi의 모든 단말 노드 인덱스 엔트리들의 크기의 합을 Lnode로 나눈 것과 같다. 부인덱스에서는 하나의 포스팅이 하나의 단말 레벨 인덱스 엔트리를 이루므로 단말 레벨 인덱스 엔트리의 크기는 포스팅의 크기와 같고 개수는 주어진 키워드의 포스팅 개수와 같다. 따라서,

$$N_{\text{leaf}}(sidx_i) = \lceil \frac{L_{posting} \times P(w)}{L_{node}} \rceil$$

4.2.3 비교 분석 및 실험

본 비교에서는 한 개의 키워드로 이루어진 문서의 추가 비용을 그 키워드의 포스팅 개수에 따라 비교해 본다. 인덱스 그래프의 비용 모델에서 사용된 문서 식별자의 크기를 LdocID = 4 로 하였다. <그림 4-2>의 (a)는 주어진 키워드의 포스팅 개수에 따라 역 인덱스와 인덱스 그래프의 비용 모델을 통한 문서 추가 비용을 나타낸 것이고, <그림 4-2>의 (b)는 실험을 통한 문서 추가 비용을 나타낸 것이다.

(a) (b)



<그림 4-2> 역 인덱스와 인덱스 그래프의 문서추가 비용 비교

역 인덱스와 인덱스 그래프의 문서추가 비용 모델에서 보는 바와 같이 분석 및 실험 결과에서도 인덱스 그래프의 문서 추가 비용은 추가된 문서에 포함된 키워드의 포스팅 리스트 크기에 큰 영향을 받지 않고 일정한 값을 유지한다. 그러나, 역 인덱스에서는 키워드의 포스팅 리스트 크기에 따라 선형적으로 값이 증가한다. 따라서 긴 포스팅 리스트를 가진 키워드가 많이 포함된 문서의 추가 비용은 인덱스 그래프에 비해 월등히 크다. 또한 <그림 4-2>의 (a)와 (b)에서 보듯이 분석과 실험은 거의 같은 결과를 보여주고 있지만 실험치가 분석치보다 높은 이유는 제 4.1절의 정보검색 비용에서와 같이 실험에서는 대용량 객체를 위한 트리 탐색 비용과 카탈로그 테이블 접근 비용이 더 추가되었기 때문이다.

4.3 문서삭제 비용

문서가 삭제되면 삭제된 문서에 포함된 모든 키워드에 대하여 각 키워드의 포스팅 리스트를 찾아서, 삭제된 문서의 포스팅을 삭제해 주어야 한다. 따라서 N개의 키워드를 가진 문서의 삭제 비용은 N개의 키워드 각각에 대한 포스팅 리스트를 찾는

비용(이하 삭제포스팅리스트-탐색 비용)과 찾은 N개의 포스팅 리스트로부터 삭제된 문서의 포스팅을 삭제하는 비용(이하 삭제포스팅-탐색 비용)의 합이다.

4.3.1 역 인덱스 비용 모델

(1) 삭제포스팅리스트-탐색 비용

역 인덱스에서는 삭제된 문서에 포함된 N개의 키워드에 대한 포스팅 리스트들을 찾기 위하여 키워드 인덱스를 N번 탐색하여야 하고 키워드 인덱스 탐색 비용은 제 4.1절의 인덱스-탐색 비용 IA(InvIndex)와 동일하다. 따라서 삭제포스팅리스트-탐색 비용 PLS(InvIndex)는 다음과 같다.

$$PLS(InvIndex) = N \times IA(InvIndex)$$

(2) 삭제포스팅-탐색 비용

역 인덱스에서는 주어진 키워드 w에 대한 포스팅 리스트에서 삭제된 문서의 포스팅을 찾아 삭제하기 위해서는 포스팅 리스트를 처음부터 찾을 때까지 순차적으로 읽어 보아야 하는데 이는 제 4.2절의 포스팅-검색 비용 PS(InvIndex,w)와 동일하다. 따라서 삭제포스팅-탐색 비용 NPS(InvIndex)는 다음과 같다.

$$NPS(InvIndex) = \sum_{i=1}^{N} PS(InvIndex, w_i)$$

4.3.2 인덱스 그래프 비용 모델

(1) 삭제리스트-탐색 비용

인덱스 그래프에서는 삭제된 문서에 포함된 N개의 키워드에 대한 포스팅 리스트들을 찾기 위하여 <그림 3-2>의 주인덱스 midx2를 한 번 탐색하면 되므로 삭제포스팅리스트-탐색 비용 PLS(IdxGraph)는 주인덱스 midx2의 인덱스-탐색 비용 IA(midx2)와 동일하다. 따라서

$$PLS(IdxGraph) = IA(midx_2) = \lceil \log_{b(midx_2)} N_{\text{leaf}}(midx_2) \rceil + 1$$

b(midx2)는 주인덱스 midx2의 내부 노드 인덱스 엔트리 구조가 <문서 식별자, 서브트리 포인터>이므로 다음과 같다.

$$b(midx_2) = \frac{L_{node}}{L_{docID} + L_{ptr}}$$

Nleaf(midx2)는 주인덱스 midx2의 모든 단말 노드 인덱스 엔트리들의 크기의

합을 Lnode로 나눈 것과 같다. 단말 노드 인덱스 엔트리의 크기는 $L_{docID} + L_{ptr} \times N_w$ 와 같고 총 개수는 문서의 총 개수와 같다. 따라서,

$$N_{\text{leaf}}(midx_2) = \lceil \frac{D \times (L_{docID} + L_{ptr} \times N_w)}{L_{node}} \rceil$$

(2) 삭제포스팅-탐색 비용

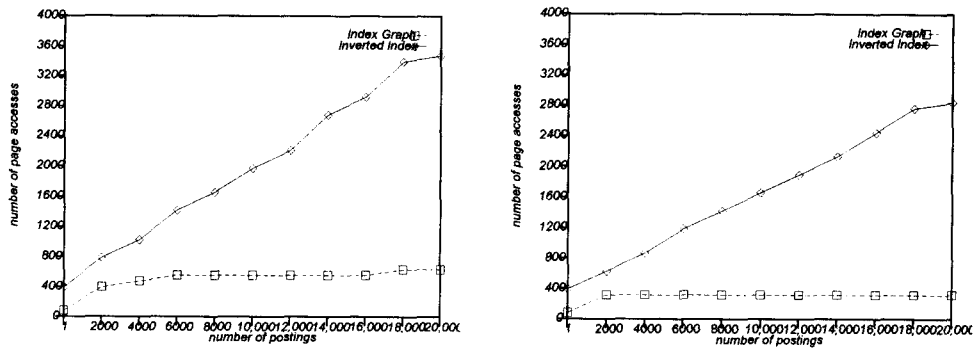
인덱스 그래프에서는 주어진 키워드 wi에 대한 포스팅 리스트에서 삭제된 문서의 포스팅을 찾아 삭제하기 위해서는 부인덱스 sidxi를 탐색하여야 하는데 이는 제 4.2절의 포스팅-검색 비용 PS(IdxGraph, wi)와 동일하다. 따라서 삭제포스팅-탐색 비용 NPS(IdxGraph)는 다음과 같다.

$$NPS(IdxGraph) = \sum_{i=1}^{N} PS(IdxGraph, w_i)$$

4.3.3 비교 분석 및 실험

본 비교에서는 평균 Nw 개의 키워드로 이루어진 문서의 삭제 비용을 한 키워드에 대한 포스팅 개수에 따라 비교해 본다. 인덱스 그래프의 비용 모델에서 사용된 파라미터의 값은 다음과 같다: LdocID = 4, D = 125,000, Nw = 79. <그림 4-3>의 (a)는 주어진 키워드의 포스팅 개수에 따라 역 인덱스와 인덱스 그래프의 비용 모델을 통한 문서 삭제 비용을 나타낸 것이고, (b)는 실험을 통한 문서 삭제 비용을 나타낸 것이다.

(a) (b)



<그림 4-3> 역 인덱스와 인덱스 그래프의 문서삭제 비용 비교

역 인덱스와 인덱스 그래프의 문서 삭제 비용 모델에서 보는 바와 같이 분석 및 실험 결과에서도 인덱스 그래프의 문서 삭제 비용은 삭제된 문서에 포함된 키워드의

포스팅 리스트 크기에 큰 영향을 받지 않고 거의 일정한 값을 유지한다. 그러나, 역 인덱스에서는 키워드의 포스팅 리스트 크기에 따라 선형적으로 값이 증가한다. 특히 역 인덱스의 문서 삭제 비용에는 삭제된 문서에 포함된 키워드를 구하는 비용이 포함되지 않았다. 따라서 이 비용을 고려하면 문서 삭제 비용은 더 증가한다. 따라서 긴 포스팅 리스트를 가진 키워드가 많이 포함된 문서의 삭제 비용은 인덱스 그래프에 비해 월등히 크다. 또한 <그림 4-3>의 (a)와 (b)에서 보듯이 분석과 실험은 거의 같은 결과를 보여주고 있지만 실험치가 분석치보다 높은 이유는 제 4.2절의 문서 추가 비용에서와 같이 실험에서는 대용량 객체를 위한 트리 탐색 비용과 카탈로그 테이블 접근 비용이 더 추가되었기 때문이다.

V. 결론

전자도서관, 인터넷 검색엔진, 멀티미디어 데이터베이스 등과 같은 응용은 새로운 문서의 추가와 기존 문서의 수정 또는 삭제가 빈번히 발생하는 동적 문서 데이터베이스를 가진다. 동적 문서 데이터베이스를 위한 IR 인덱스 구조는 정보검색 및 문서 추가, 삭제, 수정의 성능 향상을 위하여 다음과 같은 사항들을 만족하여야 한다. 첫째, 문서의 추가 또는 수정에 따라 새로운 포스팅들을 IR 인덱스에 추가할 때 포스팅 리스트의 문서 식별자 순 정렬이 효율적으로 유지되어야 한다. 둘째, 문서의 삭제에 따라 삭제된 문서의 포스팅들을 IR 인덱스에서 삭제할 때 삭제할 포스팅 리스트의 탐색과 포스팅 리스트에서의 포스팅 삭제가 효율적으로 이루어져야 한다. 그러나, 기존 IR 인덱스 구조는 문서의 추가, 삭제, 수정이 발생하지 않는 정적 문서 데이타베이스를 위한 정보검색 만을 고려하였으므로 동적 문서 데이타베이스를 위한 위의 요구사항들을 만족하지 못한다.

본 논문에서는 동적 문서 데이타베이스를 위한 위의 요구 사항들을 만족할 수 있는 새로운 IR 인덱스 구조로서 인덱스 그래프를 제안하였다. 인덱스 그래프는 다음과 같은 세 가지 종류의 인덱스들을 구축하고 이들을 계단식 관계를 통해 서로 연결시켜서 하나의 인덱스처럼 사용한다. 첫째, 각 포스팅 리스트마다 문서 식별자 순 정렬을 유지하기 위하여 문서 식별자를 주 키(primary key)로 하는 인덱스를 구축하였다. 이 인덱스는 문서의 추가, 삭제, 수정 시 주어진 문서 식별자를 가진 포스팅을 삽입, 삭제, 또는 검색해 준다. 둘째, 정보검색 시 주어진 키워드에 해당하는 포스팅 리스트를 찾기 위하여 키워드를 키로 하는 인덱스를 구축하고 이 인덱스가 각 포스팅 리스트에 구축된 인덱스들을 가리키도록 하였다. 세째, 문서 삭제 시 삭제된 문서의 식별자가 포함된 포스팅 리스트들을 찾기 위하여 문서 식별자를 키로 하는 인덱스를 구축하고 이 인덱스가 각 포스팅 리스트에 구축된 인덱스들을 가리키도록 하였다. 분석 및 실험을 통하여 기존 IR 인덱스 구조는 동적 문서 데이타베이스에서 문서의 추

가, 삭제 성능이 현저히 저하됨을 보였고 인덱스 그래프 구조의 IR 인덱스는 기존의 IR 인덱스 구조와 같은 정도의 정보검색 성능을 가지면서도 문서의 동적 추가, 삭제, 수정에 따른 인덱스 갱신 성능이 기존의 IR 인덱스 구조보다 훨씬 우수함을 보였다.

IR 인덱스 구조로서의 인덱스 그래프는 서로 다른 인덱스 구조를 연결할 수 있다는 또 다른 장점을 가진다. 즉, 키워드에 대한 인덱스 구조와 문서 식별자에 대한 인덱스 구조를 각각에 가장 적합한 것으로 선택할 수 있다는 것이다. 예를 들어, 키워드에 대한 인덱스 구조로서 트라이(Trie) 구조가 적합하고, 문서 식별자에 대한 인덱스 구조로서 B+ 트리가 적합할 때 구조가 다른 이들 두 인덱스를 주인덱스와 부인덱스로 서로 연동시켜 사용하는 것이 가능하다.

향후 연구로서, 다차원 인덱스 구조를 이용한 IR 인덱스 구조와 인덱스 그래프를 이용한 IR 인덱스 구조의 성능을 비교해 보고자 한다. 인덱스 그래프는 여러 개의 키를 사용할 수 있다는 점에서 다차원 인덱스 구조로 볼 수 있으나, 기존의 다차원 인덱스 구조는 하나의 인덱스 구조를 사용하는데 비하여, 인덱스 그래프는 여러 개의 인덱스 구조를 서로 연결하여 사용할 수 있다는 장점을 가진다. 즉, 각 키에 대한 인덱스 구조는 각각에 가장 적합한 것을 선택할 수 있으면서도 이들을 서로 연결하여 다차원 인덱스 구조로 사용할 수 있다는 것이다. 따라서, 동적 문서 데이타베이스를 위한 IR 인덱스의 경우 키워드와 문서 식별자를 두 키로 하는 이차원 인덱스 구조를 생각해 볼 수 있으며 이와의 성능 비교가 필요하다.

참고문헌

Baeza-Yates, R. A. and Larson, P. (1989), "Performance of B^+-Trees with Partial Expansions," *IEEE Trans. on Knowledge and Data Engineering,* Vol. 1, No. 2, pp. 248-257.

Blair, D. C. (1988), "An Extended Relational Document Retrieval Model," *Information Processing and Management,* Vol. 24, No. 3, pp. 349-371.

Blake, G. E. et al. (1995), "Text/Relational Database Management Systems: Overview and Proposed SQL Extensions," Tech. Report CS-95-25, Univ. of Waterloo, Canada.

Brown, E. W. et al. (1994), "Fast Incremental Indexing for Full-Text Information Retrieval," *Proc. Intl. Conf. on Very Large Data Bases*, Santiago, 192-202.

Comer, D. (1979), "The ubiquitous B-tree", *ACM Computing Surveys*, Vol. 11, No.2, pp. 345-363.

Cutting, D. and Pedersen, J. (1990), "Optimizations for Dynamic Inverted Index Maintenance," *Proc. Intl. Conf. on Information Retrieval*, Brussels, pp. 405-411.

DeFazio, S. et al. (1995), "Integrating IR and RDBMS Using Cooperative Indexing," *Proc. Intl. Conf. on Information Retrieval*, Seattle, pp. 84-92.

Elmasri, R. and Navathe, S. B. (1994), *Fundamentals of Database Systems*, Benjamin/Cummings.

Faloutsos, C. and Jagadish, H. V. (1992), "On B-tree Indices for Skewed Distribution", *Proc. Intl. Conf. on Very Large Data Bases,* Vancouver, pp. 363-374.

Faloutsos, C. and Oard, D. W. (1995), "A Survey of Information Retrieval and Filtering Methods", Tech. Report CS-TR-3514, Univ. of Maryland.

Frakes, W. B. and Baeza-Yates, R. (1992), Information Retrieval - Data Structures & Algorithms, Prentice Hall.

Kriegel, Hans-Peter (1984), "Performance Comparison of Index Structures for Multi-Key Retrieval," *Proc. Intl. Conf. on Management of Data,* ACM SIGMOD, Boston, pp. 186-196.

Lee, W. L. and Woelk, D. (1990), "Integration of Text Search with ORION," *IEEE Data Engineering Bulletin,* Vol. 13, No. 1, pp. 58-64.

Lee, J. H. and Whang, K. Y. (1997), "A Region Splitting Strategy for Physical Database Design of Multidimensional File Organizations," *Proc. Intl. Conf. on Very Large Data Bases*, Athens.

Lynch, C. A. and Stonebraker, M. (1988), "Extending User-Defined Indexing with Application to Textual Databases," *Proc. Intl. Conf. on Very Large Data Bases,* Los Angeles, pp. 306-317.

Mohan, C. et al. (1990), "Single Table Access Using Multiple Indexes: Optimization, Execution, and Concurrency Control Techniques," *Proc. Intl. Conf. On Extending Database Technology*, Venice, pp. 29-43.

Nievergelt, J. et al. (1984), "The Grid File: An Adaptable, Symmetric Multikey File Structure," *ACM Trans. on Database Systems,* Vol. 9, No. 1, pp. 38-71.

Ribeiro-Neto, B. et al. (1999), "Efficient Distributed Algorithms to Build Inverted Files," *Proc. Intl. Conf. on Research and Development in Information Retrieval,* ACM SIGIR, Berkeley, pp. 105-112.

Robinson, J. T. (1981), "The K-D-B-Tree: A Search Structure for Large Multidimensional Dynamic Indexes," *Proc. Intl. Conf. on Management of Data,* ACM SIGMOD, Ann Arbor, pp. 10-18.

Sacks-Davis, R. et al. (1995), "Atlas: A Nested Relational Database System for Text Applications," *IEEE Trans. on Knowledge and Data Engineering,* Vol. 7, No. 3, pp. 454-470.

Salton, G. (1988), Automatic Text Processing - The Transformation, Analysis, and Retrieval of Information by Computer, Addison-Wesley.

Tomasic, A., Garcia-Molina, H., and Shoens, K. (1994), "Incremental Updates of Inverted Lists for Text Document Retrieval," *ACM SIGMOD RECORD*, Vol. 23, No. 2, pp. 289-300.

Vasanthakumar, S. R., Callan, J. P., and Croft, W. B. (1996), "Integrating INQUERY with an RDBMS to Support Text Retrieval," *IEEE Data Engineering Bulletin*, Vol. 19, No. 1, pp. 24-33.

Whang, K. Y. and Krishnamurthy, R. (1991), "The Multilevel Grid File - A Dynamic Hierarchical Multidimensional File Structure," *Proc. Intl. Conf. on Database Systems for Advanced Applications*, Tokyo, pp. 449-459.

Zipf, G. K. (1949), *Human Behavior and the Principle of Least Effort*, Addison-Wesley.

Zobel,~J. et~al. (1991), "Efficiency of Nested Relational Document Database Systems," *Proc. Intl. Conf. on Very Large Data Bases*, Barcelona, pp. 91-102.

<Abstract>

Index Graph : An IR Index Structure for Dynamic Document Databases

Byung-Kwon Park Dong-A University bpark@daunet.donga.ac.kr

An IR(information retrieval) index for dynamic document databases where insertion, deletion, and update of documents happen frequently should be frequently updated. As the conventional structure of IR index is, however, focused on the information retrieval purpose, its structure is inefficient to handle dynamic update of it. In this paper, we propose a new structure for IR Index, we call it Index Graph, which is organized by connecting multiple indexes into a graph structure. By analysis and experiment, we prove the Index Graph is superior to the conventional structure of IR index in the performance of insertion, deletion, and update of documents as well as the performance of information retrieval.

key words : information retrieval, inverted index, index structure, dynamic document Database