동적 환경에 적합한 SGML인덱스 관리자의 설계 및 구현

한 성 근[†] · 손 정 한[†] · 장 재 우[††] · 김 현 기[†††] · 강 현 규[††††]

요 약

SGML문서는 정보 표현의 기본 단위인 엘리먼트로 구성되어져 있기 때문에 SGML 정보 검색은 기존의 정보 검색에서의 문서 단위 검색뿐만 아니라 엘리먼트 단위 검색이 이루어져야 한다. 또한, SGML 인덱스 구조는 동적 환경을 위해 문서의 부분 삭제와 부분 삽입을 지원해야 한다. 이를 위해 본 연구에서는 동적 환경하에서 구조 질의에 적합한 SGML인덱스 구조를 제안한다. 그리고, 제안된 인덱스 구조에 근거하여 내용 및 구조-기반 검색을 효율적으로 지원하는 인덱스 관리자를 설계하고, O2시스템을 기반으로 SGML정보 검색 인덱스 관리자를 구현하며, 기존 인덱스 관리자와 성능 비교를 수행한다. 검색 성능 비교 결과, 본 연구에서 제안한 방법이 기존의 K-ary 완전 트리를 사용한 방법보다 더 우수함을 나타낸다.

Design and Implementation of a SGML Index Manager for Dynamic Environment

Sung-Geun Han[†] · Jeong-Han Son[†] · Jae-Woo Chang[††] · Hyun-Ki Kim[†††] · Hyun-Kyu Kang[††††]

ABSTRACT

Since a SGML document is composed of elements, the primitive unit of information, SGML information retrieval should support retrieval on element as well as document. In addition, SGML index organization should support the partial insertion and deletion of document for the dynamic environment. For this, we propose a SGML index organization suited to structured-based retrieval for dynamic environment. Based on the proposed index organization, we design a SGML index manager to support content-based and structure-based retrieval efficiently. We implement the SGML index manager based on O2 storage system and compare the performance of our SGML index manager with the conventional SGML index manager. According to the performance comparison, it is shown that the proposed index structure achieves better retrieval performance than the conventional K-ary complete tree.

1. 서 론

논문이나 보고서와 같이 내부적으로 복잡한 구조를 지니는 문서를 효과적으로 처리하기 위해서는 문서의 논리적 구조를 기술하기 위한 표준이 필요하다. 국제 표준화 기구(International Organization for Standardization)는 이러한 표준 언어로서 SGML(Standard Generalized Markup Language)을 제정하였다. 따라서 SGML로 작성된 문서는 문서정보를 체계적으로 조직, 생성하고, 전송하기 위한 문서로서, 문서의 적절한 구조를 유지할 수 있다. 이의 유용성 때문에 미국 국방성

※ 본 연구는 한국전자통신연구원에서 수행한 우리말 정보 처리 기술 개발 사업의 위탁과제로 수행되었음.
† 준 회 원 : 전북대학교 대학원 컴퓨터공학과
†† 비 회 원 : 전북대학교 컴퓨터공학과 교수
††† 정 회 원 : 한국전자통신연구원 연구원
†††† 종신회원 : 한국전자통신연구원 선임연구원, 문서정보 연구팀장
논문접수 : 1999년 2월 18일, 심사완료 : 1999년 9월 20일

의 CALS(Commerce At Light Speed)프로젝트는 국가기관,기업과 산업체등에서 시스템의 환경에 제약 없이 서로의 정보를 공유할 수 있는 SGML을 사용하고 있다[1, 2]. 이에 따라 기존의 전자 문서 들을 SGML로 변환하는 작업이 활발히 진행되고 있으며, SGML문서의 논리적 구조를 반영하는 요구도 크게 증가하고 있다.

한편 기존의 정보검색 시스템은 문서의 내용을 분석하여 색인어를 추출하고 사용자의 질의가 주어졌을 때 질의에 사용된 단어와 색인어의 유사성을 계산하여 관련 문서를 제공한다. 이러한 전문(full-text) 검색 시스템은 문서의 구조적인 정보를 적절하게 활용하지 못하는 단점을 지니고 있다. SGML문서를 위한 정보 검색 시스템을 구축하는데 있어서 중요한 것은 첫째, SGML 문서의 추상화된 정보 표현의 기본단위는 엘리먼트이므로 기존의 정보 검색에서의 문서 단위와는 달리 엘리먼트 단위의 검색이 이루어져야 한다는 점이다[3]. 둘째, 문서의 부분 삭제와 부분 삽입이 가능한 동적인 환경에 적합한 인덱스 구조(Index Organization)가 설계되어야 한다는 점이다. SGML문서의 정보검색을 위한 다수의 연구들이 수행되었으나, 기존 방법들은 문서의 트리 구조 및 내용을 직접적으로 데이터베이스에 매핑시키기 때문에, 문서 구조에 대한 질의를 효과적으로 처리하지 못한다. 따라서, 본 연구에서는 효과적인 구조 검색 질의를 지원하는 SGML 정보 검색 인덱스 구조를 제안한다. 제안된 인덱스 관리자를 기반으로 SGML정보를 저장 및 삭제하고, 내용 검색과 구조 검색을 지원하는 인덱스 관리자를 설계한다. 설계한 인덱스 관리자를 O2시스템에 기반하여 구현하며, 성능평가를 위해 기존 인덱스 관리자와 문서 삽입시간, 색인 정보 검색 시간, 문서 삭제 시간과 부가 저장 공간의 측면에서 성능비교를 수행한다.

본 연구의 구성은 다음과 같다. 제2장에서는 SGML 인덱스 구조에 관한 기존 연구를 분석한다. 제3장에서는 이를 바탕으로 동적 환경에 적합한 구조 인덱스를 설계하고, 제4장에서는 효율적인 SGML 문서를 위한 정보 검색 인덱스 관리자를 설계한다. 제5장에서는 설계한 인덱스 관리자를 구현하고 성능 평가를 수행한다. 제6장에서는 결론 및 향후 연구 방향을 제시한다.

2. SGML인덱스 구조의 기존 연구

SGML문서를 구성하는 기본 단위는 엘리먼트이므로, SGML 정보 검색 시스템은 기존의 정보검색 시스템에서 사용한 문서 단위를 위주로 한 검색 이외에 임의 깊이에서 나타나는 엘리먼트 단위의 검색을 지원해야 한다. 또한, 엘리먼트 사이의 논리적인 포함관계 및 엘리먼트의 특성값에 대한 질의도 지원해야 한다. SGML로 표현된 문서의 관리 및 구조 검색에 대한 기존연구는 크게 엘리먼트를 상용 데이터베이스 시스템에 매핑하는 방법과 엘리먼트-기반 정보검색 시스템으로 분류할 수 있다[4, 5, 6, 7, 8, 9, 10].

2.1 엘리먼트를 상용데이타베이스 시스템에 매핑

SGML문서를 구성하는데 있어 규칙을 제공하는 DTD를 상용 데이타베이스 시스템의 스키마로 매핑하는 방법[5, 10]으로서 대표적인 두 가지 연구가 있다. 첫째, 엘리먼트를 RDBMS로 매핑하는 것으로서, DTD안에 있는 엘리먼트에 관한 선언들을 관계형 스키마를 사용하여 관계형 모델의 테이블로 변환하는 연구이다. 둘째, 엘리먼트를 OODBMS로 매핑하는 것으로서 DTD안에 선언된 엘리먼트들을 객체 지향 모델의 클래스를 사용하여 오브젝트로 변환하는 것이다. 이 방법은 기존의 상용 데이타베이스 시스템이 지니고 있는 고수준의 질의와 회복, 일치성 유지 등을 모두 사용할 수 있는 장점이 있다.

그러나, 다음과 같은 단점을 지닌다. 첫째, DTD를 데이타베이스의 스키마로 변환하는 복잡한 과정이 필요하다. 예를 들면 문자열 대체를 위한 약어 선언, 특수문자, 강조문자, 기호 등의 표현 및 외부 파일 참조 기능을 제공하는 엔터티와 같은 특징을 데이타베이스의 스키마로 표현하는 것은 어렵기 때문에 스키마로 변환하는 도중에 손실이 발생한다. 둘째, 같은 DTD에 의해 생성된 문서라도 문서의 구조가 각각 다르기 때문에 많은 테이블이 요구되고 테이블과 객체 인스턴스내에 많은 NULL값이 발생하게 된다. 그리고 상위 엘리먼트에 접근하기 위해서 많은 join연산들이 필요하게 된다. 마지막으로 새로운 DTD의 문서들을 데이타베이스에 저장할 때마다 기존의 데이타베이스 스키마를 변경시켜 주어야 한다는 단점이 있다.

2.2 엘리먼트-기반 정보검색 시스템

기존의 문서 기반 정보검색 시스템은 문서에 대한 전문 검색만을 지원하기 때문에 문서의 논리적 구조에 기반한 검색에는 적절치 못하다. 그러나 엘리먼트-기

반 정보검색 시스템은 DTD를 이용한 구문 분석 후, 문서의 구조를 지시하는 마크업(Mark-up)에 대해 인덱스를 구성함으로써 구조질의를 가능하게 한다. 따라서, DTD를 특정 데이터베이스의 스키마로 변환하는 과정이 존재하지 않는다.

엘리먼트-기반 정보 검색 시스템으로서 첫째, 호주 RMIT에서 제시한 SCL(Simple Concordance List)[7] 은GCL(Generalized Concordance Lists)구조를 확장한 구조이며, SGML문서내의 텍스트와 마크업에 대해 색인 번호를 부여한 후, 이들의 텍스트 간격과 포함관계를 사용하는 방법이다. Stoplist단어들을 제외한 텍스트 어휘들은 text index에 색인 번호로 저장되고, 마크업은 start position과 end position의 쌍으로 markup index에 저장된다. 그리고, 실제 텍스트를 저장된 물리적인 위치로 사상시키기 위해 매핑 테이블을 사용한다. 그러나, SCL구조는 색인 엘리먼트의 조상, 형제 엘리먼트를 알 수 없으며 포함하고 있는 엘리먼트가 몇 번째 자손 엘리먼트에 해당하는지 알 수 없다.

둘째, SERI가 제안한 K-ary 완전트리구조[9]는 문서구조를 K-ary 완전트리구조로 매핑하는 방법이다. 이 방법에서 논리적 구조 관계인 부모 노드와 자식 노드의 관계는 아래와 같은 연산식으로 표현할 수 있다.

i번째 노드의 부모 노드를 구하는 식

$$Parent(i) = (i-2) / k + 1$$

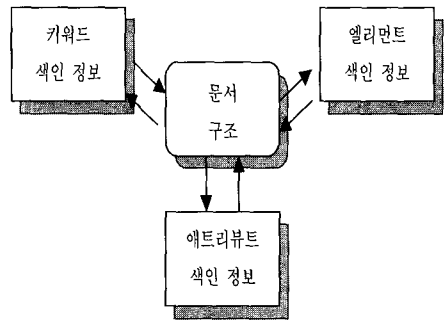
i번째 노드의 j번째 자식 노드를 구하는 식

$$Child(i,j) = k (i-1) + j + 1$$

K-ary완전 트리 방식은 계산에 의해서 논리적 포함관계인 엘리먼트를 빠르게 찾을 수 있는 장점이 있으나, 노드의 깊이가 깊어질수록 노드변화가 지수에 비례하여 커지고 사용하지 않는 노드번호가 많아지므로 데이터량이 커지는 단점이 있다. 또한 문서 구조의 부분 삽입과 부분 삭제가 발생했을 때, 노드번호를 새로이 다시 계산해야 하는 단점이 있다.

3. 동적 환경에 적합한 구조 인덱스 설계

엘리먼트 기반 정보 검색 시스템은 엘리먼트 이름, 키워드, 애트리뷰트의 문서정보를 논리적 정보 단위인 엘리먼트 단위로 색인을 한다. 엘리먼트들은 구조를 형성하기 때문에 키워드 색인 정보, 엘리먼트 색인 정보와 애트리뷰트 색인 정보는 (그림 3.1)과 같이 문서 구조와 상호 연관성을 형성한다. 이로 인해 엘리먼트의 구조 정보 변경은 색인 정보의 변경을 야기시킨다.



(그림 3.1) SGML 정보들의 상호 관계

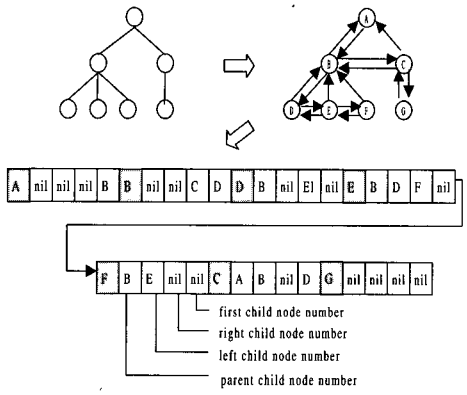
따라서 문서에 대하여 엘리먼트의 부분 삽입 및 부분 삭제가 발생해도 색인 정보의 변경 없이, SGML엘리먼트와 데이터의 위치, 특성값 등을 잘 표현할 수 있는 인덱스 구조가 설계되어야만 한다. 그러나 SCL 방법은 문서내의 마크업(Mark-up)에 대하여 색인 번호를 부여하므로 중간에 엘리먼트가 삽입하게 되면 삽입부 이후부터 엘리먼트에 대한 색인 번호가 변하고 그에 따라 텍스트에 대한 색인 번호의 변경이 발생하게 된다. 또한, K-ary완전트리방법도 중간에 엘리먼트가 삽입되면, 그 이후의 엘리먼트에 대한 K-ary완전 노드 번호가 변경되므로 부분삽입과 부분삭제를 수행할 수가 없다. 이에 본 연구에서는 부분 삽입과 부분 삭제를 효과적으로 수행할 수 있는 문서 단위 구문 트리구조(Document Unit Parse Tree)의 엘리먼트 단위 구문 트리구조(Element Unit Parse Tree)인덱스 방법을 제안한다.

3.1문서 단위 구문 트리 구조

문서 단위 구문 트리 구조는 SGML 파서(parser)를 통해 나온 문서구조 정보를 하나의 레코드에 반영하는 방식이다. 각 엘리먼트의 노드는 부모, 오른쪽 첫번째 형제, 왼쪽 첫번째 형제 및 첫번째 자식 노드를 가리키는 포인터로 구성한 후, 실제 레코드에 저장시 포인터에 해당하는 위치의 노드 번호 값을 저장하게 된다. (그림 3.2)는 SGML Parse 트리를 문서 단위 구문 트리 구조로 변환하는 과정을 나타낸다.

문서 단위 구문 트리 구조는 구조 정보를 부분 삽입, 부분 삭제하는데 있어서 구조에 변경을 가해주는

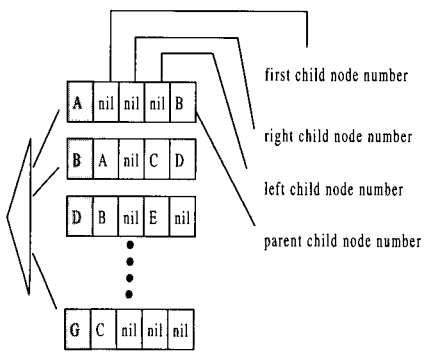
특정 레코드에만 변경이 가해지고, 다른 엘리먼트들에 대해서는 영향을 주지 않는 장점을 가진다. 그러나 문서의 구조 정보 전체를 하나의 레코드로 유지하기 때문에 커다란 정보를 유지할 수 있는 긴 자료타입(Long Data type)레코드[11]로 문서의 구조 정보를 관리해야 하는 단점을 지닌다. 그리고 이 방식은 문서의 특정 구조 정보만을 접근하고자 할 때 전체 구조 정보를 읽어야 하는 단점을 가진다.



(그림 3.2) 문서 단위 구문 트리 구조

3.2 엘리먼트 단위 구문 트리 구조

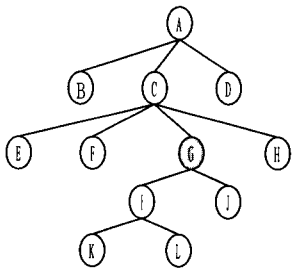
엘리먼트 단위 구문 트리 구조는 SGML 파서를 통해 나온 구문 트리 정보를 문서단위로 하나의 레코드에 저장하는 것이 아니라, 하나의 문서의 구조 정보를 여러 개의 엘리먼트 단위 레코드로 저장하는 방식이다. 문서 구조를 이루는 각각의 엘리먼트를 빠르게 접근하기 위해 B+트리를 사용한다. (그림 3.3)은 (그림 3.2)의 SGML parse 트리를 엘리먼트 단위 구문 트리 구조로 변환하는 과정을 나타낸다. 엘리먼트 단위 구문 트리 구조도 부분 삽입 및 부분 삭제에 있어서 문서 단위 구문 트리의 구조와 마찬가지로 용이하게 처리되며, 문서의 특정 부분 구조를 접근할 때 B+트리를 통해 빠르게 접근할 수 있는 장점이 있다. 하지만 엘리먼트에 대한 B+트리를 유지해야 하기 때문에 부가 저장 공간이 문서 단위 구문 트리보다 크다는 단점이 있다.



(그림 3.3) 엘리먼트 단위 구문 트리

3.3 제안하는 구문 트리의 비교

제안하는 두 가지 구문 트리의 비교를 위해서, 평가에 사용될 구조 질의를 3가지 형태로 분류한다. 첫째, 문서의 현재 엘리먼트를 중심으로 직접 접근할 수 있는 질의 그룹. 둘째, 집합(set)의 형태로 지원되는 질의 그룹. 마지막으로, 순서(order)의 형태로 지원되는 질의 그룹으로 분류한다. 이렇게 3가지 형태로 분류하는 이유는 각각의 형태가 현저히 다른 구조 질의의 특성을 보이기 때문이다. <표 3.1>은 (그림 3.4)에서 현재 엘리먼트가 G일 때의 3가지 질의 그룹을 나타내고 있다.



(그림 3.4) 임의 문서 구조

〈표 3.1〉 구조 질의 분류

직접 질의 : Query on direct access for relationship	Parent 1st Child 1st LeftSibling 1st RightSibling	C I F H
집합 질의 : Query on set with relationship	Ancestor Descendant LeftSibling RightSibling AllSibling AllChildren	C, A I, J, K, L F, E H E, F, H I, J
순서 질의 : Query on relationship with order	2nd Ancestor 2nd Descendant 2nd Child 2nd leftSibling 2nd RightShibling	A K, L J E Empty

문서 단위 구문 트리 구조는 SGML 구문 파서를 통해 구문 트리 구조정보를 문서단위로 저장한다. 따라서 엘리먼트들간의 포함관계를 모두 알 수 있는 전역적 접근 방법이다. 반면, 엘리먼트 단위의 구문 트리 구조는 엘리먼트 단위로 구문 트리 구조 정보를 저장하기 때문에 하나의 특정 엘리먼트에 이웃한 엘리먼트들간의 포함관계만 접근할 수 있는 지역적 접근 방식이다. 문서 단위 구문 트리는 구조 정보를 레코드 하나로 관리하기 때문에 문서의 구조 중에 일부분만 탐색하고자 할 때도 구조 전체를 탐색해야 한다. 그러나 엘리먼트 단위 구문 트리는 엘리먼트 단위로 레코드를 구성하고 레코드에 대해 B+트리를 구성하기 때문에, 검색하고자 하는 구조 중 특정 부분만 탐색할 수 있다.

두 방법의 구조 탐색의 효율성을 측정하기 위해서, 분석 모델(analytic model)을 통해 I/O페이지 접근 횟수를 계산한다. 이때 한 페이지의 크기는 4096byte로 가정하며 하나의 엘리먼트는 자기 자신의 ID와 Parent ID, 1st LeftSibling, 1st RightSibling, 1st Child Id정보를 가지기 때문에 약 50byte의 크기가 필요하다고 가정한다. 먼저 구조 검색 효율성 비교를 위한 대상 문서는 <표 3.2>와 같다.

〈표 3.2〉 문서의 특성

	논문 문서[A]	Reuter 문서[B]
문서당 평균 엘리먼트 수	683,96개	24,080,61개
엘리먼트의 평균 높이	10.13	44.98
엘리먼트의 평균 차수	7.11	5.59

아울러 구조 인덱스를 구성할 때, 구조 검색은 키(Key)가 있는 B+트리 화일과 레코드가 있는 포스팅 화일 접근으로 수행되어진다. 따라서 각각에 대하여 화일 접근 횟수를 계산한다. 먼저 B+트리 화일 접근 횟수를 계산해 본면, 키의 크기가 10byte이고 포인터가 4byte일 때 B+트리 화일을 한번 접근으로 약 292개의 키를 얻어올 수 있다. 이 때 문서 단위 구문 트리는 문서 단위로 키를 구성하고 엘리먼트 단위 구문 트리는 엘리먼트 단위로 키를 구성한다. 이를 기준으로 문서 단위 구문 트리와 엘리먼트 단위 구문 트리의 B+트리 화일 페이지 I/O횟수를 계산하면 <표 3.3>과 같고, 이를 통해 논문문서[A]와Reuter 문서[B]에 대하여 페이지 I/O횟수를 구한 결과는 <표 3.4>와 같다.

〈표 3.3〉 B+트리 파일 페이지 I/O

	문서 단위	엘리먼트 단위	
문서 종류	논문/Reuter	논문	Reuter
KEY 수	1,000	68,396	24,080,610
B+트리화일 접근 수	2	2	3

〈표 3.4〉 구조 검색의 수학적 비교

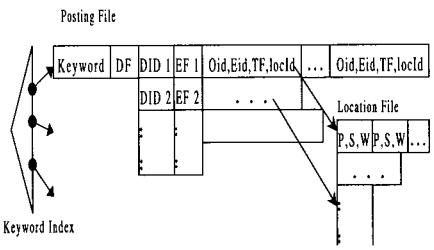
비교 \ 단위			문서 단위 구문 트리	엘리먼트 단위 구문 트리
구조 탐색 범위			전역적	지역적
구조 질의	직접 질의	[A]	8.3	3.5
		[B]	293.9	3.5
	집합 질의	[A]	8.3	30.17
		[B]	293.9	88.50
	순서 질의	[A]	8.3	15.90
		[B]	293.9	44.25

4. SGML 정보 검색 인덱스 관리자의 설계

SGML정보 검색 인덱스 관리자는 내용 인덱스, 구조 인덱스, 엘리먼트 인덱스와 애트리뷰트 인덱스로 구성되며, SGML의 기본 표현 단위인 엘리먼트 단위로 색인하여 인덱스를 구성한다.

4.1내용 인덱스

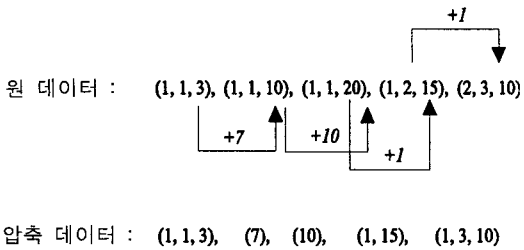
내용 인덱스는 SGML 문서의 데이터 토큰(PCDATA, RCDATA, CDATA) 엘리먼트로부터 추출된 색인어들로 구성된 단어 색인 화일(Keyword Index File), 색인어가 출현한 문서와 엘리먼트의 정보를 나타내는 포스팅 화일(Posting File), 엘리먼트에서 색인어의 출현 위치에 대한 정보를 포함하는 위치 정보 화일(Location File)로 구성된다[12]. (그림 4.1)은 내용 인덱스의 구조를 나타내며, 기존의 정보 검색 인덱스에 엘리먼트에 대한 정보가 삽입되어 한 단계 확장된 형태를 가진다. 또한, 위치 정보 화일은 압축된 형태로 저장된다.



(그림 4.1) 내용 인덱스의 구조

DF(Document Frequency)는 문서 출현 횟수로서, 색인어가 출현한 문서의 개수를 나타내며, DID(Document ID)는 각각의 문서가 가지는 고유한 문서 식별자를 나타낸다. EF(Element Frequency)는 문서에서 색인어가 출현한 엘리먼트 갯수를 나타내며, 각각의 색인어는 하나의 문서를 구성하는 여러 엘리먼트에서 출현할 수 있으므로 여러 개의 엘리먼트 정보를 갖게 되며, 각 문서마다 서로 다른 EF 값을 가지게 된다. OID(Object ID)는 엘리먼트마다 가지는 고유한 식별자이며, EID(Element name ID)는 각각의 엘리먼트 이름에 대한 식별자이다. 엘리먼트 이름은 그 크기가 가변이고, 대부분 식별자의 크기보다 큰 특성을 가지므로 엘리먼트 이름을 저장하는 대신 각각의 엘리먼트 이름에 대해서 고유한 식별자를 부여하여 이를 저장한다. TF(Term Frequency)는 하나의 엘리먼트 안에서 출현한 색인어의 갯수를 나타내며, 각각의 위치 정보 (P, S, W)를 위치 정보 화일에 저장한 후, locId(Location ID)로 연결한다.

그리고, 내용인덱스의 저장공간을 줄이기 위해 위치 정보 화일의 압축 방법을 수행한다. 위치 정보는 문서에 대해서 색인어가 출현한 위치를 나타내는 것으로 P(Paragraph), S(Sentence), W(Word)의 세 가지 값으로 표현된다. 하나의 엘리먼트 안에서 색인어는 여러 번 나올 수 있으므로 위치 정보 화일은 (P, S, W)의 연속된 값을 가진다. 하나의 엘리먼트 안에서 출현한 (P, S, W) 정보들은 P또는 S값의 중복이 종종 일어난다. 따라서, 압축 알고리즘에서는 (P, S, W) 중에서 중복된 값은 저장하지 않고, 변경된 값의 증가분(Increment 값)만을 저장하여 저장 공간을 줄인다. (그림 4.2)는 그 예이다.



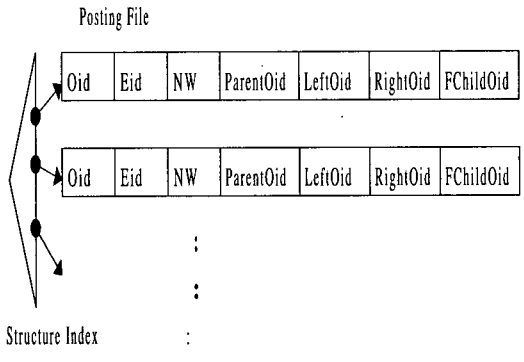
(그림 4.2) 압축 예제

위의 예에서5개의 위치 정보를 저장하는데 원 데이타는 모두 15개의 원소를 저장해야 하는데 반해, 압축 데이타는 10개의 원소만 저장하면 된다. 또한, 압축 데이타는 실제값이 아닌 증가값(Increment)만을 저장하므로 원 데이타를 저장하는 것보다 기본 데이타 크기가 더 작을 수 있다. 이를 위해 비트-패턴(bit-pattern)을 사용한다. 즉, 각 원소의 증가값을 저장하는데 기본적으로1바이트를 사용하며, 이 크기를 넘는 증가값에 대해서는 2바이트를 사용한다. 각각의 정보에 대한 증가값의 비트-패턴 정의는 다음과 같다.

00XXXXXX : 1바이트W증가값 저장 (0~63)
01XXXXXX : 1바이트S증가값 저장 (0~63)
100XXXXX : 2바이트W증가값 저장 (0~8191)
101XXXXX : 2바이트S증가값 저장 (0~8191)
110XXXXX : 1바이트P증가값 저장 (0~31)
111XXXXX : 2바이트P증가값 저장 (0~8191)

4.2 구조 인덱스

구조 인덱스는 문서를 구성하는 엘리먼트들 간의 포함 관계를 검색하기 위한 것으로, 문서의 논리적인 구조를 정보의 손실 없이 표현할 수 있어야 하며, 효율적인 저장 공간과 탐색 시간을 보장해야 한다. 3장에서의 구문 트리 비교를 통해 효율성이 입증된 엘리먼트 단위 구문 트리를 기반으로 구조 인덱스를 설계한다. (그림 4.3)은 구조 인덱스의 구조를 나타낸다.



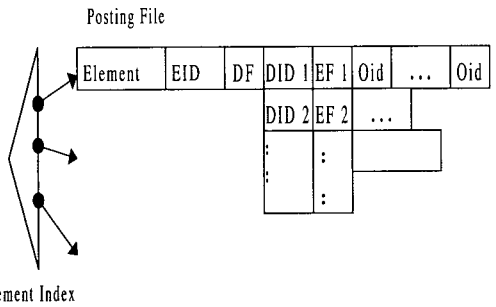
(그림 4.3) 구조 인덱스의 구조

구조 인덱스는 엘리먼트마다 고유한 식별자 값인 OID값으로 인덱스가 구성되며, 엘리먼트 간의 포함 관계에 대한 빠른 탐색을 위해 부모 노드의 OID값(ParentOid), 왼쪽 형제 노드의 OID값(LeftOid), 오른

쪽 형제 노드의 OID값(RightOid), 그리고 첫번째 자식 노드의 OID값(FChildOid)에 대한 정보를 가지고 있다. 또한, 부모 노드와 자식 노드 간의 중요도를 나타내기 위해 가중치 NW(Node Weight)에 대한 정보를 포함한다. NW는 엘리먼트 내의 색인어(term)를 벡터로 표현하여 엘리먼트 term 벡터간의 유사도로 나타낸다[17].

4.3 엘리먼트 인덱스

엘리먼트 인덱스는 구조 검색의 경우 시작 엘리먼트를 검색하기 위해서 사용되며, 또한 내용 인덱스나 애트리뷰트 인덱스에서 얻은 엘리먼트 이름 식별자(EID)와 엘리먼트 이름(Element Name)을 매핑시켜 주는 역할을 수행한다. (그림 4.4)은 엘리먼트 인덱스의 구조를 나타내며, 각 필드의 내용은 4.1에서의 내용 인덱스 구조와 동일한 의미를 지닌다.

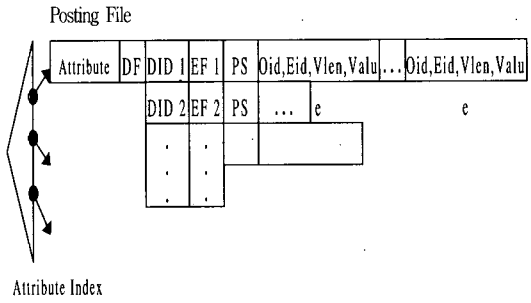


(그림 4.4) 엘리먼트 인덱스의 구조

4.4 애트리뷰트 인덱스

애트리뷰트 인덱스는 엘리먼트가 가지는 애트리뷰트 이름(Attribute Name)과 그 특성값(Attribute Value)에 대한 검색을 수행한다. 애트리뷰트가 가지는 값은 여러 형태로 존재할 수 있다. 즉, 문자열의 값을 가지는 경우도 있고, 숫자가 올 수도 있으며, 특성값을 DTD에 제한된 형태로 정의할 수도 있다. 하지만, 이러한 특성값의 형태적인 분류는 인덱스 관점에서 큰 의미가 없으므로, 설계하는 인덱스 구조에서는 애트리뷰트의 특성값을 실제 문서에서 나타나는 문자열의 형태로 저장한다.

(그림 4.5)는 애트리뷰트 인덱스의 구조를 나타낸다. 애트리뷰트가 지니는 특성값의 길이는 가변이므로 특성값의 길이를 나타내는 Vlen(Value Length)을 실제 특성값(Value)과 함께 저장한다. 또한, 검색시 다음 문서의 포스팅 정보를 빠르게 접근하기 위해서, 문서가 가지는 포스팅 정보의 길이를 PS(Post Size)에 저장하도록 한다. 엘리먼트 이름과 애트리뷰트 이름은 문서의 해당 SGML DTD에 따라 고정된 개수로 정해지며, 그 수가 내용 인덱스의 색인어에 비해 상당히 적다는 특징을 갖는다. 따라서, 엘리먼트 인덱스나 애트리뷰트 인덱스는 빠른 탐색을 위해 인덱스 초기화 시 메모리에 상주시켜 사용할 수 있다.



(그림 4.5) 애트리뷰트

5. 시스템 구현 및 성능 평가

설계된 SGML 정보 검색 인덱스 관리자는 듀얼(dual) 168MHz CPU, 메모리 516MB의 SUN 엔터프라이즈 3000에서 개발되었으며, SPARCompiler C++ 4.0.1컴파일러를 사용하여 구현되었다. 하부 저장 시스템으로는 O2 OODBMS v4.6의 O2Store를 사용하였으며[13], SGML 파서로는 시스템 공학 연구서의 KLE파서를 이용하였다. 성능 평가에 사용된 데이타는 실제 응용에서 사용되고 있고, 시험에 따른 결과의 분석이 용이한 ETRI로부터 제공된 논문 인스턴스를 사용하였다. 사용된 논문 인스턴스는 <표 5.1>과 같고, 이는 정보과학회 학회 및 정보처리 학회에 실린 논문을 SGML문서로 구성한 것이다.

〈표 5.1〉 시험 데이터의 구성 요소

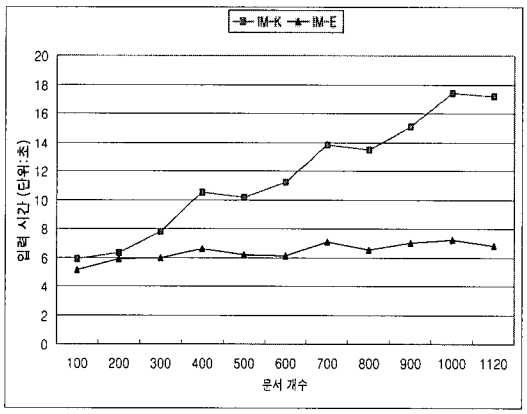
구 분	논문 인스턴스
문서 개수	1,120 건
평균 문서 크기	46.3KB
엘리먼트 개수	1,419,870
애트리뷰트 개수	301,860

구현된 SGML 정보 검색 인덱스 관리자의 성능평가

는 크게 다음과 같이 4가지 성능평가 항목을 통해 이루어진다. 4가지 항목을 통해 본 논문에서 설계한 엘리먼트 단위 구문 트리에 기반한 인덱스 관리자(Index Manger based on Element Unit Parse Tree : 이하 IM-E라 호칭한다)를 기존의 K-ary기반 인덱스 관리자(Index Manager based on K-ary : 이하 IM-K라 호칭한다)와 성능 비교를 수행한다.

5.1 문서 삽입 시간

삽입에 대한 성능 평가는 각 문서에서 추출된 색인어 및 구조 정보를 정보 검색 인덱스 관리자를 통하여 삽입하는 데 소요되는 시간을 측정한다. (그림 5.1)은 문서 삽입에 대한 성능 평가 결과이다. 엘리먼트 단위로 색인을 하기 때문에 각각의 키워드에는 엘리먼트 식인 정보가 삽입된다. IM-E의 경우는 문서 하나 삽입할 때의 평균 시간이 6.44이고 IM-K는 11.84초가 소요된다. 이때, IM-E가 IM-K보다 성능이 우수한 이유는 IM-E가 문서 구조를 그대로 반영하여 엘리먼트 ID 값을 저장하는 반면에 IM-K는 문서의 구조 정보를 유지하기 위해서 엘리먼트 ID값과 더불어 K-ary 완전 트리의 노드 번호가 같이 입력이 되어야 한다. 이 때문에 IM-K는 엘리먼트에 대한 색인 정보가 커지게 되고 삽입시간도 IM-E보다 더 많이 소요된다.



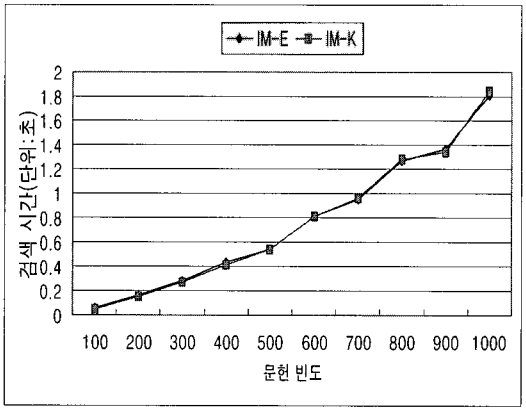
(그림 5.1) 문서 삽입 시간

5.2 색인 정보 검색 시간

검색에 대한 평가는 내용, 엘리먼트, 애트리뷰트, 구조 질의로 나누어 수행을 하며, 특히 구조 질의의 경우 <표 3.1>에서와 같이 세가지로 분류하여 수행한다.

5.2.1 내용 검색

내용 검색의 경우 저장된 색인어의 분포도와 검색되어질 문서의 수를 고려하여 색인어를 254개를 선정한 후, 각각의 색인어에 대해 관련 문서를 찾는 시간을 측정하였다. (그림 5.2)는 내용 검색에 대한 성능 평가 결과를 나타낸다. 키워드 하나에 대해 걸리는 시간은 IM-E경우에 0.69초이고, IM-K의 경우는 0.69초이다. IM-E는 색인 정보가 엘리먼트 ID인데 반해, IM-K는 K-ary 정보가 더 추가 된다. 따라서 색인 정보를 읽어올 때 IM-E가 더 우수한 성능을 보일 것으로 예상되나 IM-E는 부가 저장 공간을 축소하기 위해서 위치 정보를 압축하기 때문에, 복원하는 시간이 소요되므로 IM- E와 IM-K는 거의 같은 검색 시간을 보인다.



(그림 5.2) 내용 검색 시간

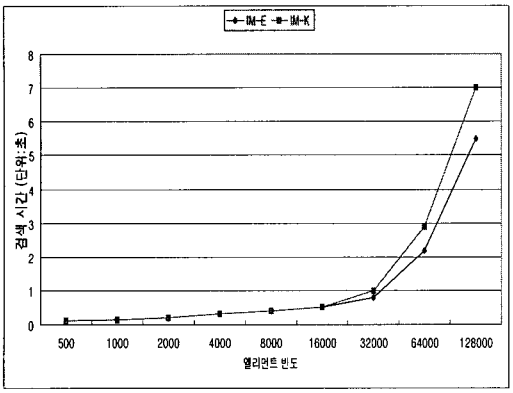
5.2.2 엘리먼트 검색

단순히 엘리먼트 이름에 대한 검색을 수행하는 엘리먼트 검색에 대한 성능 결과는 (그림 5.3)과 같다. 하나의 엘리먼트 이름에 대한 평균 검색 시간은 IM-E의 경우0.42초, IM-K의 경우 0.49초이다. 두 인덱스 관리자가 거의 유사한 성능 결과를 보였다.

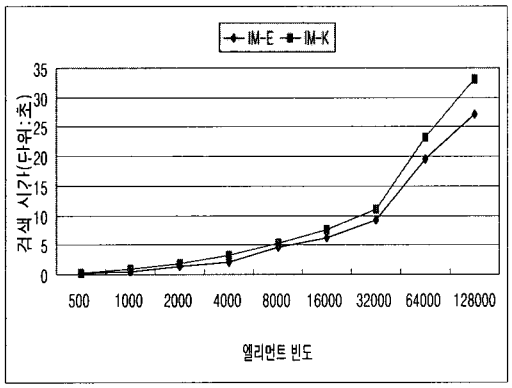
5.2.3 구조 검색

구조 검색의 경우, 먼저 직접 질의로서 특정 엘리먼트의 첫번째 자식을 검색하는 질의를 수행하였고 그에 대한 성능 결과는 (그림 5.4)와 같다. 첫번째 자식을 검색하는 경우에 있어, IM-E는 평균 4.16초이다. 그리고 IM-K는 평균 5.25초이다. IM-E가 IM-K보다 우수한 이유는 특정 엘리먼트가 직접적으로 탐색할 수 있

는 첫번째 자식과 왼쪽/오른쪽 첫번째 형제, 그리고 부모에 대한 정보를 가지고 있으므로 실제 첫번째 자식이 없는 경우를 바로 알 수 있다. 그러나 IM-K는 첫번째 자식에 대해 공식을 이용해 계산하고 다시 한번 구조 인덱스를 접근해야 하기 때문이다.

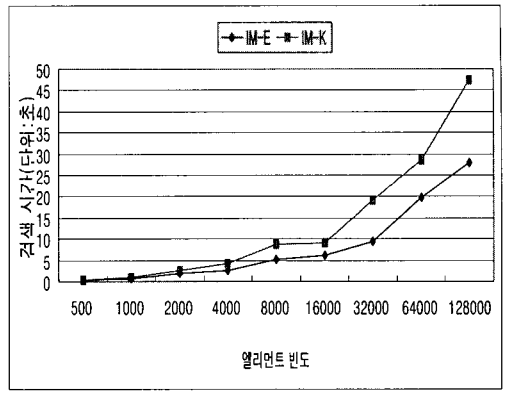


(그림 5.3) 엘리먼트 검색 시간



(그림 5.4) 직접 질의

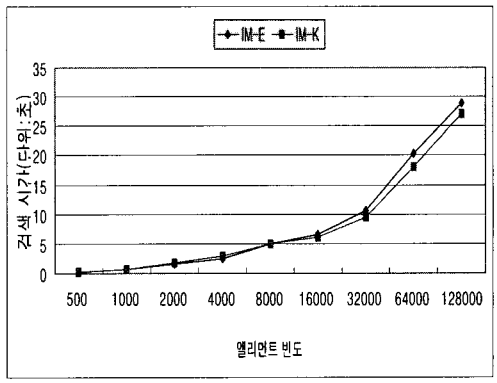
집합 질의에 대한 성능 평가를 하기 위해서, 특정 엘리먼트들의 모든 자식을 검색하는 질의를 수행하였고 그에 대한 성능 결과는 (그림 5.5)와 같다. 특정 엘리먼트의 모든 자식을 검색하는 경우에 있어, IM-E는 4.60초, IM-K는7.27초이다. IM-E가 IM-K보다 우수한 이유는 검색할 집합에 대해 실제 구성 요소에 해당하는 엘리먼트가 없을 경우, IM-K는 계산된 K-ary 완전 트리 노드 번호를 B+트리에서 탐색한 후에야 알 수 있기 때문이다. 반면에 IM-E는 특정 엘리먼트를 중심으로 직접 탐색할 수 있는 구조 정보를 가지고 있으며,



(그림 5.5) 집합 질의

따라서 집합에 해당되지 않는다는 정보를 바로 알 수 있다. 그리고 구조 인덱스를 구성할 때, IM-K는K-ary 완전 노드 번호와 문서 번호로 구성된 복합키로서 B+트리를 구성하는 반면에 IM-E는 엘리먼트 ID로만 B+트리를 구성하기 때문이다.

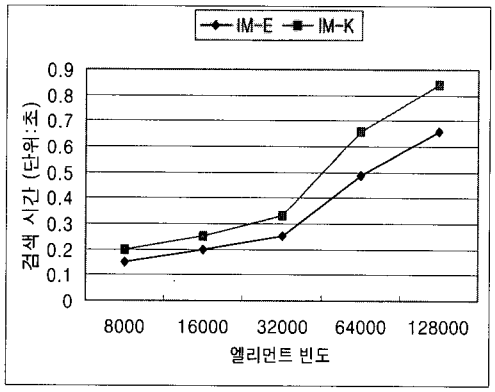
마지막으로 순서 질의에 대한 성능 평가를 위해 먼저 엘리먼트가 지니는 자식에 대한 차수가 전체 문서에 대하여 평균 2.81임을 구했다. 따라서, 순서에 대한 평균적인 검색 시간을 얻기 위해서, 특정 엘리먼트의 3번째 자식을 검색하는 질의를 수행하였고 그에 대한 성능 결과는 (그림 5.6)과 같다. 특정 엘리먼트의 3번째 자식을 검색하는 순서 검색의 경우, IM-E는 평균 4.60초이고 IM-K는4.48초이다. IM-E가 IM-K보다 성능이 저조한 이유는 IM-K는 공식을 이용해 바로 검색이 가능하지만 IM-E는 검색하고자 하는 엘리먼트까지 순회를 해야 하기 때문이다.



(그림 5.6) 순서 질의

5.2.4 애트리뷰트 검색

애트리뷰트에 대한 성능 결과는 (그림 5.7)과 같다.

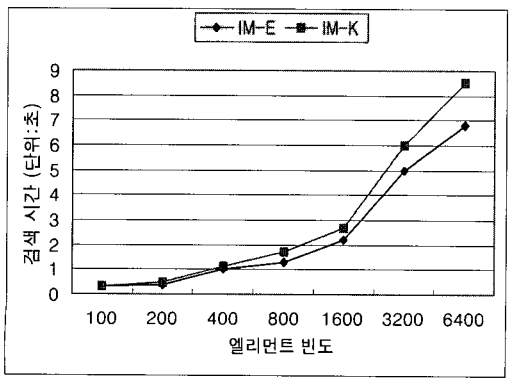


(그림 5.7) 애트리뷰트 검색 시간

애트리뷰트 검색 시간에 있어, IM-E는 0.34초, IM-K는 0.41초이다. IM-E가 IM-K보다 우수한 성능을 보이는 이유는 K-ary는 애트리뷰트 색인 정보에 엘리먼트 ID와 함께 크기가 큰 K-ary 완전 노드 정보를 유지하는 반면, IM-E는 상대적으로 크기가 작은 엘리먼트 ID를 유지하기 때문이다.

5.2.5 복합 검색

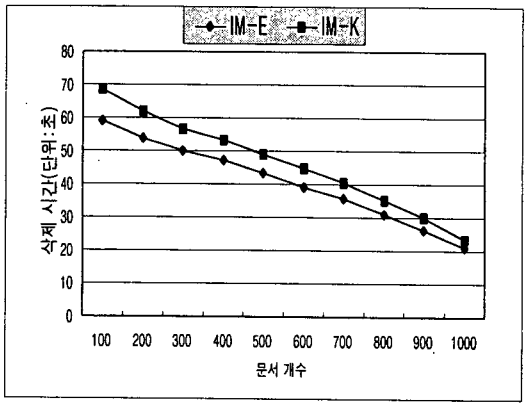
복합 질의에 대한 성능평가를 수행하기 위해, SGML 정보 검색에서 널리 수행되는 내용 +구조에 대해서 질의 평가를 수행하였다. 그 예로서 특정 키워드를 가지는 엘리먼트의 조상 가운데에 특정제목을 가지는 엘리먼트에 대한 검색을 수행하였다. 이에 대한 성능 결과는 (그림 5.8)과 같다. IM-E는 평균 2.57초이고 IM-K는 3.11초가 소요되었다. 이것은 IM-K가 구조 정보를 유지하기 위해 K-ary 완전 트리 노드 번호를 유지할 뿐만 아니라, 조상이 없는 경우에도 공식을 통해 계산된 노드 번호를 다시 한번 트리를 통해 탐색해야 하기 때문이다. 단순 검색보다 시간이 더 많이 소요되는 이유는 두 개 이상의 인덱스를 탐색해야 하기 때문이다.



(그림 5.8) 복합 질의 검색 시간

5.3 문서 삭제 시간

삭제에 대한 성능 평가를 위해 문서 1,120건에 대하여 936건을 삭제하였다. 문서 삭제가 진행됨에 따라 탐색해야 할 포스팅 화일 크기가 줄어들기 때문에, 시간이 점점 단축됨을 알 수 있다. 그리고 문서 삭제 시간이 문서 삽입 시간보다 많이 걸리는 이유는, 문서 삽입 때에는 엘리먼트 단위 색인 정보를 키워드 인덱스, 엘리먼트 인덱스, 구조 인덱스 및 애트리뷰트 인덱스의 B^+트리를 통해 입력이 되지만, 문서 삭제 때에는 포스팅 화일에 대하여 문서 ID를 통한 순차 탐색을 수행하여 삭제하기 때문이다. 문서 삭제에 대한 성능 결과는 (그림 5.9)와 같다. 문서 1건당 소요되는 시간은 IM-E가 39.30초이고 IM-K가 49.30초이다. IM-E가 IM-K보다 성능이 좋은 이유는 IM-K가 K-ary 완전 트리 노드 정보를 유지하기 위한 부가 저장 공간이 별도로 필요하고 이에 따라 포스팅 파일의 크기가 IM-E보다 크기 때문이다.



(그림 5.9) 문서 삭제 시간

5.4 부가 저장 공간

부가 저장 공간의 평가는 SGML문서의 화일 크기에

대한 전체 인덱스의 화일 크기의 비율을 측정한다. 표 5.2는 인덱스의 부가 저장 공간 비율을 나타낸다. IM-E가 IM-K보다 훨씬 더 좋은 성능을 보이는 것은, IM-K가 문서 구조를K-ary 완전 트리 구조로 변환해서 저장하기 때문이다. 또한 부분적으로 부가저장 공간의 효율성을 높이는 것은 위치 정보에 압축을 수행하기 때문이다. 실제 구조화 인덱스에 위치 정보를 저장하는 데 있어, 압축했을 때 14256page이며, 압축을 수행하지 않았을 때 15833page이다. 약 10% 정도 절약이 된다. 그러나 이론적으로는 논문 1,120건에서 색인 추출된 키워드의 위치 정보(P,S,W) 4,174,584개를 압축하였을 때에는 11,251,505byte, 압축을 하지 않을 때에는 25,047,504byte였다. 약 55%의 압축 효율을 얻었다. 이론 값과 실제 값의 차이가 나는 이유는 O2 Store가 페이지 단위(4KB)로 데이터를 관리하기 때문이다. 전체 인덱스에서 위치 정보가 차지하는 비율이 14.34%정도에 지나지 않기 때문에, 실제 압축으로 인해 축소된 부가 저장 공간은 적다.

〈표 5.2〉 부가 저장 공간 비율

	IM-E	IM-K
원 데이터 크기	51 MB	
인덱스 파일 크기	388 MB	468 MB
부가 저장 공간	760 %	918 %

5.5 고 찰

앞 절에서 수행한 성능 평가의 전체적인 비교 결과를 <표 5.3>에 나타낸다. 표에서 알 수 있듯이 전반적으로 제안한 방법(IM-E)이 기존의 방법(IM-K)보다 우수한 성능을 나타낸다. 그러나, 제안한 방법이 순서 질의에 대한 구조 검색에서는 기존의 방법보다 성능이 떨어지는 단점을 가진다. 그 이유는 기존의 방법은 공식을 이용해 원하는 엘리먼트를 바로 검색할 수 있지만, 제안한 방법은 검색하고자 하는 엘리먼트까지 순차적으로 탐색해야 하기 때문이다.

〈표 5.3〉 전체적인 성능 평가 결과

		IM-E	IM-K
문서 평균 삽입 시간		6.44초(우수)	11.84 초
부가 저장 공간		760 %(우수)	918 %
평균 검색 시간	내용 검색 시간	0.69 초	0.69 초
	엘리먼트 검색	0.42 초(우수)	0.49 초
	직접질의 검색	4.16 초(우수)	5.25 초
	집합질의 검색	4.6 초(우수)	7.27 초
	순서질의 검색	4.60 초	4.48 초(우수)
	애트리뷰트 검색	0.34 초(우수)	0.41 초

6. 결 론

본 연구에서는 동적 환경에 적합한 문서 단위 구문 트리와 엘리먼트 단위 구문 트리 인덱스 구조를 제안하였다. 또한 문서 단위 구문 트리보다 구조 검색 측면에서 우수한 엘리먼트 단위 구문 트리 인덱스 구조에 기반하여 SGML 정보 검색 인덱스 관리자를 설계 및 구현하였다. 구현된 SGML 정보 검색 인덱스 관리자는 SGML문서의 기본 단위인 엘리먼트에 대한 검색을 지원하며, 문서의 내용에 대한 검색 및 구조에 대한 검색을 동시에 지원한다. 그리고, 일관성을 유지하면서 문서의 일부분을 삽입하거나 삭제할 수 있는 엘리먼트 단위 삽입, 삭제를 지원한다. 성능 평가를 위해 기존에 구현된 K-ary 완전 트리를 기반한 인덱스 관리자와 성능 비교를 수행하였다. 검색 성능을 평가한 결과, 직접 질의나 집합 질의에서는 본 연구에서 제안한 엘리먼트 단위 구문 트리에 기반한 인덱스 관리자가 K-ary 완전 트리를 기반한 인덱스 관리자보다 우수하였으며, 순서를 부여한 구조 검색에서는 K-ary 완전 트리를 기반한 인덱스 관리자가 우수하였다.

현재 설계된 인덱스 관리자는 부가 저장 공간을 줄이기 위해 위치 정보 화일을 압축하였다. 그러나, 전체 인덱스에서 차지하는 비율이 상대적으로 작기 때문에 압축 효율이 좋지 못하다. 따라서, 앞으로의 연구는 부가 저장 공간을 줄이기 위해 전체 인덱스에서 큰 비율을 차지하는 포스팅 화일의 압축에 관한 연구를 수행하는 것이다.

참 고 문 헌

[1] Martin Colby, David S. Jackson, "SGML의 모든 것", 성안당, 1997

[2] International Organization for Standardization. Information processing - text and office system - standard generalized markup language (SGML), 1986, ISO/IEC 8879 : 1986

[3] 김현기, 이상기, 주종철, 박동인, "SGML 정보 관

리 시스템의 설계", 정보과학회 가을 학술발표논문집(A) Vol.23, No.2, pp.151-154, 1996

[4] R. Sacks-Davis, T. Arnold-Moore and J. Zobel, "Database Systems for Structured Documents," Informational Symposium on Advanced Database Technologies and Their Integration, 1994

[5] V. Christophides, S. Abiteboul, S. Cluet and M. Scholl, "From Structured Documents to Novel Query Facilities," ACM SIGMOD'94, 1994

[6] B. Lowe, J. Zobel and R. Sacks-Davis, "A Formal Model for Databases of Structured Text," In Proceedings of DASFAA'95, 1995

[7] T. Dao and R. Sacks-Davis, "Indexing Structured Text for Queries on Containment Relationships," In Proceedings of the 7th Australasian Database Conference, Melbourne, 1996

[8] M. Volz, K. Aberer and K. Boem, "Applying a Flexible OODBMS-IRS-Coupling to Structured Document Handling," In Proceedings of 12th ICDE, 1996

[9] 이희주, 장재우, 심부성, 주종철, "구조화 문서를 위한 정보 검색 인덱스의 설계", 정보과학회 가을 학술발표논문집(I) Vol.24, No.2, pp.337-340, 1997.

[10] G. E. Blake, et. al, "Text/Relational Database Management System : Harmoniziong SQL and SGML," In Proceedings of the International Conference on Applications of Databases, Vadstena, Sweden, 1994.

[11] 장재우, 이정기, 김강석, 이진수, 정보검색 응용을 지원하기 위한 MIDAS 저장시스템의 확장, 정보과학회논문지(B), Vol.25, No.4, April 1998

[12] W.B. Frakes and R. Baeza-Yates, "Information Retrieval : Data Structure & Algorithms," Prentice Hall, 1992

[13] O. Deux et al. "The O_2 System," IEEE Communication of the ACM, Vol.34, No.10, 1991

한 성 근

e-mail : sghan@dblab.chonbuk.ac.kr

1998년 전북대학교 컴퓨터공학과 졸업(학사)

1999년~현재 전북대학교 컴퓨터공학과 석사과정

관심분야 : SGML, XML, 멀티미디어 정보 검색

손 정 한

e-mail : jhson@dblab.chonbuk.ac.kr

1997년 원광대학교 컴퓨터공학과 졸업(학사)

1999년 전북대학교 컴퓨터공학과 (공학석사)

관심분야 : 멀티미디어 데이터베이스, XML, 자연어처리

장 재 우

e-mail : jwchang@dblab.chonbuk.ac.kr

1984년 서울대학교 전자계산기공학과 졸업(학사)

1986년 한국과학기술원 전산학과 (공학석사)

1991년 한국과학기술원 전산학과 (공학박사)

1996년~1997년 Univ. of Minnesota, Visiting Scholar

1991년~현재 전북대학교 컴퓨터공학과 부교수

관심분야 : 멀티미디어 데이터베이스, 멀티미디어 정보 검색, 하부 저장 구조

김 현 기

e-mail : hkk@etri.re.kr

1994년 전북대학교 컴퓨터공학과 (학사)

1996년 전북대학교 컴퓨터공학과 (공학석사)

1996년~현재 한국전자통신연구원 연구원

관심분야 : SGML/XML 문서 관리 시스템, XML EDI, STEP-SGML

강 현 규

e-mail : hkkang@etri.re.kr

1985년 홍익대학교 전자계산학과 (학사)

1987년 한국과학기술원 전산학과 (공학석사)

1992년 정보처리 기술사 자격 취득

1997년 한국과학기술원 전산학과 (공학박사)

1987년~현재 한국전자통신연구원 선임연구원, 문서정보연구팀장

관심분야 : 정보 검색, 자연언어 처리, SGML/XML, 지식 정보