xPlaneb: XML 문서 검색을 위한 3차원 비트맵 인덱스

(xPlaneb: 3-Dimensional Bitmap Index for XML Document Retrieval)

이 재 민 † 황 병 연 ††
(Jae-Min Lee) (Byung-Yeon Hwang)

요 약 XML은 다양한 장점을 통해 데이타를 표현하고 교환하기 위한 새로운 표준이 되었으며 현대의 많은 연구와 새로운 기술들에서 핵심적인 요소가 되고 있다. 그러나 XML의 장점인 자기 서술적인 특징은 구조적으로 상이한 XML 문서의 확산을 초래하게 되었고 이에 따라 XML의 효과적인 검색에 대한 연구의 필요성이 대두되게 되었다. 본 논문에서는 빠른 검색 속도를 통해 뛰어난 성능을 입증한 비트맵 인덱싱인 BitCube의 문제점을 분석한다. 또한 BitCube의 문제점을 해결하기 위해 연결 리스트를 이용한 새로운 3차원 비트맵 인덱싱인 xPlaneb(XML Plane Web)를 설계 및 구현한다. 제안된 기법은 BitCube의 3차원 배열 인덱스를 효율적인 노드로 재구성하고 BitCube의 연산을 대체하는 새로운 연산들을 활용하여 효과적으로 정보를 추출한다. 성능 평가를 통해 제안된 기법이 클러스터내의 문서의 양이 증가함에 따라 BitCube보다 메모리 사용량과 연산 수행 속도면에서 더 우수하다는 것을 보였다.

키워드 : XML, 정보검색, 비트맵 인덱싱, 연결 리스트

Abstract XML has got to be a new standard for data representation and exchanging by its many good points, and the core part of many new researches and emerging technologies. However, the self-describing characteristic, which is one of XML's good points, caused the spreading of XML documents with different structures, and so the need of the research for the effective XML-document search has been proposed. This paper is for the analysis of the problem in BitCube, which is a bitmap indexing that shows high performance grounded on its fast retrieval. In addition, to resolve the problem of BitCube, we did design and implement xPlaneb(XML Plane Web) which is a new 3-dimensional bitmap indexing made of linked lists. We propose an effective information retrieval technique by replacing BitCube operations with new ones and reconstructing 3-dimensional array index of BitCube with effective nodes. Performance evaluation shows that the proposed technique is better than BitCube, as the amount of document increases, in terms of memory consumptions and operation speed.

Key words : XML, Information Retrieval, Bitmap Indexing, Linked Lists

1. 서 론

W3C에 의해 1998년에 표준으로 제정된 XML (eXtended Markup Language)은 기존의 웹 컨텐츠가 표현에 중점을 둠으로써 발생하는 다양한 문제를 해결하기 위해 메타데이타에 대한 연구의 산물로 등장하였다[1]. XML은 문서의 구조를 표현할 수 있는 자기 서술적인(self-describing) 특징을 통해 데이타의 표현과 내용 그리고 구조를 분리시킴으로써 데이타를 표현하고 교환하는 새로운 표준으로 부상하였다. 이에 따라 현대의 많은 연구들과 새로운 기술들에서 XML이 핵심적인 요소로 자리매김하는 경우가 점차 많아지고 있다. 그러나 XML의 장점인 자기 서술적인 특징은 데이타의 구조를 표현하는 것에는 성공하였으나 구조가 상이한 수많은 XML 문서의 확산을 초래하게 됨으로써 XML을 효과적으로 저장 및 검색하고 이것을 활용하기 위한 연

· 본 연구는 2003년도 산학협동재단 학술연구비와 2004년도 가톨릭대학교 교비연구비의 지원으로 이루어졌음
† 학생회원 : 가톨릭대학교 컴퓨터공학과
likedawn@catholic.ac.kr
†† 종신회원 : 가톨릭대학교 컴퓨터정보공학부 교수
byhwang@catholic.ac.kr
논문접수 : 2003년 11월 7일
심사완료 : 2004년 2월 20일

구의 필요성이 대두되게 되었다.

XML에 대한 연구는 일반적으로 특정 자료를 표현하기 위한 구조를 미리 정의함으로써 XML의 구조적 상이성을 미연에 방지하기 위한 노력과 구조적으로 상이한 문서 그 자체를 효율적으로 저장 및 검색하기 위한 노력으로 나눌 수 있다. 구조적으로 상이한 문서에서 효과적으로 정보를 추출하기 위한 연구는 주로 기존의 RDBMS를 능동적으로 활용하기 위한 방법과 XML 문서의 정규표현을 효율적으로 재구성하는 방법에 대한 고찰을 통해 이루어진다. 현재에도 유명한 RDBMS 벤더들은 자신의 시스템에서 XML 검색을 효과적으로 지원하기 위한 방법을 제시하고 있으며 다양한 XML 전문 검색 시스템들이 등장하고 있다.

BitCube[2,3,4]는 XML 검색에서 빠른 검색 속도를 통해 뛰어난 성능을 입증한 3차원 비트맵 인덱싱이다. 본 논문에서는 BitCube가 클러스터내의 문서의 증가에 따라 급격하게 인덱스의 크기가 증가하게 되고 이로 인해 연산 수행 속도가 저하되는 문제점을 해결하기 위해 연결리스트를 이용한 새로운 3차원 비트맵 인덱싱인 xPlaneb를 설계 및 구현하고 성능을 평가한다. xPlaneb는 BitCube의 1비트의 필드들로 구성된 3차원 배열 인덱스를 효율적인 노드로 재구성함으로써 클러스터내의 문서의 증가에도 낮은 메모리 사용량을 유지하고 매우 안정적인 연산 수행 속도를 갖게 된다. 실험을 통해 우리는 클러스터내의 문서의 양이 증가할수록 xPlaneb가 BitCube보다 우수한 성능을 보임을 입증하였다.

본 논문의 구성은 다음과 같다. 2장에서는 XML 검색에 관하여 소개하고 BitCube와 그 연산들 그리고 그것의 문제점에 대하여 기술한다. 3장에서는 본 논문에서 제안된 연결 리스트를 이용한 3차원 비트맵 인덱싱인 xPlaneb에 대해 논의하고 4장에서는 실험을 통해 두 기법의 성능을 평가한다. 마지막으로 5장에서는 결론 및 향후 연구과제에 대하여 제시한다.

2. 관련 연구

본 장에서는 XML 검색과 본 논문의 기반이 되는 3차원 비트맵 인덱싱과 그 연산들에 대하여 소개한다.

2.1 XML 검색

XML 검색에 관한 연구는 크게 두 가지 방향에서 접근된다. 하나는 XML 문서의 목적에 따른 구조의 표준화를 통해 XML문서의 구조적 상이성을 감소시키는 것이고 다른 하나는 산재된 다양한 구조를 갖는 XML 문서 그 자체에서 효과적으로 정보를 추출하기 위한 연구이다. 실제로 기존의 유명한 RDBMS 벤더들도 Oracle XML DB[5], MS SQL Server[6], IBM DB2 Extender[7], Sybase ASE[8]와 같은 자신의 시스템을 통해 XML 검색을 효과적으로 수행할 수 있는 해결책을 제시하고 있다. 그러나 구조가 자주 변화하는 문서 중심의 XML 문서의 경우 기존의 RDBMS를 활용하여 이것을 저장 및 검색하는 것에는 한계가 있다. 최근에는 XYZFind[9], XQEngine[10], Xyleme[11]와 같이 XML과 같은 반 구조적인 문서를 전문적으로 검색하기 위한 시스템들도 등장하고 있다. 이와 같은 시스템들은 기존의 RDBMS를 활용하는 것보다는 순수한 XML 문서 그 자체에서 더 효율적으로 정보를 추출하기 위한 연구[12,13,14,15]라 할 수 있다. 이런 연구는 일반적으로 XML 문서의 정규 표현을 효율적으로 재구성하고 질의를 최적화함으로써 검색 속도를 향상시키고 필요한 기능을 효과적으로 지원할 수 있는 방법을 연구한다.

XML 검색을 위한 기존의 인덱싱 기법들은 대부분 트리 구조에 다양한 방법으로 XML 문서를 매핑시키고 트리의 노드를 단위로 하여 정보를 추출하거나 정보의 의미를 해석한다. 이것은 XML 문서 자체가 반구조적인 트리 형태인 것에 기인한다. 최근의 인덱싱 기법으로 트리를 부분적으로 분할하고 이것으로 계층적인 인덱스를 구성하여 탐색을 수행하는 기법[14]이 있으며 데이타베이스의 뷰와 같은 원리로 어떤 탐색을 위한 질의를 저장하고 후에 다시 이 질의를 활용하여 탐색을 수행하는 기법[15]도 연구되고 있다. 이와 같은 기법들은 모두 트리 구조를 기반으로 접근하고 그 트리의 노드를 효율적으로 탐색하기 위한 연구라 할 수 있다.

3차원 비트맵 인덱싱은 다른 인덱싱 기법과는 달리 인덱스를 트리 구조로 구성하지 않는다. 이것은 빠른 검색을 위해 기존의 인덱싱 기법이 트리의 노드를 단위로 탐색을 수행하는 것과 달리 XML 문서의 경로를 추출하고 이 경로를 단위로 탐색을 수행한다. 다음에 오는 2.2절에서는 3차원 비트맵 인덱싱에 대해 더 자세하게 소개한다.

2.2 3차원 비트맵 인덱싱과 그 연산들

3차원 비트맵 인덱싱인 BitCube는 XML 문서에서 효과적으로 정보를 추출하기 위한 기법으로 기존의 인덱싱 기법과는 달리 트리 구조의 인덱스 대신 Bit-wise 연산이 가능한 3차원 비트맵 인덱스를 사용한다. 하나의 인덱스는 특정 클러스터를 의미하고 이 클러스터는 유사한 구조의 문서들이 모여있는 집합이라 할 수 있다. 각각의 클러스터는 자신에게 포함된 문서와 그 문서에 포함된 경로 그리고 그 경로에 포함된 단어를 사용하여 3차원 비트맵 인덱스를 구성한다. BitCube는 유사한 구조의 문서들을 적절한 클러스터에 수집하기 위해 경로를 중심으로 클러스터링을 수행한다. 예를 들어 시스템에 어떤 문서가 삽입된다면 시스템은 삽입된 문서와 기존의 클러스터들간의 구조적인 유사도를 측정하고 가장

유사도가 높은 클러스터에 그 문서를 삽입하고 인덱스를 새롭게 갱신한다. 여기서 유사도는 특정 문서가 갖고 있는 경로들이 얼마나 고르게 클러스터에 분포되어 있는지를 나타내는 정도이다. 각각의 클러스터는 유사한 경로를 많이 포함하는 문서들의 집합이므로 클러스터내의 문서들이 갖고 있는 보편적인 경로들은 해당 클러스터를 대표하는 특징이라 볼 수 있다. 이 특징들의 집합을 클러스터의 중심이라 한다. 그러므로 삽입된 문서는 그것이 내포하고 있는 경로들과 클러스터의 중심과의 구성을 비교하고 가장 구성이 유사한 클러스터에 삽입된다.

BitCube는 구성된 3차원 인덱스에서 질의에 대한 결과를 추출하기 위해 기본적으로 세가지 연산을 지원하는데 ePath Slice, Word Slice 그리고 Document Projection 연산이 그것이다. 첫째로 *ePath Slice* 연산은 경로의 이름이 파라미터로 주어지면 그 경로를 포함하는 문서와 단어의 쌍으로 이루어진 리스트를 결과로 반환한다. 둘째로 Word Slice 연산은 단어가 파라미터로 주어지면 그 단어를 포함하는 문서와 경로의 쌍으로 이루어진 리스트를 결과로 반환하는 연산이다. 마지막으로 Document Projection 연산은 문서의 이름이 파라미터로 주어지면 그 문서에 포함되어 있는 경로와 단어의 쌍으로 이루어진 리스트를 결과로 반환하는 연산이다.

BitCube는 이 연산들을 사용하여 구성된 3차원 비트맵 인덱스에서 Bit-wise 연산을 통해 빠른 속도로 결과를 도출해낸다. 예를 들어 사용자가 특정한 경로를 포함하는 문서와 그 경로에 들어있는 단어들을 알고자 한다면 ePath Slice 연산을 사용한다. BitCube는 우선 사용자가 지정한 경로가 존재하는 클러스터들을 찾기 위해 메모리에 적재된 클러스터의 중심을 검색한다. 그리고 경로가 존재한다고 판단되는 클러스터를 메모리에 적재한 뒤에 결과를 추출한다. 이 때 메모리에 적재되는 특정 클러스터의 인덱스는 정사면체의 모양을 갖는 3차원 비트맵 인덱스이며 이것은 결국 해당 클러스터에 속하는 문서와 경로 그리고 단어의 ID가 의미있게 묶여있는 면들의 집합이라 볼 수 있다. 그리고 연산을 통해 추출되는 결과물도 이 정사면체 모양의 인덱스의 한 단면이라 할 수 있으며 시스템은 사용자에게 질의에 맞는 단면들의 집합을 제공하게 된다. 본 논문에서는 앞으로 정사면체인 3차원 비트맵 인덱스의 어떤 한 단면을 '평면'이라 한다.

BitCube는 기존의 XQEngine, XYZFind와 같은 시스템들과의 성능 평가에서 빠른 검색 속도를 통해 이미 뛰어난 성능을 입증하였다[2,3].

2.3 BitCube의 문제점

BitCube는 빠른 검색 성능으로 뛰어난 성능을 입증하였으나 문서의 증가에 따라 인덱스의 크기가 급격하게 늘어나는 단점을 지니고 있다. 이는 클러스터내의 문서의 개수 증가에 따라 메모리 사용량이 굉장히 높아지는 것을 의미하며, 연산 수행 속도를 저하시키는 원인이 된다. 다음은 이런 문제의 세가지 원인이다. 첫째, 클러스터와 클러스터내의 단어들 사이의 무관계성이다. BitCube는 클러스터에 유사한 구조의 문서를 수집하기 위해 경로에 중점을 두고 있으므로 경로의 내부에 들어있는 단어들 사이에는 어떤 관계성이 없다. 컴퓨터 전공 서적과 물리학 전공 서적은 모두 유사한 태그를 갖으므로 '책'이라는 하나의 클러스터에 포함될 수 있지만 컴퓨터 전공 서적에 관련된 단어와 물리학 전공 서적에 관련된 단어 사이에 어떤 유사성이 있을 수 없다. 다시말해 인덱스에 삽입되는 단어들은 이미 클러스터에 존재하는 단어이기 보다 새로운 단어일 확률이 매우 높다. 이것은 다음에 논의할 '평면에 의존적인 인덱스의 크기 증가'를 심화시키는 원인이 된다. 둘째, 평면에 의존적인 인덱스의 크기 증가이다. BitCube는 인덱스의 크기 증가가 평면에 의존적이다. BitCube는 어떤 문서나 경로 혹은 단어가 삽입될 때, 이것들이 이미 클러스터내에 존재한다면 인덱스를 확장할 필요가 없지만 그렇지 않은 경우 인덱스의 크기가 기존의 인덱스의 한 평면만큼 확장된다. 다시말해 새로운 단어가 하나 들어오고 그 단어가 해당 클러스터에 존재하지 않는다면 BitCube는 기존 인덱스에 존재하는 모든 문서의 개수와 경로의 개수를 곱한 만큼의 공간을 필요로 하게된다. 그리고 '클러스터와 클러스터내의 단어들 사이의 무관계성'에 의해 새롭게 들어오는 단어는 기존에 존재하지 않는 경우가 많다. 그러므로 BitCube의 인덱스의 크기 증가는 클러스터내의 문서의 개수가 증가할수록 급격하게 늘어나게 된다. 이것은 4.2절의 '메모리 사용량의 실험'에서 더 자세하게 논의한다. 셋째, 평면에 의존적인 연산이다. BitCube의 연산은 평면에 의존적이다. BitCube는 결과를 도출하기 위해 하나의 질의에 대해 하나의 평면 전체를 대상으로 Bit-wise 연산을 수행한다. Bit-wise 연산은 매우 빠른 연산이지만 인덱스의 크기가 급격하게 증가하면 전체 연산의 성능 저하를 초래한다. 이것은 4.3절의 '연산 수행 속도의 실험'에서 더 자세하게 논의한다.

결과적으로 BitCube는 적재된 문서의 개수가 증가할수록 심각한 성능 저하를 초래하게 된다. 특히 BitCube는 클러스터 단위로 인덱스를 메모리에 적재하는 시스템이므로 단일 클러스터의 인덱스 크기가 지나치게 커지는 경우 연산의 성능 저하는 급격하게 증가할 수 있으며 극단적인 경우 시스템이 연산을 전혀 수행할 수 없는 상황이 발생할 수도 있다.

3. 연결 리스트를 이용한 3차원 비트맵 인덱스

본 장에서는 BitCube의 문제점을 해결하기 위한 새로운 인덱싱인 xPlaneb에 관하여 기술한다. xPlaneb는 BitCube의 3차원 Boolean 배열 대신 연결 리스트를 사용하여 인덱스를 구성한다. 그리고 BitCube에서 사용되는 Bit-wise 연산대신 노드들을 연결하는 포인터 연산을 사용한다. 이로 인해 xPlaneb는 BitCube보다 메모리 사용과 연산 수행 속도면에서 더 뛰어난 성능을 갖게 된다.

3.1 xPlaneb의 설계 및 구조

연결 리스트를 이용한 3차원 비트맵 인덱스를 구현하기 위해서는 효율적인 노드의 설계가 선행되어야 한다. 그림 1은 전체 시스템의 구조를 나타낸다.

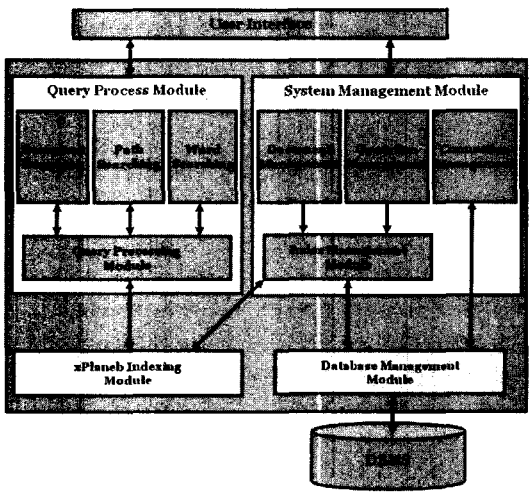

그림 1 전체 시스템의 구조

BitCube의 ePath Slice와 Document Project 연산을 수행하기 위해 리스트는 해당 노드의 단어 ID와 다음 단어ID를 가리키는 포인터가 있어야 한다. Word Slice 연산의 경우는 경로 ID와 다음 경로 ID를 가리키는 포인터가 필요하고 추가적으로 문서 ID를 알 수 있어야 한다. 결과적으로 노드는 문서와 경로 그리고 단어 ID에 대한 정보를 가지고 있어야 하며 다음 경로와 단어에 대한 포인터를 가지고 있어야 한다. 하나의 노드는 문서와 경로 그리고 단어 ID에 대한 정보를 담고 있어야 하므로 각각에 대해 4바이트의 공간이 필요하고 다음 경로와 단어에 대한 포인터를 가지고 있어야 하므로 각각에 대해 2바이트의 공간이 필요하게 된다. 결과적으로 하나의 노드는 16바이트(128비트)의 공간을 사용하게 된다. BitCube가 하나의 정보를 표현하기 위해 1비트의 공간을 사용하는 것에 비해 굉장히 많은 공간을 사용하는 것처럼 느껴질 수 있으나 2.3절에서 언급한 '클러스터와 클러스터내의 단어들 사이의 무관계성'과 '평면에 의존적인 인덱스의 크기 증가'와 같은 이유로 xPlaneb는 클러스터 내에 문서의 개수가 증가할수록 상대적으로 효율적인 메모리 사용량을 유지하게 된다. 표 1은 각각의 연산이 필요로 하는 필드에 대한 판별표이다.

표 1 연산이 필요로 하는 필드에 대한 판별표

Field / Operation	Doc. ID	Path ID	Word ID	Next Doc. Ptr.	Next Path Ptr.	Next Word Ptr
ePath Slice	X	X	O	X	X	O
Word Slice	O	O	X	X	O	X
Document Projection	X	X	O	X	X	O
	O	O	O	X	O	O

BitCube는 경로를 중심으로 클러스터링을 수행하므로 문서 ID와 경로 ID로 구성된 첫 평면은 매우 견고하다. 다시말해 평면을 구성하는 필드들에 True 값이 많은 자리를 차지하게 된다. 이 평면을 'Base Bitmap'이라 한다. 제안된 기법에서는 'Base Bitmap'의 각각의 필드와 대응되는 노드를 가르키는 포인터 배열이 존재하는데 이를 'Doc. and Path Ptr. Reference Map'이라 한다. 그리고 클러스터에 존재하는 단어들의 목록인 'Word Name List'의 각각의 필드와 대응되는 노드를 가르키는 포인터 배열이 존재하는데 이를 'Word Ptr. Reference List'라 한다. xPlaneb는 이 두 개의 '포인터 참조 인덱스'를 사용하여 연산 수행시에 인덱스에 접근하고 결과를 도출한다. 그림 2는 본 논문에서 제안된 연결 리스트를 이용한 3차원 비트맵 인덱스를 나타낸다.

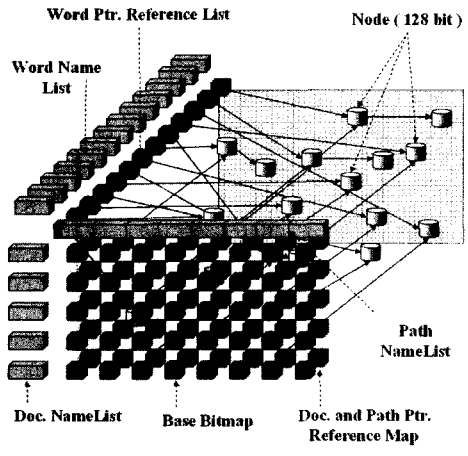

그림 2 연결 리스트를 이용한 3차원 비트맵 인덱스

3.2 xPlaneb의 연산

xPlaneb의 연산은 기존의 BitCube가 빠른 Bit-wise 연산을 수행하는 대신에 연결 리스트의 노드들이 가리키는 포인터를 순차적으로 따라가는 연산을 수행한다. Bit-wise 연산은 다른 연산보다 매우 빠른 연산이지만 2.3절에서 언급한 '평면에 의존적인 연산'으로 인해 클러스터내의 문서의 개수 증가에 따른 인덱스의 급격한 크기 증가로 성능 저하가 발생하게 된다. 반면에 xPlaneb의 연산은 Bit-wise 연산을 사용하지 않는 대신 연산 대상이 되는 필드를 대폭 감소시킴으로써 성능 저하를 효과적으로 막을 수 있다. 다시말해 xPlaneb의 연산은 결과가 되는 값만을 연결 리스트를 따라 순차적으로 읽어오게 되므로 연산의 수행 속도가 인덱스의 크기에 기인하지 않는다. 다음은 각각의 연산에 대한 정의와 알고리즘에 대해 논의한다.

3.2.1 Path Linking

이 연산은 경로의 이름이 파라미터로 주어지면 그 경로를 포함하는 문서와 단어의 쌍으로 이루어진 리스트를 결과로 반환하는 연산이다. 이것은 일단 전체 클러스터의 각각의 중심을 통해 이 경로가 해당클러스터에 포함되어 있는지를 검색한 뒤에 대상이 되는 클러스터를 탐색하여 결과 셋을 순차적으로 반환한다. 다음은 이 연산의 알고리즘을 표현한 것이다.

```
Method PathLinking()
for i = 1 ~ cNumber
LoadSquare(i)
if center[pID] = true then
for j = 1 ~ docSize
        temp = referenceMap[j, pID].nextWord
          while temp != null
            readData()
            temp = temp.nextWord
```

- *cNumber* :: 모든 클러스터의 개수
- *square[n]* :: 클러스터 ID가 n인 클러스터
- *pID* :: 특정한 경로의 ID
- *docSize* :: 클러스터내에 존재하는 모든 문서의 개수
- *referenceMap[n,m]* :: 문서와 경로의 ID로 구성된 리스트에서 특정 문서의ID(n)과 경로의 ID(m)에 대한 노드 주소
- *node.nextWord* :: 다음 경로를 포함하고 있는 노드

3.2.2 Word Linking

이 연산은 단어가 파라미터로 주어지면 그 단어를 포함하는 문서와 경로의 쌍으로 이루어진 리스트를 결과로 반환하는 연산이다. 이것은 일단 전체 클러스터의 단어 이름 리스트(Word Name List)를 통해 단어가 해당 클러스터에 포함되어 있는지를 검색한 뒤에 대상이 되는 클러스터를 탐색하여 결과 셋을 순차적으로 반환한다. 다음은 이 연산의 알고리즘을 표현한 것이다.

```
Method WordLinking()
for i = 1 ~ cNumber
LoadSquare(i)
temp = referenceList[wID].nextPath
   while  temp != null
      readData()
      temp = temp.nextPath
```

- *referenceList[n]* :: 단어의 ID로 구성된 리스트에서 특정 단어의 ID(n)에 대한 노드 주소
- *wID* :: 특정한 단어의 ID
- *node.nextPath* :: 다음 단어를 포함하고 있는 노드

3.2.3 Document Linking

이 연산은 문서의 이름이 파라미터로 주어지면 그 문서에 포함되어 있는 경로와 단어의 쌍으로 이루어진 리스트를 결과로 반환하는 연산이다. 이것은 해당문서가 들어있는 클러스터를 일단 탐색하고 해당하는 유일한 클러스터에서 문서의 위치를 검색한 뒤에 결과 셋을 순차적으로 반환한다. 다음은 이 연산의 알고리즘을 표현한 것이다.

```
Method DocLinking()
LoadSquare(getCID(dID))
for i = 1 ~ pathSize
temp = referenceMap[dID, i].nextWord
   while temp != null
readData()
      temp = temp.nextWord
```

- *getCID(n)* :: 특정 문서의 ID(n)가 들어있는 클러스터의 ID
- *dID* :: 특정한 문서의 ID
- *pathSize* :: 클러스터에 존재하는 모든 경로의 개수

4. 성능 평가

본 장에서는 기존의 3차원 비트맵 인덱싱인 BitCube와 본 논문에서 제안된 기법인 xPlaneb의 성능 평가를 수행한다. 4.1절에서는 실험 환경 및 평가 기준에 대해서 설명을 하고 4.2절과 4.3절에서는 두 기법간의 메모리 사용량과 연산 수행 속도를 차례로 실험 및 평가한다.

4.1 실험 환경 및 평가 기준

이 논문에서 제안된 기법은 MS Visual C#.NET으로 구현되었으며 운영체제는 MS Windows 2000 Server가 사용되었고 MS SQL Server 2000이 데이타베이스 관리 시스템으로 연동되었다. 실험은 256MB의 메인 메모리와 Intel Pentium III(1.0GHz) CPU가 장착된 컴퓨터에서 수행하였다.

기존의 BitCube와 관련된 논문에서는 하나의 클러스터가 아닌 전체의 클러스터를 대상으로 실험을 수행하였다. 그러나 본 논문에서는 하나의 클러스터를 대상으로 실험을 수행한다. 이는 전체 클러스터를 대상으로 수행하는 실험은 실험 대상이 되는 XML 문서들이 얼마나 유사한 구조를 갖고 빈번하게 등장하느냐에 따라 실험의 결과가 달라질 수 있기 때문이다.

실험에서 성능을 평가하게 되는 부분은 두 가지이다. 하나는 클러스터내의 문서의 개수와 단어 중복률에 따른 메모리 사용량의 비교이고 다른 하나는 클러스터내의 문서의 개수와 단어 중복률에 따른 연산 수행 속도의 비교이다. 여기서 단어의 중복률은 이미 클러스터에 삽입된 하나의 문서에서 등장했던 단어가 클러스터에 삽입될 다른 문서에서 다시 등장하게 될 확률을 의미한다. BitCube의 경우 한 번 등장한 단어가 다음에 다시 등장하게 될 경우에 메모리 사용량이 전혀 늘어나지 않으므로 이것은 중요한 실험의 조건이 된다. 본 논문에서는 단어의 중복률이 30%, 45%, 60%인 경우에 대하여 실험을 수행한다.

4.2 메모리 사용량의 실험

본 논문에서 제안된 기법인 xPlaneb나 BitCube와 같은 인덱싱 기법은 일반적으로 인덱스의 크기에 의해 연산 수행 속도에 직접적인 영향을 받게된다. xPlaneb와 같은 경우 인덱스의 크기 증가가 주로 단어의 개수 증가에 의존적인 반면에 BitCube와 같은 경우 인덱스의 크기 증가는 단순하게 단어의 개수 증가에 의존적인 것이 아니라 해당 클러스터의 3차원 비트맵 인덱스가 이루는 한 평면이 갖는 크기에 의존적이게 된다. 이것은 인덱스의 크기가 xPlaneb와 같은 경우 선형적으로 증가하지만 BitCube의 경우 지수적으로 증가하게 됨을 의미한다. 다시말해 xPlaneb의 경우 그 단어가 이미 클러스터내에 등장했던 단어인지 아닌지의 여부에 상관없이 순수하게 등장한 단어의 개수에 따라 크기가 증가하며 문서나 경로의 증가가 인덱스의 크기에 큰 영향을 주지 않는다. 그러나 BitCube의 경우 해당하는 단어가 이미 등장한 단어가 아닌 경우에 대하여 단어의 개수에 따라 크기가 증가하는데 단어 하나에 대해 필요한 인덱스의 크기는 '문서의 개수 × 경로의 개수' 비트가 된다. 이것은 BitCube가 단순히 증가하는 하나의 요소에 영향을 받는 것이 아니라 다른 요소들에 영향을 받게 되는 것을 의미한다. 또 BitCube는 문서나 경로와 같은 다른 요소들의 증가에서도 마찬가지로 각각의 요소와 다른 요소들의 영향을 받게 된다. 문서의 증가는 전체 인덱스에서 '경로의 개수 × 단어의 개수' 비트의 공간이 더 늘어나며 경로의 증가는 전체 인덱스에서 '문서의 개수 × 단어의 개수' 비트의 공간이 더 늘어난다.

표 2는 클러스터에 적재된 문서의 개수가 최대 500개까지 100개 단위로 증가할 때 BitCube와 xPlaneb가 단어 중복률 30%, 45%, 60%에서 사용하게 되는 메모리 사용량을 실험한 결과이다. 결과를 통해 알 수 있듯이 BitCube의 메모리 사용량은 지수적으로 증가하고 xPlaneb의 메모리 사용량은 선형적으로 증가하게 되므로 문서의 개수가 증가함에 따라 더욱 격차가 심해진다.

표 2 클러스터내의 문서의 개수와 단어 중복률에 따른 메모리 사용량 (단위 : Kbyte)

Duplication Rate \ Number of Documents		100	200	300	400	500
BitCube	60%	508.2	2016.4	4524.6	8032.8	12541.0
	45%	698.7	2772.4	6221.1	11044.8	17243.5
	30%	889.2	3528.4	7917.6	14056.8	21946.0
xPlaneb	60%	178.2	356.3	534.6	712.8	891.0
	45%	184.2	368.4	552.6	736.8	921.0
	30%	190.2	380.4	570.7	760.9	951.1

4.3 연산 수행 속도의 실험

BitCube와 같은 인덱싱 기법은 연산을 수행하기 위해서 굉장히 속도가 빠른 Bit-wise 연산을 사용하지만 실제 결과를 추출하고 반환하기 위해서는 3차원 비트맵 인덱스의 한 평면 전체에 대하여 이 연산을 수행해야만 한다. 그러므로 인덱스의 크기가 연산의 수행 속도에 직접적인 영향을 미치게 된다. 우리는 이미 4.2절의 메모리 사용량에 대한 실험을 통해 BitCube의 인덱스가 급격하게 증가하는 것을 확인하였다. 반면에 xPlaneb는 인덱스의 크기가 아닌 반환해야 하는 결과 값의 개수가 수행 속도를 결정하는 주된 요인이 된다. 이것은 xPlaneb가 Bit-wise 연산과 같은 빠른 연산을 수행하는 대신에 연결 리스트를 이용하여 반환해야 하는 결과만을 추출하기 때문이다. 물론 연결 리스트를 구현하기 위해서는 하나의 노드가 128비트의 공간을 사용해야 하므로 문서가 소량일 경우에는 BitCube보다 더 많은 공간을 사용하게 된다. 그러나 4.2절의 실험을 통해 문서의 개수가 증가함에 따라 메모리 사용량이 BitCube에 비해 매우 적게 유지됨을 확인했다.

다음은 두 기법의 각각의 기본적인 연산인 ePath Slice와 Path Linking, Word Slice와 Word Linking 그리고 Document Projection과 Document Linking의 문서의 개수 증가에 따른 연산 수행 속도의 변화를 측정한 결과이다.

그림 3은 클러스터내의 적재된 문서의 개수가 증가함

에따라 BitCube의 ePath Slice 연산과 xPlaneb의 Path Linking 연산의 수행 속도의 변화를 보여준다. ePath Slice 연산의 경우 단어 중복률에 따라 정도는 틀리지만 1개 이상의 문서가 클러스터에 적재되었을 때부터 서서히 성능 저하가 발생하며 그 성능 저하의 정도는 급격하게 높아진다. 이 연산이 다른 연산보다 성능 저하가 매우 높은 이유는 경로를 검색하는 연산에 영향을 미치는 평면이 문서의 개수와 단어의 개수에 기인하기 때문이다.

그림 4는 클러스터내의 적재된 문서의 개수가 증가함에따라 BitCube의 Word Slice 연산과 xPlaneb의 Word Linking 연산의 수행 속도의 변화를 보여준다. Word Slice 연산의 경우 대략 400개 이상의 문서가 클러스터에 적재되었을 때부터 성능의 저하가 나타난다. 다른 연산에 비해 이 연산의 성능 저하가 낮은 이유는 단어를 검색하는 연산에 영향을 미치는 평면이 경로의 개수와 문서의 개수에 기인하기 때문이다.

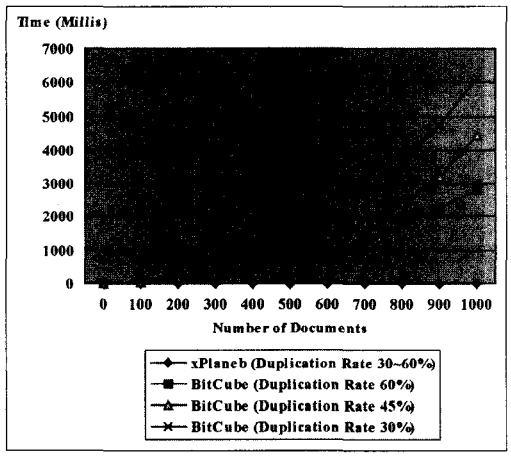

그림 3 클러스터내 문서량의 증가에 따른 경로 검색 속도 비교

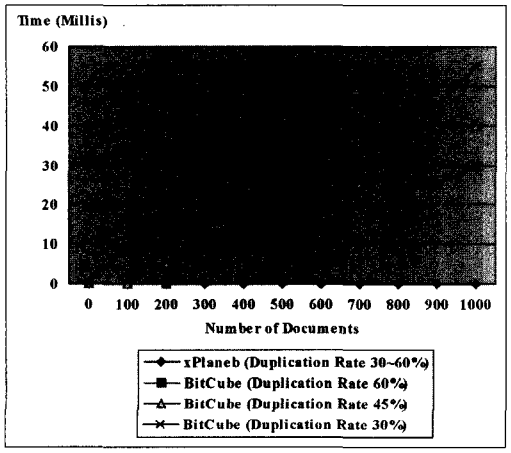

그림 4 클러스터내 문서량의 증가에 따른 단어 검색 속도 비교

그림 5는 클러스터내의 적재된 문서의 개수가 증가함에따라 BitCube의 Document Projection 연산과 xPlaneb의 Document Linking 연산의 수행 속도의 변화를 보여준다. Document Projection 연산의 경우 단어 중복률에 따라 결과의 성능 저하의 정도에 차이가 발생하는데 단어 중복률이 30%인 경우는 1개 이상의 문서가 클러스터에 적재되었을 때부터 그리고 단어 중복률이 45%~60%인 경우 100개 이상의 문서가 클러스터에 적재되었을 때부터 성능의 저하가 나타난다. 이 연산이 단어를 검색하는 연산보다 성능 저하가 높은 이유는 문서를 검색하는 연산에 영향을 미치는 평면이 경로의 개수와 단어의 개수에 기인하기 때문이다.

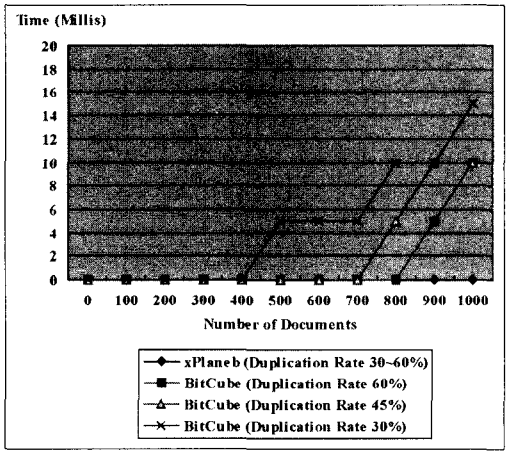

그림 5 클러스터내 문서량의 증가에 따른 문서 검색 속도 비교

결과에서 볼 수 있듯이 xPlaneb는 500개 이하의 문서에서 전혀 연산의 성능 저하가 발생하지 않지만 BitCube는 연산들마다 정도의 차이는 있으나 성능저하가 나타남을 알 수 있다. BitCube 연산들의 성능 저하에 정도의 차이가 있는 것은 위에서 언급한 대로 각각의 연산에 영향을 미치는 평면이 다르기 때문이다. 예를 들어 경로를 검색하는 연산의 경우 경로의 개수와는 상관없이 문서와 단어의 개수에 영향을 받게 되는데 일반적으로 문서와 단어의 개수는 경로의 개수보다 많으므로 이 연산의 성능 저하가 두드러지게 커지는 것이다. 경로의 개수가 단어나 문서의 개수보다 적은 것은 BitCube가 유사한 구조의 문서를 하나의 클러스터로 수집하는 과정에서 유사도를 측정하는 것이 경로를 중심

으로 수행되기 때문이다. 즉, 클러스터를 구성하는 경로들은 해당 클러스터들의 내부에 들어있는 문서들의 대표적인 특징과 같은 것이며 클러스터내의 문서들에 각각의 경로가 포함되어 있는 확률이 일정수준 아래로 내려가면 해당 경로는 클러스터에서 배제된다. 그러므로 클러스터내의 문서의 개수가 증가할수록 경로의 개수는 늘어나지 않거나 오히려 줄어드는 경우도 발생하게 된다. 그래서 상대적으로 경로의 개수는 문서와 단어의 개수보다 작아지는 것이다. 반대로 단어의 개수는 문서와 경로의 개수보다 상대적으로 매우 많다. 이것은 BitCube가 단어가 들어있는 경로만을 의미있는 자료로 인식하기 때문이다. 문서가 있다는 것은 단어를 포함하는 경로가 있다는 것이고 어떤 경로에 포함되어 있는 단어의 개수는 최소 하나 이상이므로 당연히 단어의 개수는 문서와 경로의 개수보다 많아진다. 일반적으로 하나의 경로에 들어있는 단어의 개수가 하나 밖에 없는 경우는 흔하지 않다. 2.3절의 'BitCube의 문제점'에서 논의한 것처럼 단어의 개수는 '클러스터와 클러스터내의 단어들 간의 무관계성'에 의해 굉장히 크게 증가하는 경우가 많다. 그러므로 BitCube의 연산은 경로의 개수에 영향을 받게되면 상대적으로 성능 저하가 미비해지고 단어의 개수에 영향을 받게되면 상대적으로 성능 저하가 두드러진다.

5. 결 론

기존의 비트맵 인덱싱인 BitCube는 하나의 클러스터에 대량의 문서가 적재되었을 때, 메모리 사용량이 급격하게 증가하는 문제점을 갖고 있었으며 이로 인하여 연산의 수행 속도도 크게 저하되는 문제점을 갖고 있었다. 본 논문에서는 이에 대한 해결책으로 연결 리스트를 이용한 새로운 3차원 비트맵 인덱싱인 xPlaneb를 설계 및 구현하고 BitCube와의 성능을 비교 및 평가하였다. 실험을 한 결과 하나의 클러스터에 적재된 문서의 개수가 약 25개 이상인 경우에 제안된 기법이 메모리 사용과 연산 수행 속도면에서 항상 BitCube보다 뛰어난 성능을 보이는 것을 볼 수 있었다. 이와 같은 성능 향상은 BitCube가 클러스터 단위로 인덱스를 메모리에 적재하는 시스템임을 고려할 때 단일 클러스터의 인덱스 크기가 지나치게 커지는 경우 메모리의 적재 한계에 의해 시스템이 연산을 전혀 수행할 수 없는 문제에 대해서도 부분적인 해결책을 제시할 수 있다. 그러나 클러스터내의 문서의 개수가 약 25개 이하인 경우에 연산 수행 속도면에서는 큰 차이가 없으나 메모리 사용량면에서 제안된 기법이 기존의 기법보다 약간 불리하다. 이는 BitCube는 인덱스의 크기 증가가 전체 인덱스의 특정 평면의 크기에 의존적이지만 인덱스를 구성하는 하나의 필드가 1비트의 매우 작은 크기로 구성되어 있으므로 소량의 문서에 대해 작은 메모리 사용량을 유지할 수 있는 것에 기인한다.

향후 연구에서는 문서 중심적인 XML 문서와 자료 중심적인 XML 문서가 산재되어 있는 환경에서 시스템의 판단에 따라 각각의 클러스터에 더 유리한 인덱스를 선택하는 하이브리드한 인덱싱 기법에 대한 연구를 수행할 것이다.

참 고 문 헌

[1] W3C, "Extensible Markup Language(XML) Version 1.0 (Second Edition)," http://www.w3c.org/TR/REC-xml, October 2000.

[2] J. Yoon, V. Raghavan, and V. Chakilam, "BitCube: Clustering and Statistical Analysis for XML Documents," 13th International Conference on Scientific and Statistical Database Management, Virginia, July 2001.

[3] J. Yoon, V. Raghavan, V. Chakilam, and L. Kerschberg, "BitCube: A Three-Dimentional Bitmap Indexing for XML Documents," Journal of Intelligent Information System, Vol.17, pp.241-254, 2001.

[4] C. Chan and Y. Ioannidis, "Bitmap Index Design and Evaluation," Proceedings of ACM SIGMOD Conference, Seattle, pp.355-366, June 1998.

[5] S. Banerjee, Oracle XML DB, An Oracle Technical White Parer, January 2003.

[6] L. Ennser, C. Delporte, M. Oba, and K. M. Sunil, Integrating XML with DB2 XML Extender and DB2 Text Extender, IBM Redbook, December 2000.

[7] S. Howlett and D. Jennings, "SQL Server 2000 and XML : Developing XML-Enabled Data Solutions for the Web," MSDN Magazine, Vol.17, No.1, January 2002.

[8] Sybase Corporation, XML Management Package for Sybase Adaptive ServerEnterprise 12.5.1, A Sybase Technical White Paper, September 2003.

[9] D. Egnor and R. Lord, "XYZFind: Structured Searching in Context with XML," ACM SIGIR Workshop, Athens, Greece, July 2000.

[10] XQEngine, http://www.fatdog.com.

[11] M. Rousset, "Semantic Data Integration in Xyleme," Presentation at INRIA, September 1999.

[12] J. McHugh, J. Wisdom, S. Abiteboul, Q. Luo, and A. Rajaraman, "Indexing Semistructured Data," Stanford University Technical Report, February 1998.

[13] N. Fuhr and K. Grossjohann, "XIRQL: An Extension of XQL for Information Retrieval," ACM SIGIR Workshop, Athens, Greece, July 2000.

[14] B. Cooper, N. Sample, M. Franklin, and M. Shadmon, "A Fast Index for Semistructured Data," Procedings of the 27th VLDB Conference, Roma, Italy, 2001.
[15] Y. Papakonstantinou and V. Vianu, "DTD Inference for Views of XML Data," Proceedings of ACM SIGACT-SIGMOD-SIGART Symposium on PODS, 2000.

이 재 민

2003년 가톨릭대학교 컴퓨터공학과(공학사). 2003년~현재 가톨릭대학교 컴퓨터공학과 석사과정. 관심분야는 XML 데이타베이스, 정보검색, 정보보안

황 병 연

1986년 서울대학교 컴퓨터공학과(공학사). 1989년 한국과학기술원 전산학과(공학석사). 1994년 한국과학기술원 전산학과(공학박사). 1999년~2000년 Univ. of Minnesota Visiting Scholar. 1994년~현재 가톨릭대학교 컴퓨터정보공학부 교수. 2002년~현재 가톨릭대학교 정보통신원장. 관심분야는 지리정보시스템, XML 데이타베이스, 데이타 마이닝, 정보검색