J. of Basic Sciences, Cheju Nat'l Univ.
16(1), 89~97, 2003

기초과학연구 제주대학교
16(1), 89~97, 2003

XML 문서의 검색 성능 평가

신 인 혜 · 선 경 희 · 강 순 철 · 박 경 린
제주대학교 전산통계학과

요 약

대부분의 사용자들은 자료를 저장하고 검색하기 위하여 관계형 데이터베이스를 사용하여 왔는데, 최근에 차세대 웹문서의 표준으로 주목받고 있는 XML은 데이터베이스와 같이 자료를 표현, 저장, 검색할 수 있는 기능을 가지고 있다. 본 논문에서는 임의로 생성한 검색 대상 자료를 데이터베이스에 테이블로 저장하여 검색하는 방법과 XML 문서로 저장하여 검색하는 방법의 검색시간이 성능비교를 행하였다. 성능평가 결과, 검색 파일의 레코드 수가 약 1만개 이하인 경우에는 XML 문서를 이용한 검색이 빨랐고, 그 이상인 경우는 데이터베이스를 이용한 검색 시간이 월등히 빨랐다. XML 관련 검색 파일들을 세부적으로 XML 문서를 저장하는 방법(어트리뷰트로 혹은 엘리먼트)과, C#에서 XML 문서를 읽어드리는 방법(XPath을 혹은 DOM)에 따라 성능평가를 하였는데, XML 문서 작성시 어트리뷰트로 구성하고 마이크로소프트 닷넷 프레임워크에서 제공하는 XPath를 이용하여 XML 문서를 검색하는 방법이 가장 효율적이었다.

1. 서 론

고도의 정보화 사회에서 인터넷의 발달과 전자문서의 증가로 인하여 인터넷 상의 문서들이 다양한 형태의 정보들을 제공하고 있다. 텍스트 뿐만 아니라 이미지, 오디오, 비디오를 포함하고 있는 멀티미디어 문서를 전달하는 것이 보편화 되어 있고, 멀티미디어 문서의 개수도 기하급수적으로 증가하고 있다. 따라서 사용자가 요구하는 멀티미디어 문서를 다양한 방식으로 관리할 수 있는 기능이 요구되고 있다. 지금까지 인터넷 상의 대부분의 정보는 HTML(HyperText Markup Language) 문서로 구성되어 있었으나 HTML은 단지 문서의 표현을 위한 정보를 나타내므로 하나의 정의된 DTD(Document Type Definition)를 사용하기 때문에 각 문서의 엘리먼트를 의미 있는 정보로 표현하는 기능이 부족하다. 이에 W3C(World Wide Web Consortium)에서는 차세대 웹 문서의 표준으로 XML(eXtensible Markup Language)을 1998년에 지정하였다[1]. XML 문서는 구조적 데이터를 표현할 수 있으며 사용자가 정의한 DTD를 만족하는 트리 구조를 가지고 있다. 이러한 XML 문서는 구조적 문서를 표현하는데 유용하여 여러 시스템에 저장되어진 많은 다양한 정보를 표현하는데 사용되어진다. 이런 XML 문서를 데이터베이스를 이용하여 효과적으로 저장하고 관리할 수 있는 시스템이 요구되어지고 있으며 또한 많은 연구가 이루어지고 있다. XML 문서가 데이터 베이스에 저장되면 문서는 다른 데이터와 같이 질의되어 질 수 있고 동시성, 복구 제어 등의 많은 이점을 가지고 있다. 하지만 관계형 데이터베이스의 표준 질의 언어인 SQL은 정확한 데이터베이스 구조를 알아야 데이터를 질의할 수 있다. 이에 W3C에서는 XML 문서에 대한 범용적인 질의 언어로 XQL(XML Query Language)을 1998년에 제안하였다[2]. 이러한 XQL의 가장 큰 특징은 문서의 엘리먼트를 항해(navigate)하기 위해 경로를 사용한다는 점이다. 이를 통해 데이터베이스 구조에 대한 정확한 지

식 없이 데이터베이스에 저장된 XML 문서에 대해 질의가 가능하다.

본 논문에서는 XML 문서를 이용한 검색과 데이터베이스를 이용한 검색하는 시스템의 성능을 비교한다. 2장에서 XML 문서와 데이터베이스와 관련하여 현재까지 연구되고 있는 내용들을 정리하고, 3장에서는 실질적인 검색 성능 비교에 앞서, 우선 비교할 대상인 임의의 레코드 생성과 관련 파일들의 내역을 소개한다. 그리고 4장에서 조건별 검색 시간 실험 결과를 확인해 보고, 마지막 5장에서 결론을 맺는다.

2. 관련 연구

XML 문서는 기본이 되는 엘리먼트를 기준으로 하여 트리 구조로 구성되어 있으며, 기존에 이러한 구조를 가지는 XML 문서를 관계형 데이터베이스에 저장하는 여러 방식이 연구 되어왔다[3,4]. 아래에서는 이와 같은 구조를 갖는 문서의 저장시스템에 대한 관련연구를 기술한다.

2.1 파일 시스템

파일 시스템은 XML 데이터를 저장하기 위한 가장 단순한 방법으로 엘리먼트를 구성하는 서브 엘리먼트들 간의 그룹화가 가능하다는 장점을 가진다. 그러나 XML 데이터에 대한 질의가 어렵다는 단점이 있다.

2.2 XML 전용 데이터베이스 시스템

XML이 semistructured 데이터의 범주에 속하므로 semistructured 데이터를 위한 독자적인 시스템을 확장하여 XML 데이터를 저장하고자 하는 접근 방식으로서 대표적인 시스템으로는 Stanford 대학에서 개발한 lore[5]가 있다. 이 방식은 XML이 가지고 있는 반 구조적인 특성을 잘 표현하고 활용할 수 있다는 장점을 가진다. 그러나 이러한 접근 방법의 문제점은 지난 20여년 간 연구되어 온 여러 가지 데이터 베이스 기능을 활용하는데 어려움이 있다는 것이다.[6]

2.3 데이터베이스 시스템

관계 데이터베이스 시스템을 이용한 방법은 XML 문서의 저장을 위한 구조 정보를 관계 데이터 베이스에 테이블 형태로 구성하여 저장한다. 그러나 검색 시 다수의 테이블에 대한 고비용의 조인 연산을 수행해야 하며 검색 결과를 추출하는데 많은 시간과 노력이 필요하고 집합값을 지원하지 않는 등의 단점을 가지고 있다.

2.3.1 리스트형태를 이용한 방법

각 엘리먼트 노드들은 자신의 고유 번호를 갖고 있고 트리의 상위 노드는 하위 노드들의 고유 번호를 리스트 형태로 갖고 있다. 이렇게 자신의 하위 노드에 대한 번호를 이용하여 트리 구조에서의 검색이나 서브 트리에 대한 구조적 검색을 할 수 있다[7]. 트리 구조에서 특정 노드를 찾기 위해서는 트리 항해를 하여 탐색해야 하므로, 많은 탐색 시간이 소요되며, 특정 노드의 하위 서브 트리에 대한 질의나 트리 구조에서의 특정 레벨에 관련된 질의를 하기 어렵거나 효율이 낮아 문서의 운용 데이터베이스로 사용하기에는 부적합하다.

2.3.2 경로 엘리먼트 ID(Path Element ID)를 이용한 방법

경로 ID는 자신이 자식 엘리먼트 중 몇 번째 엘리먼트인가를 표시하는 엘리먼트 ID를 자신의 조상에 대한 엘리먼트 ID로부터 계속 이어받아 구성된다[8]. 즉 자기 자신에 대한 접근 경로가 된다. 이와 같은 경로 엘리먼트 ID를 사용할 경우, XML 문서의 트리 구조상 어느 위치의 엘리먼트라도 쉽게 검색해 낼 수 있다. 그러나 반복되는 엘리먼트가 존재하면 경로 엘리먼트 ID가 무한이 증가되는 경우가 발생하므로 관계형 데이터베이스에 저장하는 것이 어렵다.

2.3.3 DFS(Depth First Search) Numbering을 이용한 방법

트리의 루트 노드로부터 DFS 방식으로 노드를 방문하여 처음 방문할 때의 순서와 자신의 자식 노드의 방문이 모두 끝난 뒤의 노드의 방문 순서를 1쌍으로 구성한다[7]. 이렇게 구성된 DFS Numbering 순서쌍은 문서의 트리 구조상에서 특정 노드에 속해있는 하위 트리에 대한 질의를 하나의 SQL 문장으로 처리할 수 있다. 하지만 몇 단계 위 또는 몇 단계 아래의 특정위치 엘리먼트에 대한 질의 처리는 DFS Numbering 하나만으로는 어렵고 부가적인 속성 필드를 필요로 하거나, 추가적인 검색 및 연산이 필요하다. 또한 문서를 데이터베이스에 추가나 삭제시 문서를 트리로 구성하여 DFS Numbering 한 뒤에 저장해야 하는 작업이 필요하다.

3. 데이터베이스와 XML의 검색 비교를 위한 전제

3.1 XML 문서와 데이터베이스를 위한 임의의 레코드 파일 생성

검색 비교를 위해 기초가 되는 레코드 파일은 제주대학교 2003년 교과과정 중에서 학생정보의 단과대학, 학과, 전공, 학년 등 실제 자료를 토대로 임의로 생성하였다. 그 외에 이름 필드는 현존하는 한국 성씨를 바탕으로 임의로 만들었다. 그러나, 이 필드는 검색 과정에서 그다지 유효하지 못하여 검색 조건에서는 제외하고 필드를 채우는 용도로 쓰였다. 아이디 필드는 레코드가 생성됨에 따라 자동 증가하는 일련번호로써, 생성된 레코드의 개수를 확인할 수 있다. 또한 데이터베이스에서는 기본키(Primary Key)로 사용되어진다. 생성 레코드 파일이 XML 문서와 데이터베이스의 테이블을 만드는데 구체적으로 어떻게 쓰여졌는지는 <표 1>와 같다. 각각의 레코드 명과 필드 명을 영문자로 사용한 것은 한글과 숫자로 이루어진 필드 값과 쉽게 구별 짓기 위함이다.

XML 문서는 엘리먼트 위주로 구성하는 방법과 어트리뷰트 위주로 구성하는 방법 두 가지가 있다. XML 문서를 전자 또는 후자로 구성하는 문제는 자료의 성질과 관련성에 따라 크게 좌우된다. 모든 자료가 두 가지 방법으로 모두 표현할 수 있는 것은 아니고, 일반적으로 엘리먼트와 어트리뷰트를 적절히 혼합하여 XML 문서를 구성한다. 그러나 현 자료에서는 두 가지 형태로 나누어 표현해도 별 무리가 없으므로, 하나는 필드들을 전부 어트리뷰트로 구성하고 다른 하나는 전부 엘리먼트로 구성하여 두 방법 사이의 검색성능의 차이를 확인해본다. 표에서 유심히 보아야 할 필드는 "전공" 필드이다. 이 필드는 제주대학교에서 현행되어지는 것처럼 2학년이 되어서야 전공이 정해지기 때문에 1학년 학생의 경우는 필드 값이 없다. 이런 경우 값이 없음을 표현하기 위해 "NULL" 등의 값을 넣어 표현할 수도 있으나 저장공간을 줄이기 위해 값을 넣지 않았다. 그래서 1학년 학생 정보에 널(Null) 값을 허용하기 위해 데이터베이스의 경우 다른 필

<표 1. 생성 레코드 파일의 XML과 데이터베이스에서의 구성 내역>

필드명	XML		데이터베이스	설명
	어트리뷰트로 구성	엘리먼트로 구성		
Student	엘리먼트 이름	엘리먼트 이름	테이블 이름	학생정보
Sid	Student의 어트리뷰트	Student의 하위 엘리먼트	자동증가정수, 기본 키	아이디
Sname	Student의 어트리뷰트	Student의 하위 엘리먼트	문자열(10), NOT NULL	이름(성)
Scollege	Student의 어트리뷰트	Student의 하위 엘리먼트	문자열(50), NOT NULL	단대
Sdept	Student의 어트리뷰트	Student의 하위 엘리먼트	문자열(60), NOT NULL	학과
Smajor	Student의 어트리뷰트	Student의 하위 엘리먼트	문자열(60)	전공
Syear	Student의 어트리뷰트	Student의 하위 엘리먼트	정수, NOT NULL	학년

드들과 다르게 "NOT NULL"을 지정하지 않았고, XML 문서에서는 그 필드는 생성하지 않았다. 다시 말해서, XML 문서가 어트리뷰트로 구성된 경우 "Smajor" 어트리뷰트가 없고, 엘리먼트로 구성된 경우는 "Smajor" 엘리먼트가 없다. 그러나 검색 조건에 1학년을 넣지 않았으므로 XML 문서와 데이터베이스에서의 조건별 출력 레코드 수는 동일하다. 실제 일반적인 자료 형태는 필드 값이 없는 (NULL) 경우가 많지만, 여기서는 NULL 값을 가지는 필드에 대한 조건 검색 비교는 다루지 않았음을 밝혀둔다.

3.2 검색 대상 파일들과 검색 조건들

데이터베이스와 XML 문서의 비교를 위해서 다음의 〈표 2〉에 주어진 세 가지의 조건으로 검색하였다. 각각의 조건에 대해서 레코드 수는 5백개, 1만개, 5만개, 10만개, 15만개, 20만개, 30만개, 40만개, 50만개이다. 각 조건별로 레코드 수에 대응되는 검색에 따른 출력 레코드 수는 〈표 3〉과 같다.

〈표 3〉을 보면 알 수 있듯이 조건이 구체화될수록(많아질수록) 검색결과의 레코드 수는 줄어들게 된다. 또한 각각의 조건들에서 레코드 수가 증가됨에 따라 검색결과의 레코드 수는 거의 비율적으로 증가됨을 알 수 있다. 이는 레코드 생성의 임의성을 보여주는 것이다.

다음 〈표 4〉는 실제 레코드 수별 XML 문서의 용량 크기를 보여주고 있다. 어트리뷰트로 구성한 XML 문서보다는 엘리먼트로 구성한 XML 문서가 보다 더 많은 용량을 차지하고 있다. 이러한 사실은 XML 관련 파일들의 검색 시간 결과와 밀접한 관련이 있다. 이에 대한 자세한 내용은 다음 장에서 설명된다.

〈표 2. 비교 검색 조건〉

	구체적인 검색 조건 내용
조건 1	Syear = 2 (학년이 2학년이 학생)
조건 2	Sdept = 정보수학.전산통계학과군 and Syear = 2 (학과가 정보수학.전산통계학과군이고, 학년이 2학년이 학생)
조건 3	Scollege = 자연과학대학 and Smajor = 전산통계학과 and Syear = 2 (단과대학이 자연과학대학이고, 학과가 정보수학.전산통계학과군이고, 학년이 2학년이 학생)

〈표 3. 생성 레코드 파일별 조건별 검색에 따른 결과 레코드 수〉

레코드수 / 검색조건	5백	1만	5만	10만	15만	20만	30만	40만	50만
조건1	123	2.602	12.545	24.917	37.420	50.038	74.643	99.993	124.724
조건2	3	76	388	744	1.159	1.588	2.214	3.166	3.913
조건3	2	33	215	359	577	775	1.095	1.585	1.932

〈표 4. 어트리뷰트와 엘리먼트로 구성한 XML 문서 각각의 파일 용량〉

레코드수 / XML문서	5백	1만	5만	10만	15만	20만	30만	40만	50만
어트리뷰트(BYTE)	51	1.027	5.173	10.357	15.593	20.825	31.282	41.753	52.210
엘리먼트(BYTE)	83	1.657	8.321	16.656	25.041	33.420	50.177	66.949	83.702

3.3 기본 프로그램 환경 및 검색 대상 파일들

XML과 데이터베이스의 실질적인 검색 시간 비교를 위해 팬티엄IV급 PC(CPU:1.8GHz. RAM:512M. HDD:40GB. OS:windows 2000)에서 C# 프로그래밍 언어를 사용하였다. C#의 실행 환경을 위해 마이크로소프트 닷넷 프레임워크(Microsoft.NET Framework) SDK와 인터넷 익스플로러 6.0. 마이크로소프트 데이터 엑세스 컴포넌트 2.7을 설치하였다. XML 문서 작성시 표준 버전은 1.0를 사용하였고 데이터베이스로는 마이크로소프트 SQL 서버 2000를 사용하였다.

〈표 5〉는 닷넷 프레임워크에 포함된 클래스들 중에서 XML 문서와 데이터베이스를 읽기 위해 사용한 클래스들을 검색 파일별로 간략히 적어 놓은 것이다. XN-A 파일은 어트리뷰트로 구성한 XML 파일을 DOM(Document Object Model : 문서 객체 모델)으로 생성하여 XmlNodeReader 클래스를 이용하여 읽어들였다. XN-E 파일은 엘리먼트로 구성한 XML 문서를 DOM으로 생성하여 XmlNode 클래스를 이용하여 각 노드들을 읽어들였다. XP-A와 XP-E는 각각 어트리뷰트와 엘리먼트로 구성된 XML 문서 중에서 Xpath를 이용하여 필요한 부분만을 선택해서 읽어들였다. 마지막으로 DB-R와 DB-D는 각각 데이터베이스를 DataReader와 DataSet을 이용해서 읽어들였다. 여기서 DataReader는 자료를 검색하고 있는 동안 데이터베이스와 연결되어 있는 반면에. DataSet는 데이터베이스에서 자료를 가져온 후에 보관하고 있는 상태에서 사용하기 때문에 여러 번 검색해서 사용할 경우 데이터베이스에 다시 접속하지 않아도 된다는 차이점이 있다.

이후에 언급되어지는 모든 검색 시간은 각 단계별 총 10번의 반복 회수를 걸쳐 얻어진 값들의 평균값이다. 또한 여기서 검색시간의 측정은 닷넷 프레임워크에서 제공하는 DateTime 클래스의 속성인 Now를 이용해서 하였다.

4. 데이터베이스와 XML 문서의 검색 비교 결과

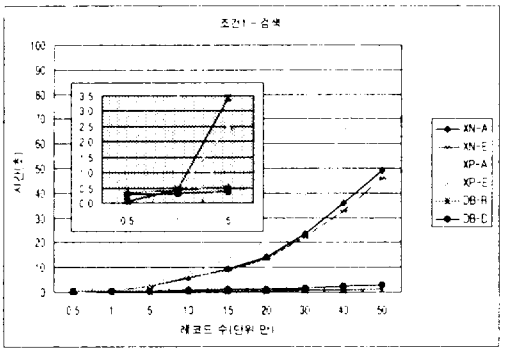
여기서 검색 시간은 순수하게 레코드를 검색하는데 걸리는 시간을 의미하는 것은 아니다. XML과 관련된 파일에서는 XML 문서 전체를 읽고 조건별로 검색한 후 조건에 맞는 레코드를 출력하는 시간을 의미한다. 데이터베이스와 관련된 파일에서는 데이터베이스연결에서부터 조건별 검색 후 해당 레코드를 출력하는데 걸리는 시간이다. 단. 실질적으로 화면에 결과를 출력하는 출력문은 주석으로 처리하였다.

4.1 XML과 데이터베이스 관련 파일들의 조건별 검색 시간 비교

다음의 〈그림 1〉은 조건 1에 대한 검색시간을 차트로 보여준다. 큰 범주인 XML와 데이터베이스를 비교해보면. 레코드 수가 증가함에 따라 XML관련 파일들(XN-A.XN-E. XP-A.XP-E)은 검색 시간이 급격히 증가하는 반면. 데이터베이스관련 파일들(DB-R. DB-D)은 상대적으로 적은 증가율을 보이고 있다. 세부적으로 살펴보면. 데이터베이스관련 파일에서 DB-R이 DB-D에 비해 작은 변동량을

〈표 5. 실질적인 검색 파일 내역〉

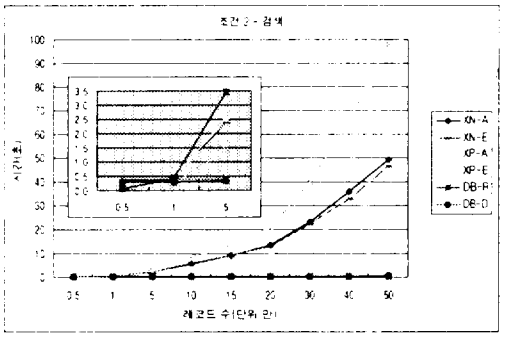
파일명	사용 레코드 형식	사용 닷넷 프레임워크 클래스들
XN-A	한 레코드를 어트리뷰트로 구성한 XML	XmlDocument. XmlNodeReader
XN-E	한 레코드를 엘리먼트로 구성한 XML	XmlDocument. XmlNode
XP-A	한 레코드를 어트리뷰트로 구성한 XML	XpathDocument. XpathNavigator
XP-E	한 레코드를 엘리먼트로 구성한 XML	XpathDocument. XpathNavigator
DB-R	데이터베이스	DataReader. Command
DB-D	데이터베이스	DataSet. DataAdapter



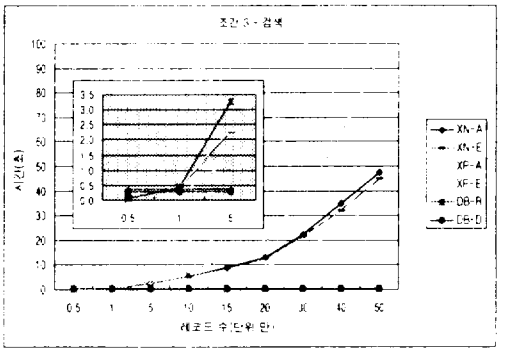
〈그림 1. 조건 1에서의 레코드 수별 각 파일들의 검색 시간〉

보이고 있으나 그 차이가 아주 작고(이에 대한 자세한 내용은 4.3절에서 설명되어진다). XML관련 파일의 경우는 XP-A가 변동량이 가장 작으며 XP-E는 가장 큰 변동량을 보인다(이에 대한 자세한 내용은 4.2절에서 설명되어진다). 그리고, XN-A와 XN-E는 중간 정도로 서로 비슷한 변동량을 보이고 있다. 주목할 점은 레코드 수가 5백 개인 경우, XML 관련 파일들이 데이터베이스관련 파일들보다 검색시간이 빠르다는 것이다. 또한 레코드 수가 1만개인 경우에도 XML 관련 파일들 중 XP-A가 데이터베이스 관련 파일들보다 더 빠른 검색시간을 보이고 있다. 게다가 XP-A이외의 다른 XML관련 파일들도 데이터베이스관련 파일들과의 검색시간 차이가 0.1초도 안 된다.

다음의 〈그림 2〉와 〈그림 3〉은 각각 조건 2와 조건 3에서의 레코드 수의 변화에 따라 XML와 데이



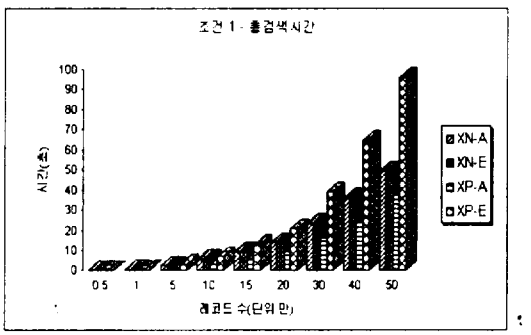
〈그림 2. 조건 2에서의 레코드 수별 각 파일들의 검색 시간〉



〈그림 3. 조건 3에서의 레코드 수별 각 파일들의 검색 시간〉

터베이스 관련 파일들의 검색시간 변화를 보여주고 있다. 앞서 나온 〈그림 1〉과 비교를 해보면, 이 그림들은 그다지 특별한 변화를 보이지 않는다. XML이나 데이터베이스 둘 다 검색조건에 따라서는, C# 파일들이 실행할 때마다 발생하는 오차에도 미치지 못하는 작은 차이를 보인다. 따라서 이 후부터는 조건1에 대해서만 고려하도록 한다.

4.2 조건 1에서의 XML 관련 파일들의 검색 시간 비교



〈그림 4. 조건 1에서 XML관련 파일들 검색시간〉

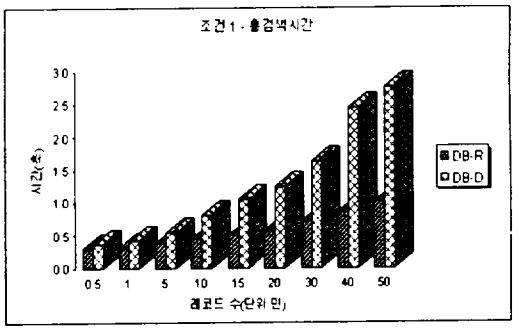
위의 〈그림 4〉는 조건 1에 대해 XML 관련 파일들에서의 검색 시간이 레코드 수의 증가에 따라 어떻게 변화하는지를 보여주고 있다. XP-A의 검색시간이 가장 빠르고, 레코드 수의 증가에 따른 검색 시간의 증가율이 가장 완만하다. 반면에 가장 느린 검색 시간을 보이는 것은 XP-E이고, 레코드

수에 따른 증가율도 가장 크다. 그리고 레코드 수가 40만개 이상인 경우 XP-E가 다른 파일에 비해 2배 이상 느린 검색 속도를 보이고 있다. 특히, XP-A와 XP-E의 검색 시간 차이는 거의 모든 경우에 2배이고, 레코드 수가 증가할수록 그 차이는 더 커지고 있다. XN-A와 XN-E는, XP-A와 XP-E의 중간 정도의 검색 시간을 보이며, 레코드 수가 커질수록 대체로 XN-E의 검색시간이 다소 느려지는 경향이 있다.

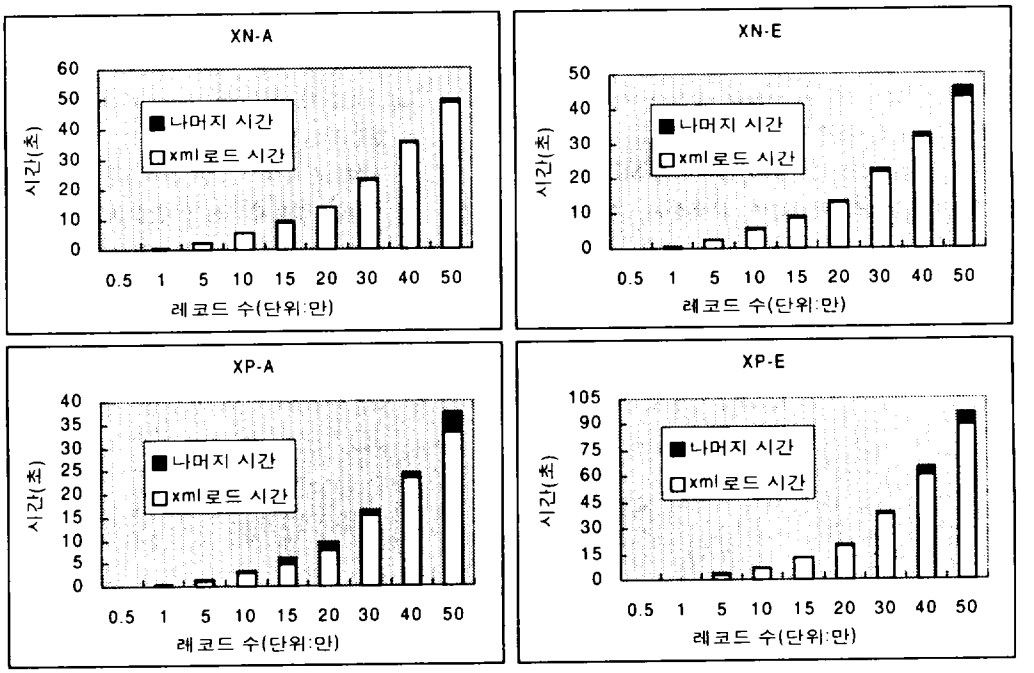
4.1에서 살펴본 바에 의하면, 레코드 수가 증가함에 따라 XML 문서에서 검색시간이 데이터베이스에서 검색시간보다 월등히 크다. XML문서의 검색시간 중에서 어떤 부분이 가장 많은 시간을 차지하는지 알아보기 위해, 검색 시간을 세부적으로 분석해보았다. 〈그림 5〉는 조건 1에 대해 XML 관련 파일들의 전체 검색 시간 중에서 XML 문서 로드시간이 레코드 수의 증가에 따라 어떻게 변화하는지를 보여준다. 모든 경우에 평균적으로 전체 검색시간에서 XML 문서 로드시간이 차지하는 비율은 거의 90%이상이다. XML 파일에서 세부적인 검색시간 차이를 분석해 본 결과, XML문서 로드시간이 상당부분 차지하고 있음을 알 수 있었다.

4.3 조건 1에서의 데이터베이스 관련 파일들의 검색 시간 비교

〈그림 6〉은 조건 1에 대해 데이터베이스 관련 파일들에서의 검색 시간이 레코드 수의 증가에 따라



〈그림 6. 조건 1에서 데이터베이스 관련 파일들 검색시간〉



〈그림 5 조건 1 : XML 관련 파일들의 전체 검색시간 중에서 XML문서 로드시간〉

어떻게 변화하는지를 보여주고 있다. DB-R과 DB-D의 차이가 비록 1. 2초 이지만, 전반적으로 DB-R이 DB-D보다 상대적으로 검색 시간이 빠르고 레코드 수에 따른 증가율도 거의 완만한 편이다.

5. 결 론

XML과 데이터베이스의 검색 시간 분석 결과, 레코드 수(파일 용량)가 많은 경우에는 XML 문서보다 데이터베이스를 이용한 검색이 보다 더 빠르지만, 레코드 수(파일 용량)가 적은 경우에는 XML 문서를 사용하는 것이 검색 시간을 줄일 수 있다. 본 논문에서 사용된 데이터를 기반으로 하여 좀 더 구체적으로 보면, 레코드 수가 약 1만개(어트리뷰트로 구성한 XML인 경우의 파일 크기:1MB 정도)를 기준으로 그 이전까지는 XML 문서를 이용한 검색이 빠르고, 그 이후부터는 데이터베이스를 이용한 검색이 더 효율적이다. 일반적으로 XML 문서를 어떻게 구성할 것인지, C#을 이용하여 XML 문서를 읽을 때는 닷넷 프레임워크에서 제공하는 어떤 클래스를 이용할 것인지 결정하는 문제는, 자료의 수가 적을 때에는 어떤 경우라도 검색 시간에 별 차이가 없으므로 그다지 중요한 문제가 아니다. 하지만 대량의 자료인 경우는 XML 문서를 어트리뷰트 위주로 구성하고, C#을 이용하여 XML 문서를 읽을 때는 Xpath를 이용하여 검색하는 것이 효율적이다.

본 논문에서는 고려하지 않았지만, XML 문서를 엘리먼트와 어트리뷰트의 혼합으로 구성한 경우나 필드 값이 NULL을 갖는 경우들을 통해 보다 자세한 검색 시간을 비교하는 것은 향후 연구과제로 남겨둔다.

참고 문헌

[1] Extensible Markup Language(XML) 1.0. "http://www.w3.org/T-R/1998/REC-xml."

[2] XML Query Language(XQL). "http://www.w3.org-/TandS/QL/-QL98/pp/xql.html."

[3] eXcelon. http://www.odi.com. 1999.

[4] D. Florescu and D. Kossmann. "Storing and Querying XML Data using an RDBMDS." IEEE Data Engineering Bulletin 22(3). p.27-34. 1999.

[5] J. McHugh. S. Abiteboul. R. Goldman. D. Quassand J. Widom. Lore. "A Database Management System for Semistructured Data." SIGMOD Recond 26(3). p.54-66. 1997.

[6] shanmugasundaram. K Tufte. C. Zhang G. He. H. J. Dewitt. and J. F. Naughtom. "Relational Databases for Querying XML Documents: Limitableon and Opportunlties." Proc. Of 25th intl conf. On VLDB. Edinburgh. Scotland. UK. p. 304-314. 1999.

[7] 이용석, 손기락 "XML문서 저장 시스템 설계 및 구현," 정보과학회 학술발표 논문집(I).1998.

[8] 연제원, 조정수, 이강찬, 이규철, "XML 문서 구조검색을 위한 저장 시스템 설계," 정보과학회 학술발표 논문집(B), 26권 1호, 1998.

Search Time Analysis for XML Documents

In-Hye Shin, Kyung-Hee Seon, Soon-Chul Kang, Gyung-Leen Park
Department of Computer Science and Statistics. Cheju National University

Abstract

While most of users have used relational databases to store and search data. XML documents also can represent. store. and search these data like database systems. This paper compares the search time of data in the XML documents with those in database systems. The performance comparison shows that the search time using the XML document is faster than that using database when the number of records is less than ten thousand. However. the latter is much faster than the former when the number of records is more than ten thousand. The XML documents can be consist of either using Attributes or Elements. Also. the document can be read either using DOM or XPath. The performance comparison shows that the XML document made up of Attributes and being read using XPath in Microsoft.Net Framework provides the fastest search time.