추론한 데이타 타입을 이용한 질의 가능 XML 압축
(A Queriable XML Compression using Inferred Data Types)

박 명 제 † 민 준 기 †† 정 진 완 †††
(Myung-Jae Park) (Jun-Ki Min) (Chin-Wan Chung)

요 약 HTML은 데이타베이스와 같은 특수한 형태의 저장소 대신, 전형적인 파일 시스템에 저장되는 경우가 대부분이다. 이와 마찬가지로, 최근 인터넷 상에서의 데이타 교환 및 표현의 표준으로 부각되는 XML 역시 파일 시스템을 통하여 저장되는 경우가 현저하다. 하지만, XML 문서가 지니는 비정규적인 구조와 장황성 때문에, 디스크 공간이나 네트워크 상의 대역폭의 사용이 정규적인 구조를 지니는 데이타에 비해 크다. 이러한 XML 문서의 비효율성을 해결하고자, XML 문서의 압축에 관한 연구가 진행되었다. 최근에 연구된 XML 압축 기법들을 살펴보면, 압축된 XML 문서에 대한 질의를 전혀 지원하지 않거나, 질의를 지원하더라도 XML 문서 내의 데이타 값들의 특성을 고려하지 않고 단순히 기존의 압축 방법들을 적용하기 때문에 영역 질의를 지원하기 위해서는 압축의 일부를 복원해야 한다. 그 결과, 압축된 XML 문서에 대한 질의 성능이 저하되었다. 따라서, 본 연구에서는 압축된 XML 문서에 직접적이고 효율적인 질의를 지원하는 XML 압축 기법을 제안하고자 한다. XML 문서의 각 태그를 사전 압축 방법을 사용하여 압축하고자 하며, 태그 별로 데이타들의 타입을 추론하여 추론된 타입에 적절한 압축 방법을 사용하여 데이타 값들을 압축하고자 한다. 또한, 제안하는 압축 기법의 구현 및 성능 평가를 통하여, 구현한 XML 압축기가 실생활에 사용되는 XML 문서들을 효율적으로 압축하며 압축된 XML 문서에 대해 향상된 질의 성능을 제공하는 것을 보인다.

키워드 : XML, XML 압축, 준적응 압축, 준동형 압축

Abstract HTML is mostly stored in native file systems instead of specialized repositories such as a database. Like HTML, XML, the standard for the exchange and the representation of data in the Internet, is mostly resident on native file systems. However, since XML data is irregular and verbose, the disk space and the network bandwidth are wasted compared to those of regularly structured data. To overcome this inefficiency of XML data, the research on the compression of XML data has been conducted. Among recently proposed XML compression techniques, some techniques do not support querying compressed data, while other techniques which support querying compressed data blindly encode data values using predefined encoding methods without considering the types of data values which necessitates partial decompression for processing range queries. As a result, the query performance on compressed XML data is degraded. Thus, this research proposes an XML compression technique which supports direct and efficient evaluations of queries on compressed XML data. This XML compression technique adopts an encoding method, called dictionary encoding, to encode each tag of XML data and applies proper encoding methods for encoding data values according to the inferred types of data values. Also, through the implementation and the performance evaluation of the XML compression technique proposed in this research, it is shown that the implemented XML compressor efficiently compresses real-life XML data sets and achieves significant improvements on query performance for compressed XML data.

Key words : XML, XML Compression, Semi-Adaptive Compression, Homomorphism

· 본 연구는 정보통신부의 대학 IT연구센터(ITRC) 지원을 받아 수행되었습니다.

† 비 회 원 : 한국과학기술원 전자전산학과
jpark@islab.kaist.ac.kr
†† 정 회 원 : 한국기술교육대학교 인터넷미디어공학부 교수
jkmin@kut.ac.kr
††† 종신회원 : 한국과학기술원 전자전산학과 교수
chungcw@islab.kaist.ac.kr

논문접수 : 2004년 1월 5일
심사완료 : 2005년 4월 1일

1. 서 론

인터넷의 확산은 다양한 형태의 전자 문서 양을 증가시켰다. 데이타를 전자 문서 형태로 표현하는 다양한 방법들 가운데 XML(eXtensible Markup Language)[1]은 정보 표현의 유연성 때문에, 인터넷 상에서의 데이타 교환(exchange) 및 표현(representation)의 표준으로 부각되고 있다. 그 결과, 인터넷에서의 XML 문서 사용은 나날이 증가하고 있으며, 이러한 XML 문서의 저장[2-4], 검색[5,6] 및 출판[7,8]에 관한 다양한 연구가 진행되었다. 또한, XML 문서에 대한 질의 언어로서, XPath[9]와 XQuery[10] 같은 질의 언어들이 제안되었다. 하지만, 현재 대부분의 XML 문서는 HTML[11] 문서처럼 전형적인 파일 시스템에 저장되는 경우가 대부분이다. 그러므로, XML이 데이타를 표현하는 진정한 표준으로 자리잡기 위해서는 파일 형태로 저장되는 XML 문서를 효율적으로 관리하는 기술에 관한 연구의 필요성이 대두되었으며, 이에 따라 XML 문서의 압축 기법에 관한 연구가 진행되었다[12,13].

XMill[12]은 압축된 XML 문서의 크기를 최소화하기 위한 압축 기법이다. 태그(tag)들은 사전(dictionary) 압축 방법을 통해 압축하며, 의미상 관련(semantically related) 있는 데이타 값(data value)은 컨테이너(container)별로 분류한다. 또한, 범용 압축 라이브러리인 zlib[14]을 통해 압축한 태그와 분류된 데이타 값을 압축한다. 하지만, 압축된 XML 문서의 구조는 본래 XML 문서의 구조와 다르므로, 압축된 XML 문서에 대한 직접적인 질의 처리는 지원하지 못한다. 따라서, 질의 처리를 위해서는 압축된 XML 문서를 본래 XML 문서로 먼저 복원해야만 한다. 반면, 최근 연구된 XGrind[13]는 압축된 XML 문서에 대한 질의를 직접적으로 처리하기 위한 압축 기법이다. 하지만, XGrind는 XML 문서의 특성을 고려하지 않고, 태그는 사전 압축 방법으로, 데이타 값은 호프만(Huffman) 압축 방법으로 압축한다. 그리고, 압축된 XML 문서에 대한 질의를 처리하는 질의 처리기는 먼저 질의에서 주어진 태그들과 데이타 값들을 압축에 사용한 압축 방법들을 통해 압축한 다음, 압축된 XML 문서를 탐색하면서 해당 결과를 찾는다. 하지만, 영역(range) 질의의 경우에는 압축된 데이타 값이 영역 내에 포함되는 데이타 값인지의 여부를 판단하기 위해서 본래 데이타 값으로의 복원이 반드시 필요하다.

따라서, 본 연구에서는 압축된 XML 문서에 대한 질의를 직접적이고 효율적으로 처리하는 XML 압축 기법을 제안한다. 본 연구에서는 XML 문서의 태그를 기존의 XML 압축 기법과 유사하게 사전 압축 방법을 사용하여 압축하고, 태그 별로 데이타 값들의 타입을 추론하여 추론한 타입에 적절한 압축 방법을 사용하여 데이타 값을 압축한다. 또한, 본 연구에서 제안하는 XML 압축 기법은 압축에 사용되는 통계 정보가 압축하는 동안에도 변경되지 않는 준적응(semi-adaptive) 방식으로써, 압축하는 데이타의 위치와 관계없이 항상 동일한 값으로 압축한다. 그리고, 본래 XML 문서의 구조와 압축된 XML 문서의 구조를 동일하게 함으로써, 압축된 XML 문서에 대한 질의 처리는 본래 XML 문서로의 복원을 필요로 하지 않는다.

본 연구에서 제안하는 압축 기법을 기반으로 XML 압축기를 구현하였으며, 실제 환경에서 사용하는 다양한 XML 문서들과 질의들을 통하여 구현된 XML 압축기의 성능을 평가하였다. 그 결과, 구현한 XML 압축기는 기존의 XML 압축 기법보다 향상된 질의 성능과 적절한 압축률(compression ratio)을 보여준다. 평균적으로, 구현한 XML 압축기의 질의 성능은 기존의 XML 압축 기법보다 1.4배 정도 나은 성능을 보이며, 압축률은 80%이다.

본 논문의 구성은 다음과 같다. 2장에서는 텍스트를 위한 압축 기법들과 기존의 XML 압축 기법들에 관해 설명한다. 3장에서는 본 논문에서 제안하는 XML 압축 기법에 대해 자세히 다루며, 4장에서는 본 논문에서 제안하는 XML 압축 기법을 기반으로 구현한 XML 압축기와 질의 처리기에 대해 다룬다. 그리고, 5장에서는 다양한 XML 문서들과 질의들을 기반으로 제안하는 XML 압축 기법과 기존의 압축 기법들간의 성능 평가 결과를 보여주며, 마지막 6장에서는 본 논문의 결론에 관해 논의한다.

2. 관련 연구

2.1 텍스트 압축 기법

일반적으로, 압축 기법들은 데이타 복원력에 따라, 손실(lossy) 압축과 무손실(lossless) 압축으로 구분된다. 손실 압축은 데이타의 일부 정보를 제거하여 압축함으로써, 압축된 데이타를 복원한 결과는 본래 데이타와 다르다. 따라서, 본 연구에서는 손실 압축을 고려하지 않는다. 반면, 무손실 압축에는 정해 놓은 통계치(statistic)를 사용하거나 전혀 사용하지 않는 정적(static) 방식, 데이타의 통계치를 먼저 추출하고 추출한 통계치를 사용하여 압축하는 준적응 방식, 그리고, 데이타를 압축하면서 통계치를 추출하는 적응(adaptive) 방식이 있다[15].

정적 방식에는 데이타를 2 진수로 변환하는 이진(binary) 압축, 새로운 단어마다 다른 정수를 부여하여 부여한 정수들로 데이타를 변환하는 사전 압축, 그리고,

선택한 기준 값과 압축하고자 하는 데이타들간의 차이들로 데이타들을 변환하는 차감(differential) 압축이 있다[16]. 특히, 사전 압축에서 부여한 정수들간의 크기는 본래 데이타의 크기와 다르므로 데이타의 크기 비교를 위해서는 본래 데이타들로의 복원이 필요이다. 반면, 차감 압축은 기준 값과 압축하려는 데이타와의 차이를 기반으로 데이타를 변환하므로 본래 데이타로의 복원 없이 데이타의 크기 비교가 가능하다. 준적응 방식에는 발생 빈도가 높은 문자를 짧은 값으로, 발생 빈도가 낮은 문자를 긴 값으로 변환하는 호프만 압축[17]이 있다. 이것은 추출한 통계치를 기반으로 생성한 호프만 트리(Huffman Tree)를 사용하여 변환 값을 제공하므로, 변환 값들간의 크기 비교는 호프만 트리를 이용한 복원을 통해서만 가능하다. 마지막으로, 적응 방식에는 압축하면서 추출한 통계치를 사용하여 호프만 트리를 계속해서 변경하면서 변환 값을 부여하는 적응형 호프만(adaptive Huffman) 압축이 있다.

2.2 XML 압축 기법

XML 압축에 관한 연구로는 압축된 XML 문서에 대한 직접적인 질의를 지원하지 않는 XMill과 직접적인 질의를 지원하는 XGrind가 있다.

XMill은 압축된 XML 문서의 크기를 최소화하기 위한 압축 기법으로써, XML 문서의 데이타 값을 태그로부터 구분하여 의미상 관련 있는 데이타 값을 컨테이너라는 자료 구조로 분류하여 압축한다. 여기서, 의미상 관련 있는 데이타 값이란, 대응되는 엘리먼트(element)의 태그가 동일한 데이타 값을 의미하며, XML 문서 상의 데이타 값들을 분류하기 위한 컨테이너는 사용자에 의해서 직접 정의될 수 있다. 그리고, 태그는 사전 압축을 통해 압축하며, 컨테이너 별로 분류된 데이타 값은 해당 컨테이너에 대해 사용자가 제공하는 압축 방법을 사용하여 압축한다. 즉, 사용자가 컨테이너에 대한 압축 방법을 제공하지 않으면, 해당 컨테이너에 포함된 데이타 값을 압축하지 않는다. 마지막으로, XMill은 압축한 태그와 분류한 데이타 값을 범용 압축 라이브러리로 사용되는 zlib을 사용하여 다시 압축한다. 이 경우, 데이타 값이 해당 태그에 의해 분류되어 있으므로, 구문적, 의미론적으로 유사하여 뛰어난 압축률을 보인다. 하지만, XMill에 의해서 압축된 XML 문서는 데이타 값을 컨테이너 별로 분류했으므로 본래 XML 문서의 구조와 다른 구조를 지닌다. 따라서, XMill은 본래 XML 문서로의 복원 없이 압축된 XML 문서에 대한 직접적인 질의를 처리하는 것은 불가능하다.

본래 XML 문서의 구조와 압축된 XML 문서의 구조를 동일하게 유지하는 준동형(homomorphism) 압축 방식을 기반으로 하는 XGrind는 XMill과는 다르게 압축된 XML 문서에 대한 직접적인 질의 처리가 가능하다. XGrind는 XML 문서의 태그를 사전 압축을 사용하여 압축하며, 데이타 값은 호프만 압축이나 사전 압축을 사용하여 압축한다. 그리고, 데이타 값의 압축에 있어서, 호프만 압축과 사전 압축 중에서 적용할 압축 방법의 선택은 XML 문서의 구조 정보를 나타내는 DTD (Document Type Definition)를 통해 결정한다. 또한, XGrind의 질의 처리기는 엘리먼트를 방문할 때마다 루트(root) 엘리먼트로부터 현재 방문한 엘리먼트까지의 압축된 경로(path)를 유지하고, 해당 경로가 질의에서 주어진 경로를 압축한 경로와 동일한가를 파악한다. 또한, 호프만 압축이나 사전 압축으로 압축한 데이타 값들간의 크기 비교는 불가능하므로, 영역 질의의 경우, 본래 데이타 값들로의 복원이 반드시 필요하다.

3. 타입 추론을 통한 XML 압축 기법

3.1 제안하는 XML 압축 기법의 특징

본 연구에서 제안하는 XML 압축 기법은 준적응 방식으로, XML 문서의 압축에 필요한 통계치를 미리 추출한다. 그 결과, 동일한 태그 및 데이타 값은 해당 태그와 데이타 값의 위치와는 관계없이 항상 같은 값으로 압축한다. 또한, 본 XML 압축 기법은 XGrind처럼 압축된 XML 문서의 구조를 본래 XML 문서의 구조와 동일하게 유지하는 준동형 압축 방식을 기반으로 하여 직접적이고 효율적인 질의 처리가 가능하다. 만약, 적응 방식을 적용한다면, 통계치를 미리 추출하는데 소요되는 시간을 절약하여 압축 시간이 단축되지만 태그나 데이타 값의 위치에 따라 다른 값으로 압축되므로 질의 처리 시에 본래 XML 문서로의 복원이 필요하게 된다.

3.2 XML 문서의 태그 압축

본 연구에서는 사전 압축으로 XML 문서의 태그를 압축한다. 새로운 태그마다 0보다 큰 정수를 부여하고, 부여한 정수들로 태그들을 변환한다. 또한, XML 문서의 유일한 태그 수에 따라 다른 바이트를 사용하여 정수들을 표현함으로써 바이트의 사용을 최소화한다.

표 1 유일한 태그 수에 따른 바이트 단위

유일한 태그 수	바이트
~ 127	1
128 ~ 32766	2
32767 ~	4

표 1은 XML 문서의 유일한 태그 수에 따라 사용하는 바이트의 단위를 보여준다. 일반적으로, 7 비트로 표현 가능한 정수의 수가 127 개이므로, 유일한 태그 수가 127 개 이하인 경우에는 1 바이트를 사용한다. 이 때, 8

비트 모두를 사용하지 않는 이유는 남은 1 비트를 사용하여 압축된 값이 태그를 위한 것인지 데이타 값을 위한 것인지의 여부를 구분하기 위해서이다.(자세한 내용은 4장에서 다룬다.) 그리고, 1 바이트를 사용하는 경우와 마찬가지로, 15 비트로 표현 가능한 정수의 수가 32766 개이므로, 유일한 태그 수가 128 개 이상이고 32766 개 이하인 경우에는 2 바이트를 사용한다. 4 바이트의 경우도 마찬가지로 해석된다.

이와 같이 XML 문서에서 사용된 유일한 태그 수에 따라 다른 크기의 바이트를 사용함으로써, XML 문서상의 유일한 태그 수가 127 개 이하인 경우에는 단순히 1 바이트만을 사용하므로 본 연구에서 제안하는 XML 압축 기법의 압축률을 향상 시킨다.

3.3 XML 문서의 데이타 값 압축

XML 문서의 데이타 값을 압축하기 위해서는, 먼저 태그 별로 데이타 값들의 타입을 귀납적 추론으로 추론한다. 표 2는 추론 가능한 데이타 값의 타입을 보여준다. 특히, 정수 타입은 표 1에서 설명한 것처럼 1, 2, 그리고 4 바이트로 표현 가능한 정수들로 세분화하며, n은 임의의 바이트를 의미한다. 이것은, 앞에서 설명한 바와 같이, 모든 정수에 대해서 항상 같은 크기의 바이트를 사용하지 않음으로써, 본 연구에서 제안하는 XML 압축 기법은 데이타 값의 압축에 있어서 바이트의 사용을 최소화한다.

표 2 추론 가능한 타입

타입	바이트	설명
정수	1	1 바이트로 표현 가능한 정수
정수	2	2 바이트로 표현 가능한 정수
정수	4	4 바이트로 표현 가능한 정수
소수	4	4 바이트로 표현하는 소수
열거형	1	유일한 단어의 수가 127 개 이하인 텍스트
스트링	n	128 개 이상인 텍스트

이와 같이 데이타 값의 타입이 추론되면, 추론된 타입에 적절한 압축 방법을 사용하여 데이타 값들을 압축한다. 정수 또는 소수 타입의 경우에는, 텍스트 형태의 데이타 값을 이진 압축을 사용하여 2 진수로 변환하여 변환된 2 진수를 차감 압축을 사용하여 압축하며, 열거형 타입은 사전 압축을, 스트링 타입은 호프만 압축을 사용하여 각각 압축한다.

텍스트 형태의 데이타 값을 압축하기 위한 사전 압축이나 호프만 압축을 통해 압축된 데이타 값들간의 크기 비교는 불가능하므로, 영역 질의의 경우, 본래 데이타 값으로의 복원이 반드시 필요하다. 하지만, 정수나 소수의 경우에는 이진 압축을 사용하여 데이타 값을 2 진수로 변환한 다음, 태그 별 데이타 값 중에서 가장 작은 값을 기준으로 차감 압축을 사용하여 압축하므로 본래 데이타 값으로의 복원을 필요로 하지 않는다.

따라서, 호프만 압축과 사전 압축만을 사용하여 데이타 값을 압축하는 XGrind가 압축된 데이타 값간의 크기 비교를 위해서 본래 데이타 값으로의 복원을 항상 필요로 하는 경우와는 달리, 복원을 필요로 하지 않는 정수나 소수의 경우로 데이타 값의 타입을 세분화하여 복원이 필요한 경우를 최소화함으로써, 본 연구에서 제안하는 XML 압축 기법의 질의 성능을 향상 시킨다. 또한, 사용자가 데이타 값의 타입을 직접 정의하도록 하는 XMill과는 반대로, 사용자의 간섭 없이 자동으로 데이타 값의 타입을 추론한다.

4. 구현한 XML 압축기의 특성

4.1 XML 압축기의 구조

그림 1은 XML 파서, XML 분석기, 그리고 XML 부호기로 구성된 XML 압축기의 전체 구조를 보여준다. XML 파서는 주어진 XML 문서로부터 XML 문서의 태그, 애트리뷰트, 그리고, 데이타 값을 순차적으로 출력한다. XML 분석기는 주어진 XML 문서에 대한 통계치를 추출하고 데이타 값의 타입을 추론한다. 그리고, XML 부호기는 추출된 통계치와 추론된 타입을 기반으로 XML 문서를 압축하여 질의 가능하도록 압축한 XML 문서를 생성한다.

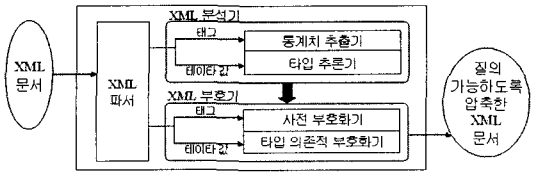

그림 1 XML 압축기 구조도

앞에서 설명하였듯이, 본 연구에서 제안하는 XML 압축 기법은 준적응 방식을 기반으로 하므로, 주어진 XML 문서를 두 번 스캔(scan)한다. XML 파서가 주어진 XML 문서를 처음 스캔하는 동안에는 XML 분석기가 동작하여, XML 문서의 압축에 필요한 정보를 수집한다. 그리고, XML 파서에 의한 두 번째 스캔에서는 XML 부호기가 동작하여, XML 문서의 태그는 사전 부호화기에 의해, 그리고 데이타 값은 타입 의존적 부호화기에 의해 압축된다.

4.1.1 XML 분석기

그림 1에서 알 수 있듯이, XML 분석기는 통계치 추출기와 타입 추론기로 구성된다.

통계치 추출기는 XML 문서의 유일한 태그 수를 계

산하고 XML 문서 상의 태그 별로 정수를 부여하며 XML 문서의 압축에 필요한 통계치를 추출한다. 이때, 각 태그에 부여된 정수는 Elemhash라는 해쉬 테이블에 저장된다. Elemhash는 XML 문서에 존재하는 각 태그에 관한 정보를 관리하는 자료 구조이며, 그림 2는 Elemhash의 구성 요소들을 보여준다.

그림 2에서와 같이, Elemhash의 구성 요소로는 해쉬 테이블의 키가 되는 태그 이름, 부여된 정수 값, 해당 태그를 가지는 엘리먼트의 데이타 값을 추론한 타입 정보, 그리고, 데이타 값과 관련된 통계치 정보가 있다. 여기서, 통계치 정보에는 최소값, 최대값, 실제 데이타 값들, 그리고, 각 문자의 발생 빈도(frequency)가 포함된다. Elemhash의 구성 요소 중에서 부여된 정수 값은 XML 부호기의 사전 부호화기가 태그를 압축하는데 필요로 하는 정보로 활용되며, 데이타 값의 추론된 타입 및 통계치 정보는 XML 부호기의 타입 의존적 부호화기가 데이타 값을 압축하는데 필요로 하는 통계 정보로 활용된다.

타입 추론기는 태그 별로 해당 태그를 가지는 엘리먼트의 데이타 값의 타입을 추론하며, XML 부호기의 타입 의존적 부호화기들이 압축에 필요로 하는 통계 정보들을 수집한다. 그림 3은 이러한 타입 추론기의 알고리즘을 보여주며, 타입 추론기의 입력으로는 Token, Pathstack, 그리고, Elemhash가 있다. Token은 그림 1의 XML 파서로부터 입력 받은 것으로 XML 문서 상에 존재하는 데이타 값을 의미하며, Pathstack은 루트 엘리먼트로부터 현재 방문 중인 엘리먼트까지의 경로를 자료 구조인 스택을 이용하여 유지한다. 따라서, Pathstack의 맨 위에는 Token으로 가지고 온 데이타 값을 가지는 엘리먼트의 태그가 존재한다. 이러한 Pathstack을 관리하는 방법으로는 XML 파서로부터 시작 태그를 받으면 해당 태그를 Pathstack에 넣고, 끝 태그를 만나면 Pathstack에서 꺼내는 방식으로 간단하게 관리할 수 있다. Elemhash는 앞에서 설명한 자료 구조로써, Pathstack의 맨 위에 있는 태그를 이용하여 해당 태그에 대한 압축 정보를 찾는다.

기본적으로, 타입 추론기는 각 태그에 대한 데이타 값들의 타입을 귀납적으로 추론한다. 먼저, Elemhash에서 각 태그의 타입은 undefined로 초기화되며, 각 데이타 값을 추론함에 따라 태그의 타입이 바뀐다. 그림 3에서 알 수 있듯이, Token, 즉 데이타 값의 타입을 추론한다.

```
Procedure Type_Inferencing(Token, Pathstack, Elemhash)
begin
1.  Tag := Pathstack.top()
2.  eleminfo := Elemhash.hash(Tag)
3.  type := Infer_Type(Token)
4.  switch(eleminfo.inferred_type) {
5.     case undefined :
6.     case integer :
7.        if(type = integer){
8.           eleminfo.inferred_type := integer
9.           intvalue := get_IntValue(Token)
10.          eleminfo.min := MIN(eleminfo.min, intvalue)
11.          eleminfo.max := MAX(eleminfo.max, intvalue)
12.          eleminfo.symhash.insert(Token)
13.          eleminfo.accumulate_chars_frq(Token)
14.       }
15.       else { // string
16.          eleminfo.symhash.insert(Token)
17.          if(the number of entries in eleminfo.symhash < 128) {
18.             eleminfo.inferred_type := enumeration
19.          }else eleminfo.inferred_type := string
20.          eleminfo.accumulate_chars_frq(Token)
21.       }
22.       break
23.    case enumeration :
24.       ...
25.       break
26.    case string :
27.       eleminfo.accumulate_chars_frq(Token)
28.       break
29. }
end
```

그림 3 타입 추론기 알고리즘

이 경우, Token을 구성하는 모든 문자가 '0'~'9'이고, 첫 번째 문자가 0이 아닐 경우, 이를 정수 타입으로 추론한다. 또한, 모든 문자가 '0'~'9'와 '.'이고, '.'은 한 번만 나타나며 첫 번째와 두 번째 문자가 각각 '0'과 '.'(예, 0.xxx), 또는 첫 번째 문자가 0이 아닐 경우(예, xxx.xxx), 이를 소수 타입으로 추론한다. 이외의 경우에는 모두 텍스트로 추론한다.

만약, 추론된 데이타 타입이 undefined이거나 정수라면(라인 5-13), 해당 데이타 값은 라인 9에서와 같이 2진수로 변환되고, 2 진수로 변환된 값을 사용하여 Elemhash의 최소값과 최대값을 수정한다(라인 10-11). 또한, 추론된 타입이 추후에 텍스트(열거형 또는 스트링)으로 바뀔 수 있으므로, 실재 데이타 값과 데이타 값 내의 문자들의 발생 빈도를 Elemhash에 포함한다(라인 12-13). 반면, 추론된 데이타 타입이 텍스트라면(라인 14-20), 현재까지 추론된 타입이 비록 정수일지라도 텍스트로 바뀐다. 이것은, 정수를 사용하여 텍스트를 표현하는 경우보다 텍스트를 사용하여 정수를 표현하는 것이 쉽기 때문이다. 추가적으로, 타입 추론기에서 추론 가능한 텍스트 타입은 표 2에서와 같이 유일한 단어의

태그 이름 (Tag)	정수 값	추론한 타입 (inferred_type)	통계치			
			최소값 (min)	최대값 (max)	실제 데이타 값 (symhash)	발생 빈도 (accumulate_chars_frq)

그림 2 Elemhash의 구성 요소

수가 127 개 이하인 열거형 타입(enumeration)과 128 개 이상인 스트링 타입(string)으로 분류된다. 그리고, 유일한 단어들만을 관리하기 위해서, Elemhash와 동일한 자료 구조인 symhash를 사용하여 반복된 단어들은 해쉬 테이블에 포함시키지 않는다(라인 12, 15). 따라서, symhash에 포함된 단어의 수에 따라, 그 수가 127 개 이하이면, 열거형 타입으로, 128 개 이상이면, 스트링 타입으로 추론한다. 또한, 열거형 타입이 추후에 스트링 타입으로 바뀔 수 있으므로, 각 문자의 발생 빈도를 Elemhash에 포함한다(라인 19).

데이타 값의 추론된 타입이 열거형인 경우(라인 22-24), 스트링 타입으로 바뀔 수 있는 가능성만 존재하므로, 라인 23은 라인 15-19와 동일하다. 그리고, 추론된 타입이 스트링인 경우에는 각 문자의 발생 빈도만 수정된다(라인 25-27). 추가적으로, 소수 타입에 대한 부분은 그림 3에서 보여주는 타입 추론기의 알고리즘을 간단하게 확장하면 가능하므로 알고리즘의 간결성을 위해서 포함하지 않는다.

4.1.2 XML 압축기

그림 1에서와 같이, XML 압축기는 태그의 압축을 위한 사전 부호화기와 데이타 값의 압축을 위한 타입 의존적 부호화기로 구성된다.

사전 부호화기는 사전 압축을 기반으로 XML 문서의 태그를 정수로 변환하고 유일한 태그의 수를 기반으로 선택된 최소한의 바이트로 변환된 정수들을 표현한다. 그리고, 압축된 값이 태그임을 효율적으로 구분하기 위해서 사전 압축으로 변환된 정수를 표현한 바이트의 MSB(Most Significant Bit)를 항상 1로 변환한다. 이것은, 앞에서 설명했듯이, 1 바이트를 사용하여 정수를 표현할 경우에 8 비트 모두를 사용하지 않고 단지 7 비트만을 사용하여 정수를 표현함으로써 표현된 바이트의 MSB는 항상 0이므로 MSB를 1로 변환하는 것은 쉽게 가능하다.(2, 4 바이트가 선택된 경우에도 동일하다.) 또한, 시작 태그와 끝 태그를 구분하기 위해서, 정수 0을 끝 태그에 할당하고, 이를 표현한 바이트의 MSB를 1로 변환한다.

타입 의존적 부호화기는 추론된 데이타 값의 타입에 적절한 다수의 부호화기들로 구성되며, 표 3은 표 2에서 설명한 추론 가능한 타입에 대한 적절한 부호화기들을 보여준다. 표 3에서 int1, int2, int4, 그리고, float은 수치 데이타 값들을 위한 차감 압축을 기반으로 하며, dict와 huff는 텍스트 형태의 데이타 값들을 위한 사전 부호화기와 호프만 부호화기이다.

수치 데이타들을 위한 부호화기들(int1, int2, int4, 그리고 float)은 XML 문서 상의 수치 값들이 문자열 형태로 표현되어 있으므로 이를 2 진수로 변환하는 이진

표 3 타입 의존적 부호화기

부호화기	설명
int1	최대값 - 최소값≤127인 정수들을 위한 부호화기
int2	128<=최대값 - 최소값≤32766인 정수들을 위한 부호화기
int4	32767<=최대값 - 최소값인 정수들을 위한 부호화기
float	소수들을 위한 부호화기
dict	열거형 타입을 위한 사전 부호화기
huff	스트링 타입을 위한 호프만 부호화기

압축을 사용하여 먼저 변환하고, 변환된 2 진수 값을 차감 압축을 사용하여 압축한다. 이 때, 차감 압축에서 사용되는 기준 값은 타입 추론기에서 추출된 통계치 중 최소값으로 한다. 따라서, 수치 데이타 값들간의 차이가 차감 압축을 통해 압축한 값에도 그대로 유지되므로, 수치 데이타에 대한 영역 질의의 경우, 본래 데이타 값으로의 복원 없이 압축된 XML 문서 상에서 질의 수행이 가능하다. dict은 유일한 데이타 값의 수가 127 개 이하인 열거형 타입을 위한 부호화기로 사전 압축을 기반으로 하며, huff는 일반적인 문자열을 위한 호프만 부호화기로 int1, int2, int4, float, dict과는 달리 가변 길이의 압축 값을 생성한다.

앞에서 설명한 태그의 경우처럼, 압축한 값이 데이타 값임을 구별하고자, 데이타 값을 압축한 값의 MSB는 태그의 경우와 반대로 항상 0이다. int1, int2, 그리고, int4는 각각 7 비트, 15 비트, 그리고, 31 비트만을 사용하여 표현함으로써, 압축한 값의 MSB는 항상 0이 되도록 한다. 또한, float은 최소값을 기준으로 차감 압축을 수행하여 음수로는 표현될 수 없으므로 MSB는 항상 0이다. 그리고, dict은 유일한 단어 수가 127개 이하인 경우를 1 바이트 중 7 비트만을 사용하여 표현하므로, MSB는 항상 0이다. 하지만, huff에 의해 압축한 데이타 값의 길이는 가변적이므로, 본 연구에서는 호프만 부호화기를 통해 생성된 압축 값, 즉 비트 스트림(bit stream)을 128 바이트보다 작은 하위 비트 스트림으로 분할하고 각 하위 비트 스트림 맨 앞에 해당 하위 비트 스트림의 바이트 길이를 표시하기 위한 1 바이트를 추가한다. 따라서, 각 하위 비트 스트림의 길이는 128 바이트보다 작으므로, 해당 비트 스트림의 길이를 표시하는 1 바이트 중에서 7 비트만을 사용하여 MSB는 항상 0이 된다.

압축된 XML 문서 상에서 압축된 값이 데이타 값을 위한 것인지, 태그를 위한 것인지의 여부는 질의 처리 또는 데이타 복원 시에 필요하다. 따라서, 앞에서 설명한 바와 같이, 데이타 값들을 압축한 값들의 MSB는 항상 0이고 태그를 압축한 값의 MSB는 항상 1이므로, 태그를 압축한 값과 데이타 값을 압축한 값은 쉽게 구분

이 가능하다.

4.2 질의 처리기

본 연구에서 제안하는 XML 압축 기법의 질의 처리기는 기본적으로 루트 엘리먼트부터 해당 엘리먼트까지의 모든 경로를 포함하는 단순 경로 표현식(simple path expression)과 일부 경로만 포함하는 부분 경로 표현식(partial matching path expression), 그리고, 특정 값을 찾는 정확 질의(exact matching query)와 특정 범위 내의 값들을 찾는 영역 질의를 지원한다.

질의 처리기는 먼저 주어진 질의에서 사용된 경로 내의 엘리먼트들을 통계치 추출기가 부여한 정수들로 변환하고, 압축된 XML 문서를 깊이 우선 검색(Depth First Search)으로 탐색하면서 방문하는 엘리먼트의 단순 경로가 정수로 변환된 경로와 동일한지의 여부를 확인한다. 만약, 주어진 질의가 정확 질의라면, 질의 조건에 사용된 데이타 값을 해당 타입 의존적 부호화기로 압축하고, 정수로 변환된 경로와 동일한 경로를 가지는 엘리먼트의 압축된 데이타 값이 질의에서 주어진 데이타 값을 압축한 값과 같은지를 비교한다. 그리고, 영역 질의의 경우, 해당 데이타 값의 타입이 int1, int2, int4, 또는 float이면, 압축된 데이타 값이 질의에서 주어진 영역 내에 포함하는지를 비교한다. 하지만, 해당 데이타 값의 타입이 dict이나 huff이면, 본래 데이타 값으로의 복원을 수행한 후 크기 비교를 한다.

5. 성능 평가

본 연구에서 제안하는 XML 압축 기법을 기반으로 구현한 XML 압축기의 성능 평가를 위해서, 실생활에서 사용되는 다양한 XML 문서들을 기반으로 기존에 연구된 XML 압축 기법인 XMill[1]과 XGrind[2], 그리고, 일반 압축기인 gzip의 성능을 비교하였다. 그 결과, 구현한 XML 압축기는 XMill에 비해 적절한 압축률을 보였다. 또한, 압축된 XML 문서에 직접적으로 질의를 수행하는 XML 압축 기법이 XGrind 외에 없는 것으로 판단되어, 질의 성능은 XGrind의 질의 성능과 비교하였으며, 그 결과, XGrind보다 현저하게 향상된 질의 성능을 보였다.

5.1 실험 환경

성능 실험은 메인 메모리 384MB에 Solaris 2.5.1이 탑재된 Sun UltraSparc-II를 사용하였고, XML 문서는 로컬 디스크에 저장하였다. 또한, 실험에서 XMill은 사용자로부터 컨테이너를 위한 압축 방법이 제공되지 않는다. 그리고, XGrind의 질의 처리기는 부분 경로 표현식을 지원하지 않으므로, 질의 성능 평가를 위하여 이 기능을 추가적으로 구현하였다.

5.1.1 XML 문서

구현한 XML 압축기의 성능 평가를 위하여 실생활에서 사용되는 네 개의 XML 문서를 사용하였으며, 표 4는 실험에서 사용된 XML 문서들의 정보를 보여준다. 크기는 메가 바이트(MB) 단위를 사용하여 문서의 크기를, 길이는 가장 긴 단순 경로의 길이를, 태그는 유일한 태그 수를, int/float은 추론한 타입이 정수 또는 소수인 엘리먼트 수를, 그리고, dict는 타입이 열거형인 엘리먼트 수를 각각 의미한다.

표 4 XML 문서 종류

XML 문서	크기	길이	태그	int/float	dict
Auction	8.53	5	30	4	19
Baseball	17.06	6	46	19	5
Course	12.28	6	18	5	4
Shakespeare	15.30	5	21	0	0

Auction[18] 문서는 경매 사이트에서 경매되는 개인 컴퓨터와 같은 경매 물품(item)들에 대한 정보를 포함하는 XML 문서로써, 지불 방법(payment types), 배달 방법(shipping information), 그리고, 물품 정보(item information)와 같은 열거형 타입을 많이 포함하며, 입찰 횟수(bid numbers)와 입찰 금액(bid amount)과 같은 수치 데이타를 포함한다. 그리고, XML 문서의 크기가 큰 경우에 대한 성능 평가를 위해서, 원래 Auction 문서를 650 배의 크기로 증가시켰다.

Baseball[19] 문서는 1998년도에 미국 메이저 리그에 참가한 각 팀의 선수들에 대한 경기 통계 정보를 포함하는 XML 문서로써, 많은 수치 데이타와 열거형 타입을 가진다. 그리고, Baseball 문서의 경우, 원래 Baseball 문서를 16 배의 크기로 증가시켰다.

Course[18] 문서는 미국 워싱턴 대학교에서 개설되는 과목들에 관한 설명이 포함되어 있는 XML 문서로써, 학점(credits)과 강의실(class rooms)과 같은 적은 수의 수치 데이타와 과목 번호(course code), 과목 이름(title), 그리고, 건물 이름(building names)과 같은 열거형 타입을 가진다. 그리고, 원래 Course 문서를 4 배의 크기로 증가시켰다.

셰익스피어의 희곡들을 XML 문서로 변환하여 만든 XML 문서인 Shakespeare[20] 문서의 경우, 모든 데이타 값들의 타입이 스트링 타입이다. 그리고, Shakespeare 문서는 37 편의 희곡을 모두 포함하며, 모두 포함한 XML 문서를 2 배의 크기로 증가시켰다.

5.1.2 XML 질의

1) http://www.research.att.com/sw/tools/xmill/에서 제공
2) http://sourceforge.net/projects/xgrind/에서 제공

표 5 XML 질의 종류

질의	질의 설명	질의	질의 설명
A1	/root/listing/bid_history/highest_bid_amount	B1	/SEASON/LEAGUE/DIVISION/TEAM/PLAYER/GIVEN_NAME
A2	//item_info[memory = '64MB']	B2	//TEAM/PLAYER[SURNAME = 'Navarro']
A3	//item_info[memory >= '32MB' and memory <= '64MB']	B3	//TEAM/PLAYER[SURNAME >= 'Navarro' and SURNAME <= 'Simas']
A4	/root/listing//seller_name[text() = 'aboutlaw']	B4	/SEASON/LEAGUE//TEAM[TEAM_CITY = 'Chicago']
A5	/root/listing//seller_name[text() >= 'aboutlaw' and text() <= 'lapro8']	B5	/SEASON/LEAGUE//TEAM[TEAM_CITY >= 'Chicago' and TEAM_CITY <= 'Toronto']
C1	/root/course/selection/session/place/building	S1	/PLAY/ACT/SCENE/SPEECH/STAGEDIR
C2	//session[section_code = 'AA']	S2	//PGROUP[PERSONA = 'EROS']
C3	//session[section_code >= 'AA' and section_code <= 'AH']	S3	//PGROUP[PERSONA >= 'EROS' and PERSONA <= 'SCARUS']
C4	/root/course//session/place[room = '100']	S4	/PLAY/ACT//SPEECH[SPEAKER = 'CLEOPATRA']
C5	/root/course//session/place[room >= '100' and room <= '200']	S5	/PLAY/ACT//SPEECH[SPEAKER >= 'CLEOPATRA' and SPEAKER <= 'PHILO']

구현한 XML 압축기의 질의 성능을 평가하기 위해서, 각각의 XML 문서마다 다섯 개의 질의를 사용하였으며, 표 5는 실험에서 사용된 질의들을 보여준다. 처음과 세 번째 열의 영문자는 질의가 수행되는 XML 문서를 의미한다. 따라서, A는 Auction 문서를, B는 Baseball 문서를, C는 Course 문서를, 그리고, S는 Shakespeare 문서를 각각 의미한다. 또한, 숫자는 사용되는 질의의 종류를 나타낸다. 질의 1은 단순 경로 표현식을 통한 질의를, 질의 2, 3은 부분 경로 표현식을 통한 정확 질의와 영역 질의를, 그리고, 질의 4, 5는 복잡한 구조의 경로 표현식을 통한 정확 질의와 영역 질의를 의미한다. 그리고, 질의 설명 열은 각각의 질의를 XPath 형태로 보여준다. 추가적으로, Auction 문서와 Baseball 문서는 열거형 타입을, Course 문서는 정수 타입을, 그리고, Shakespeare 문서는 스트링 타입에 대한 정확 질의와 영역 질의를 사용하였다.

5.2 실험 결과

실험 결과로는 먼저 각각의 XML 문서에 대한 다양한 압축기의 압축 시간을 보인다. 그리고, 각 압축기의 압축률과 XGrind와 구현한 XML 압축기에 의해 압축된 XML 문서를 XMill에서 사용한 zlib을 통해 다시 압축한 압축률을 보인다. 여기서, 압축률은 다음과 같이 계산된다.

$$\text{압축률}(\%) = \left(1 - \frac{\text{압축된 XML 문서의 크기}}{\text{원래 XML 문서의 크기}}\right) \times 100$$

마지막으로, XGrind와 구현한 XML 압축기의 질의 성능을 보인다.

5.2.1 압축 시간

그림 4는 다양한 XML 문서에 대한 여러 압축기의 압축 시간을 보여준다. 각각의 XML 문서에 대해, 네 개의 연결된 막대는 XMill, gzip, XGrind, 그리고, 구현한 XML 압축기의 압축 시간을 각각 나타낸다.

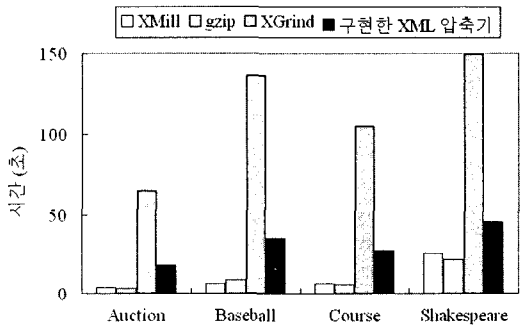

그림 4 압축 시간

그림 4에서 볼 수 있듯이, 압축에 필요한 일부 정보를 DTD의 분석을 통해 얻는 XGrind가 가장 많은 압축 시간을 필요로 한다. 또한, XGrind는 데이타 값을 호프만 압축을 사용하여 압축한다. 일반적으로, 호프만 압축은 호프만 트리의 탐색을 통해 압축 값을 생성하므로, 차감 압축에 비해 비효율적이다. 게다가, XGrind는 호프만 압축에 의해 압축된 데이타 값이 태그를 위해 사전에 정의한 값을 포함하는지의 여부를 확인하므로, 다른 압축기에 비해 더 많은 압축 시간을 필요로 한다.

반면, 구현한 XML 압축기의 호프만 부호화기는 압축된 데이타 값의 맨 앞에 압축된 데이타 값의 길이를 표시하여 압축하여, 압축된 값이 데이타 값을 압축한 값임을 구분한다. 따라서, XGrind처럼 압축된 데이타 값이 태그를 위해 사전에 정의한 값인지의 여부를 확인하는 작업을 추가적으로 필요로 않는다. 또한, 사전 압축 또는 호프만 압축만을 사용하여 데이타 값을 압축하는 XGrind와는 달리, 추론한 타입에 적절한 압축 방법(예, 차감 압축)을 사용하여 압축함으로써, 구현한 XML 압축기의 압축 시간은 XGrind의 압축 시간에 비해 현저하게 단축되었다.

그리고, XMill과 gzip은 한 번의 스캔으로 압축을 수행함으로써, 구현한 XML 압축기나 XGrind에 비해 가장 적은 압축 시간을 필요로 한다.

5.2.2 압축률

그림 5는 각각의 XML 문서에 대한 다양한 압축기의 압축률을 보여준다. 예상한 바와 같이, XMill의 압축률은 평균 92%로 가장 좋았다. 이것은, XMill이 태그와 같은 구조적인 정보를 사전 압축을 사용하여 압축하고, zlib을 사용하여 압축하기 전에 의미상 관련 있는 데이타 값을 컨테이너 별로 분류하기 때문이다.

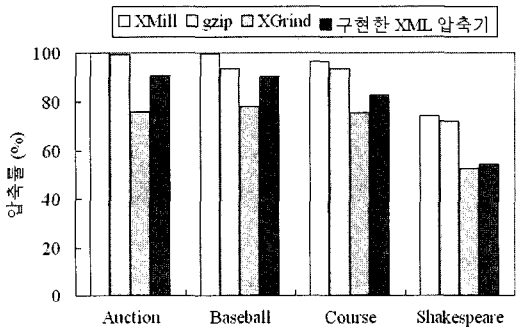

그림 5 압축률

반면, 구현한 XML 압축기의 압축률은 평균적으로 80%이다. 하지만, 구현한 XML 압축기는 데이타 값의 타입 추론을 통해 추론한 타입에 적절한 압축 방법을 제공함으로써, 데이타 값의 타입이 정수, 소수, 또는 열거형인 경우, 구현한 XML 압축기의 압축률은 더 좋아진다. 그림 5에서도 볼 수 있듯이, 열거형이나 정수 또는 소수 타입을 많이 포함하는 Auction 문서와 Baseball 문서에 대한 압축률이 다른 XML 문서들에 대한 압축률에 비해 보다 나은 성능을 보인다. 하지만, Shakespeare 문서의 경우, 구현한 XML 압축기의 압축률은 XGrind의 압축률보다 조금 나은 성능을 보인다. 이것은, 스트링 타입을 위한 호프만 부호화기가 압축된 데이타 값의 길이를 압축된 데이타 값의 맨 앞에 표시하기 때문이다. 결과적으로, 구현한 XML 압축기는 XMill이나 gzip에 비해 낮은 압축률을 보이지만, 각각의 XML 문서에 대해 적절한 압축률을 보여준다.

5.2.3 zlib을 적용한 압축률

XMill에서 사용한 범용 압축 라이브러리인 zlib에 의한 압축 효과를 알아보기 위하여, XGrind와 구현한 XML 압축기에 의해서 압축된 XML 문서를 zlib을 사용하여 다시 압축하였다. 그림 6은 zlib을 사용하여 압축한 결과를 보여준다.

XGrind와 구현한 XML 압축기에 의해 압축된 XML

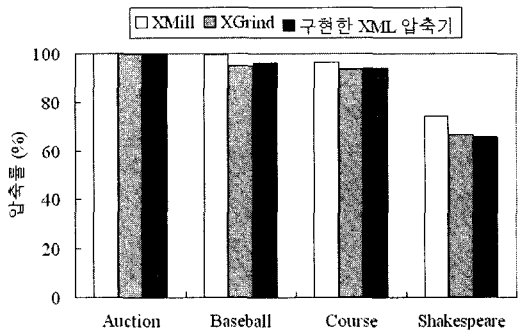

그림 6 zlib을 적용한 압축률

문서를 zlib을 사용하여 다시 압축하여도, XMill이 여전히 나은 압축률을 보인다. 이것은, XMill이 의미상 관련 있는 데이타 값들을 컨테이너 별로 분류한 뒤 분류된 컨테이너 별로 zlib을 사용하여 압축하므로, zlib의 성능이 보다 효율적이기 때문이다. 하지만, XGrind와 구현한 XML 압축기에 의해 압축된 XML 문서를 zlib을 사용하여 다시 압축한 결과, XMill의 압축률과 거의 비슷한 압축률을 보인다. 따라서, 압축된 XML 문서들 중에서 질의가 적게 사용되는 경우, zlib을 사용하여 다시 압축하여 저장한다면, 데이타의 저장 차원에서 보다 효율적일 것이다.

5.2.4 질의 성능

비록 XMill이 압축률이나 압축 시간에서 가장 나은 성능을 보이지만, XMill은 압축된 XML 문서에 대한 직접적인 질의를 지원하지 않는다. 일반적으로, XMill을 통해 압축된 XML 문서에 대한 질의 처리는 압축된 XML 문서에 대한 전체 복원을 우선적으로 필요로 하며, 전체 복원된 XML 문서를 기반으로 질의 처리를 수행해야만 해당 결과를 추출할 수 있다. 하지만, 압축된 XML 문서에 대한 전체 복원 및 복원된 XML 문서에 대한 질의 처리는 기본적으로 많은 I/O(Input/Output)를 필요로 하며, I/O의 횟수가 많으면 많을수록 질의 처리 성능은 현저하게 저하된다. 예를 들어, XML 문서를 읽어오는데 필요한 디스크 I/O가 100이라고 가정을 한다면, XMill의 경우(92% 압축률), 압축된 XML 문서를 읽어오기 위한 디스크 I/O(8), 압축된 XML 문서의 전체 복원 후 저장하기 위한 디스크 I/O(100), 그리고 전체 복원된 XML 문서를 읽어오기 위한 디스크 I/O(100)의 합인 208 I/O를 질의 처리 이전에 필요로 한다. 반면, 구현한 압축기의 경우(80% 압축률), 압축된 XML 문서에 대한 디스크 I/O(20)만을 필요로 한다. 그 결과, 구현한 XML 압축기는 XMill에 비해 현저하게 나은 질의 처리 성능을 보일 것으로 분석된다. 따라서, 구현한 XML 압축기의 질의 성능은 압축된 XML 문서에 대한

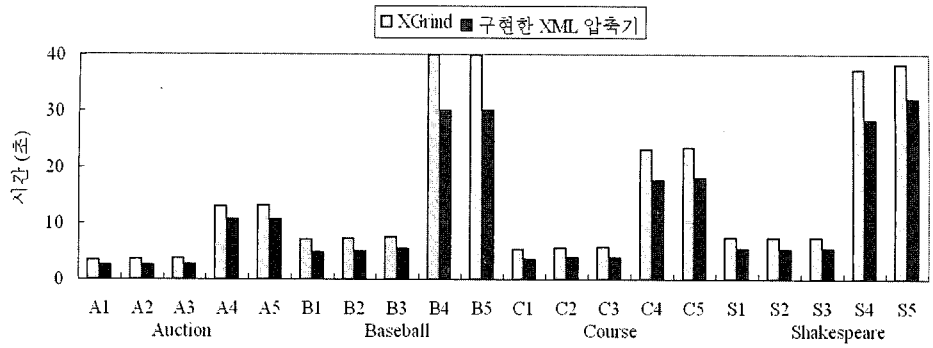

그림 7 질의 수행 시간

직접적인 질의를 지원하는 XGrind의 질의 성능을 기반으로 평가하였다. 그림 7은 XGrind와 구현한 XML 압축기에 의해서 압축된 XML 문서에 대한 질의 수행 시간을 보여준다.

그림 7에서 볼 수 있듯이, 각각의 XML 문서에 대한 모든 질의에 대해서 구현한 XML 압축기가 XGrind보다 나은 성능을 보였다. 질의 1, 2, 4는 구현한 XML 압축기가 XML 문서 상의 태그를 사전 압축을 사용하여 변환한 정수를 최소한의 바이트를 사용하여 표현하므로 항상 2 바이트를 사용하는 XGrind보다 효율적으로 경로 표현식을 처리할 수 있다. 또한, 질의 3, 5는 압축된 데이타 값이 영역 질의의 영역 내에 포함하는지의 여부를 파악하기 위해서 압축된 데이타 값을 본래 데이타 값으로 항상 복원해야 하는 XGrind의 경우를 차감 압축을 사용함으로써 최소화하였다. 따라서, 구현한 XML 압축기의 질의 성능은 XGrind보다 평균적으로 1.4 배 향상된 성능을 보였다.

6. 결 론

본 연구에서는 압축된 XML 문서에 대한 질의를 직접적이고 효율적으로 지원하는 XML 압축 기법을 제안하였다. XML 문서의 태그는 사전 압축 방법을 사용하여 정수로 변환하였으며, 정수를 표현하는데 필요한 최소한의 바이트를 사용하여 표현하였다. 또한, 데이타 값은 타입 추론기가 추론한 타입에 가장 적절한 압축 방법을 사용하여 데이타 값을 변환하였다. 특히, 정수 또는 소수 타입으로 추론된 데이타 값의 압축의 경우, 압축된 값들간의 크기 비교가 가능한 차감 압축 방법을 적용함으로써 압축된 데이타 값에 대한 복원이 필요한 경우를 최소화하였다.

그리고, 본 연구에서 제안하는 XML 압축 기법을 기반으로 XML 압축기와 질의를 압축된 XML 문서에 직접적으로 수행하는 질의 처리기를 구현하였다. 또한, 실생활에 사용되는 다양한 XML 문서들과 질의들을 선택하여 구현한 XML 압축기와 질의 처리기의 성능을 평가하였다. 성능 평가 결과, 구현한 XML 압축기의 압축률은 80%로서, 압축된 XML 문서에 직접적으로 질의를 수행하는 기존의 XML 압축 기법보다 향상된 압축률을 보였다. 또한, 질의 성능의 경우, 기존의 XML 압축 기법보다 1.4 배 향상된 질의 성능을 보였다.

참 고 문 헌

[1] T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler, "Extensible Markup Language(XML) 1.0 W3C Recommendation," http://www.w3c.org/TR/REC-XML, 1998.

[2] D. Florescu, and D. Kossman, "Storing and Querying XML Data using an RDBMS," IEEE Data Engineering Bulletin, Vol. 22, No. 3, pp. 27-34, 1999.

[3] T. Shimura, M. Yoshikawa, and S. Uemura, "Storing and Retrieval of XML Documents using Object-Relational Databases," Proc. of 10th International Conference, DEXA, pp. 206-217, 1999.

[4] I. Tatarinov, S. D. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, and C. Zhang, "Storing and Querying Ordered XML Using a Relational Database System," Proc. of the 2002 ACM SIGMOD International Conference on Management of Data, pp. 204-215, 2002.

[5] M. F. Fernandez and D. Suciu, "Optimizing Regular Path Expressions Using Graph Schemas," Proc. of the 14th International Conference on Data Engineering, pp. 14-23, 1998.

[6] R. Goldman and J. Widom, "DataGuides: Enable Query Formulation and Optimization in Semistructured DataBases," Proc. of 23rd International Conference on Very Large Data Bases, pp. 436-445, 1997.

[7] M. F. Fernandez, W. C. Tan, and D. Suciu, "SilkRoute: trading between relations and XML," WWW9/Computer Networks, Vol. 33, No. 1-6, pp.

723-745, 2000.
[8] J. Shanmugasundaram, E. J. Shekita, R. Barr, M. J. Carey, B. G. Lindsay, H. Pirahesh, and B. Reinwald, "Efficiently Publishing Relational Data as XML Documents," Proc. of 26th International Conference on Very Large Data Bases, pp. 65-76, 2000.
[9] J. Clark and S. DeRose, "XML Path Language(XPath) Version 1.0, W3C Recommendation," http://www.w3.org/TR/xpath, 1999.
[10] S. Boag, D. Chamberlin, M. F. Fernandez, D. Florescu, J. Robie, and J. Simeon, "XQuery 1.0: An XML Query Language, Working Draft," http://www.w3.org/TR/2002/WD-xquery-20020816, 2002.
[11] D. Raggett, A. L. Hors, and I. Jacobs, "HTML 4.01 Specification, W3C Recommendation," http://www.w3.org/TR/html4/, 1999.
[12] H. Liefke and D. Suciu, "XMill: An Efficient Compressor for XML Data," Proc. of the 2000 ACM SIGMOD International Conference on Management of Data, pp. 153-164, 2000.
[13] P. M. Tolani and J. R. Haritsa, "XGRIND: A Query-friendly XML Compressor," Proc. of 18th International Conference on Database Engineering, pp. 225-234, 2002.
[14] J. Gailly and M. Adler, zlib 1.1.4, http://www.gzip.org/zlib/, 2002.
[15] P. G. Howard and J. S. Vitter, "Analysis of Arithmetic Coding for Data Compression," Proc. of the IEEE Data Compression Conference, pp. 3-12, 1991.
[16] D. Salomon, "Data Compression, the complete reference," Springer-Verlag, New York, 1998.
[17] D. A. Huffman, "A Method for the Construction of Minimum Redundancy Codes," Proc. of the Institute of Radio Engineers 40, pp. 1098-1101, 1952.
[18] Anonymous, http://www.cs.washington.edu/research/projects/xmltk/www/xmlproperties.html.
[19] E. R. Harold, Long Baseball Examples from The XML Bible. ibiblio, http://www.ibiblio.org/xml/examples/baseball/.
[20] R. Cover, The XML Cover Pages, http://www.oasis-open.org/cover/xml.html, 2001.

박 명 제
2001년 한국과학기술원 전자전산학과 전산학전공(학사). 2003년 한국과학기술원 전자전산학과 전산학전공(석사). 2003년~현재 한국과학기술원 전자전산학과 전산학전공 박사 과정 재학중. 관심분야는 XML, Semantic Web

민 준 기
정보과학회논문지 : 데이타베이스
제 32 권 제 2 호 참조

정 진 완
정보과학회논문지 : 데이타베이스
제 32 권 제 2 호 참조