# XML 시맨틱 캐쉬의 교체 기법*

## A Technique of Replacing XML Semantic Cache

홍정우(Jung-Woo Hong)**, 강현철(Hyunchul Kang)***


### 초 록

전자 거래에 있어 XML로 기술된 데이터가 증가하고, 이로부터 효율적인 질의 처리를 수행하는 기능의 중요성이 커지고 있다. 질의 처리 성능을 향상하기 위해 XML 질의 결과를 캐쉬하는 방법이 주목을 받고 있는데 XML 질의 캐쉬 기법을 활용하기 위해서는 효율적인 캐쉬 교체 기법이 요구된다. 기존의 XML 캐쉬 교체 기법에는 질의 결과를 교체 단위로 하는 방법과 질의 결과 내의 각 경로들을 교체 단위로 하는 방법이 있다. 첫번째 방법은 간단한 운용이 가능하지만 효율적이지 못하고 두번째 방법은 첫번째 방법에 비해 효율적이지만 교체 단위 크기의 차이가 커서 캐쉬의 효율을 높이는 데 한계가 있다. 본 논문에서는 위 두 방법의 단점을 해결하기 위해 XML 질의 결과 내의 모든 엘리먼트를 교체 단위로 하는 방법을 제시한다. 이는 교체 단위의 크기가 작고 최대 크기와 최소 크기의 차이도 작으므로 새로 캐쉬에 추가할 데이터의 크기보다 과도하게 큰 희생자가 발생하지 않고, 교체 후 캐쉬 내의 사용하지 않는 공간도 작아지게 되어 캐쉬의 효율성을 크게 향상시킬 수 있는 방법이다. 캐쉬 적중 빈도, 최근 접근 시간, 인출 지연 시간, XML 시맨틱 영역의 크기, XML 시맨틱 영역 내의 엘리먼트 크기 등을 종합적으로 고려하여 교체 희생자를 선택하기 위한 교체 함수를 바탕으로 하는 XML 시맨틱 캐쉬 교체 기법을 제시한다. 본 논문에서 제시한 기법을 적용한 XML 시맨틱 캐쉬 시스템의 프로토타입을 구현하여 실제 LAN 환경에서 실험하였다. 실험 결과 기존의 XML 캐쉬 교체 기법에 비해 본 논문에서 제시한 XML 시맨틱 캐쉬 교체 기법이 더욱 효율적이었다.

### ABSTRACT

In e-business, XML is a major format of data and it is essential to efficiently process queries against XML data. XML query caching has received much attention for query performance improvement. In employing XML query caching, some efficient technique of cache replacement is required. The previous techniques considered as a replacement unit either the whole query result or the path in the query result. The former is simple to employ but it is not efficient whereas the latter is more efficient and yet the size difference among the potential victims is large, and thus, efficiency of caching would be limited. In this paper, we propose a new technique where the element in the query result is a replacement unit to overcome the limitations of the previous techniques. The proposed technique could enhance the cache efficiency to a great extent because it would not pick a victim whose size is too large to store a new cached item, the variance in the size of victims would be


* 본 논문은 정보통신부의 정보통신기초기술연구지원사업(정보통신연구진흥원)으로 수행한 연구 결과입니다.
** 중앙대학교 대학원 컴퓨터공학과
*** 중앙대학교 컴퓨터공학부 교수

small, and the unused space of the cache storage would be small. A technique of XML semantic cache replacement is presented which is based on the replacement function that takes into account cache hit ratio, last access time, fetch time, size of XML semantic region, size of element in XML semantic region, etc. We implemented a prototype XML semantic cache system that employs the proposed technique, and conducted a detailed set of experiments over a LAN environment. The experimental results showed that our proposed technique outperformed the previous ones.

키워드 : 전자 거래, XML, XML 질의처리, XML 시맨틱 캐쉬, 캐쉬 교체, 웹 캐싱
e-business, XML, XML query processing, XML semantic cache, cache replacement, web caching

## 1. 서 론

전자 거래에 있어 XML로 기술된 데이터가 증가하고, 이로부터 효율적인 질의 처리를 수행하는 기능의 중요성이 커지고 있다. 전자 거래와 같은 XML 데이터베이스 기반 웹 응용(XML database-backed Web application)을 지원하기 위한 다층 구조(multi-tier architecture)에서 더 좋은 질의 처리 성능의 기준으로 XML 데이터베이스 접근 요청에 대한 빠른 응답 시간을 들 수 있다. 이를 위한 대표적 기술이 웹 캐싱으로, 응용에서 자주 사용 될 것이라 판단되는 데이터를 중간층에 캐쉬해 둠으로써 질의 처리 성능을 향상시킬 수 있다.

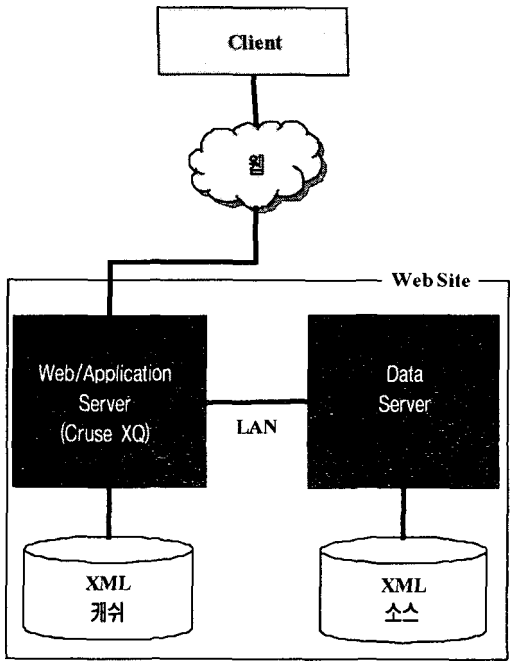
이러한 목적으로 XML 질의 결과의 캐쉬에 관한 연구들이 수행되어왔는데, XCacher[1]는 시맨틱 영역을 메인 메모리에 저장하는 시맨틱 캐쉬 기법을, ACE-XQ[2]는 시맨틱 영역을 XML 문서 형식으로 디스크에 저장하는 시맨틱 캐쉬 기법을 제시하였다. 이들 캐쉬 기법이 실용적으로 활용되기 위해서는 적절한 캐쉬 교체 기법이 필요하다. XCacher에서는 캐쉬된 내용의 하부 트리를 교체 단위로 하고 캐쉬 교체 희생자 선택을 위한 교체 함수로는 LRU를 사용하였다[1]. 이는 기본적인 방법으로서 캐쉬의 효율이 높지 않게 나타났다. ACE-XQ의 캐쉬 교체 기법[3,4]에서는 XQuery 결과가 가질 수 있는 모든 경로를 교체 단위로 하였다. 그리고 각 경로의 적중 빈도, 인출 지연 시간, 객체 크기, XML 문서 크기를 교체 함수에서 참조하였다. 따라서 XQuery 결과 전체를 교체 단위로 하는 것 보다 좋은 효율을 보인다. 하지만 교체 단위 크기의 편차가 커서 캐쉬의 효율을 높이는 데 한계가 있다.

본 논문은 XML 질의어로 표현되는 질의 결과인 XML 시맨틱 영역을 캐쉬하는 XML 시맨틱 캐쉬의 효율적인 교체 기법에 관한 것이다. XML 시맨틱 캐쉬의 효율적인 교체를 위하여 XML 시맨틱 영역 내의 모든 엘리먼트를 교체 단위로 하는 새로운 교체 기법을 제시한다. 이는 XQuery 결과 전체를 교체 단위로 할 경우와 XQuery 결과 내의 모든 경로를 교체 단위로 할 경우의 문제점을 모두 해결할 수 있는 기법이다. 제시하는 기법의 장점은, 캐쉬에 새로 추가할 데이터의 크기보다 과도하게 큰 희생자가 발생하지 않고, 교체 후 캐쉬 내에 미사용 공간이 작기 때문에 캐쉬의 효율을 높일 수 있다는 것이다.

캐쉬 교체를 위해서는 희생자 선택을 위한 교체 함수가 필요하다. 기존의 일반적인 웹 캐쉬 교체 기법에는 단순히 하나 이상의 요소(객체의 최근 접근 시간, 사용 빈도, 크기 등)를 사용해서 교체 희생자를 선택하는 키 기반 정책이 있고, 여러 요소를 조화롭게 결합하여 교체 희생자를 선택하는 함수 기반 정책이 있다. 키 기반 정책으로는 LRU, LRU-MIN[6], LRU-threshold[6], LOG2-SIZE[6], Hyper-G[7]가 있고, 함수 기반 정책으로는 GDS[8], PGDS[9], GD*[10], SLRU[11], LRV[12], LNC-R-W3[13], LUV[14], Hybrid[15], Mix[16]가 있다. 이들 캐쉬 교체 기법은 객체를 사용한 시간, 객체의 크기, 객체의 사용 빈도, 인출 지연 시간, 캐쉬 여유

공간 등을 고려한 것이다. 한편, XML 질의 캐쉬 교체 기법으로 ACE-XQ[3, 4]에서는 XQuery 결과가 가질 수 있는 모든 경로를 교체 단위로 할 경우에 적합하도록 객체의 사용 빈도, 인출 지연 시간, 각 경로의 크기, XQuery 결과의 크기를 고려한 교체 함수를 제시하였다. 본 논문에서는 XML 질의 캐쉬 교체에서 엘리먼트를 교체 단위로 할 경우의 새로운 교체 함수도 제시한다. 본 논문에서 제시한 엘리먼트 교체 기법에서는 캐쉬 효율을 향상시키기 위하여 XML 시맨틱 영역의 크기와 영역 내의 엘리먼트 크기를 모두 고려한다. 즉, XML 시맨틱 영역 내의 모든 엘리먼트를 교체 대상으로 하는 XML 시맨틱 캐쉬 교체 기법에 적합하도록 적중 빈도, 최근 접근 시간, 인출 지연 시간, XML 시맨틱 영역의 크기, XML 시맨틱 영역 내의 엘리먼트 크기를 이용하여 희생자를 선택하는 교체 함수를 사용하는 것이다. 이는 캐쉬 교체 단위인 각 엘리먼트의 정보뿐만 아니라 그 엘리먼트가 속한 XML 시맨틱 영역의 크기 정보를 이용함으로써 더 높은 캐쉬 효율을 얻기 위한 것이다. 본 논문에서 제시하는 캐쉬 교체 함수는 LRFF-LS라 명명하였다.

본 논문에서는 제시한 XML 시맨틱 영역 내의 엘리먼트를 캐쉬 교체 단위로 사용하고, 이에 적합한 교체 함수를 사용하는 엘리먼트 교체 기법을 적용시킨 XML 시맨틱 캐쉬 프로토타입 시스템을 설계 및 구현하였다. 이 시스템의 핵심 모듈은 캐쉬 교체 기능을 담당하는 것으로 CruseXQ라고 명명하였으며 응용 서버 및 웹 서버와 연동한




〈그림 1〉 LAN 환경에서 동작하도록 CruseXQ가 포함된 다층 구조

다. 구현된 프로토타입 시스템은 실세계 수준의 성능 평가를 위해 실제 LAN 환경에서 동작하도록 하였다. CruseXQ가 동작하는 환경은 간략히 나타내면 〈그림 1〉과 같다. 이 시스템을 기반으로 실험을 통해 ACE-XQ[3, 4]의 교체 기법인 전체 교체(Total Replacement)와 부분 교체(Partial Replacement)를 본 논문에서 제시한 엘리먼트 교체 기법과 비교 평가하였다.

본 논문에서 XML 시맨틱 캐쉬의 효율적인 교체를 연구함으로써 기여하는 바는 다음과 같다.

- XML 질의 결과를 캐쉬하는 시스템에서 질의 처리의 성능을 향상(캐쉬 적중률 향상 및 질의 처리 시간 감소)시

킬 수 있는 교체 기법으로 XML 시맨틱 영역 내의 엘리먼트를 교체 단위로 사용하고, 적절한 정보를 이용하여 캐쉬 교체를 수행하는 엘리먼트 교체 기법을 제시하였다.

- 제시된 기법을 적용한 CruseXQ를 제시하고, 이 모듈을 적용한 프로토타입 시스템을 구현하여 실제 LAN 환경에서 성능 평가 실험을 수행하였다.
- 제시한 기법은 전자 거래와 같은 실제 웹 응용 환경에서 향상된 질의 처리 성능을 보이는 XML 캐쉬 시스템에 활용될 수 있다.

본 논문의 구성은 다음과 같다. 2절에서는 관련 연구를 살펴보고, 본 논문과의 차이점을 기술한다. 3절에서는 XML 시맨틱 캐쉬 기법의 바탕을 이루는 XML 시맨틱 영역에 대해서 설명한다. 4절에서는 새로운 XML 시맨틱 캐쉬 교체 기법으로서 엘리먼트 교체 기법을 제시하고, 기존 연구에서 제시한 교체 기법들과 비교한다. 5절에서는 CruseXQ의 구조와 알고리즘을 설명하고, 이를 적용한 프로토타입 시스템의 설계 및 구현에 대해 기술한 후, 구현된 시스템으로 실제 LAN 환경에서 본 논문에서 제시한 기법과 기존 기법과의 성능을 실험을 통해 비교 평가한 결과를 기술한다. 마지막으로 6절에서 결론을 맺고 향후 연구 내용을 기술한다.

## 2. 관련 연구

시맨틱 영역이란 XML 데이터에 대한 질의 결과이다. [17]에서는 시맨틱 영역에 대한 캐쉬 교체 방법에 관한 연구를 수행하였고, XML 질의 캐쉬 교체 기법에 관해서는 [1,3,4]에서 연구 되었다. 또한 XML 시맨틱 캐쉬 교체 기법에서 희생자를 선택하기 위한 교체 함수가 필요하고, 교체 함수로는 객체의 사용 빈도, 사용 시간, 크기를 이용하는 기존의 웹 캐쉬 교체 함수 또는 [3, 4]에서 제시한 XML 캐쉬 교체 함수 등이 있다. 대표적인 웹 캐쉬 교체 함수는 [6-16, 18]에서 연구되었고, [19]에서는 웹 캐쉬에서 교체 함수에 관한 비교 연구를 수행하였다.

[1]의 기법에서는 XQuery 결과에서 캐쉬에 추가될 내용을 선택하여 추가하고 캐쉬의 내용을 나타내는 정의를 재구성한다. 재구성된 정의 내에서 하부 트리로 나타내어지는 부분을 캐쉬 교체 단위로 하고, 희생자를 선택하기 위한 교체함수로는 LRU를 사용하였다. 이 방법에서 교체 단위의 개수는 적고 크기는 크다. 따라서 희생될 교체 단위의 크기가 추가될 XQuery 결과 크기보다 현저히 큰 경우가 자주 발생하고, 이로 인하여 캐쉬의 활용도가 떨어져서 성능 저하의 원인이 된다.

[3, 4]에서는 캐쉬 교체의 단위로 XQuery 결과가 가질 수 있는 모든 경로를 사용하고, 적절한 교체 희생자를 선택하기 위하여 각 경로의 사용 빈도, 인출 지연 시간, 크기,

XQuery 결과의 크기를 사용하였다. 이는 XQuery 결과 전체를 교체 단위로 하는 것에 비해 교체 단위의 개수가 많고, 크기가 작아서 높은 적중률을 나타낸다. 그러나 교체 단위마다 크기의 차이가 크고, 가장 큰 크기의 교체 단위 크기와 XQuery 결과가 가지는 크기의 차이가 크지 않을 경우가 있다. 예를 들어, XQuery 결과 내의 경로 중 하나의 경로가 XQuery 결과의 대부분을 차지할 수도 있다. 이 경로가 희생자로 선택될 경우 XQuery 결과 전체를 희생자로 선택하면서 나타나는 문제와 동일한 문제가 발생할 수 있다.

XML 캐쉬의 효율적인 교체를 위해 교체 희생자의 적절한 선택이 필요하다. [3, 4]에서는 XQuery 결과가 가질 수 있는 모든 경로를 교체 단위로 할 경우 효율적인 교체 함수를 사용하였다. 기존의 웹 캐쉬 교체 함수도 XML 캐쉬 교체에 적용할 수 있다. [19]에서는 기존의 각 교체 함수의 성능을 비교하였다. 간단한 것으로 웹 캐쉬 교체 함수에서 가장 많이 쓰는 정책으로는 가장 최근에 사용한 객체를 보호하고 가장 오래전에 사용한 객체를 희생자로 선택하는 LRU가 있고, 가장 빈번하게 사용하는 객체를 보호하고 가장 적게 사용한 객체를 희생자로 선택하는 LFU(Least Frequently Used)가 있다. 그리고 캐쉬에서 크기가 큰 객체를 가장 먼저 희생자로 선택하는 SIZE[7] 방법이 있다. 그러나 사용 시간, 사용 빈도, 객체 크기 각각의 요소 하나만으로는 높은 효율을 보장하는 교체 희생자 선택이 어렵다. 이 세가지 요소 이외에 몇몇 다른 요소(인출 지연 시간, 캐쉬 여유 공간 등)들을 적절하게 혼합하여 사용함으로써 캐쉬의 효율을 높일 수 있는 교체 희생자 선택이 가능하다. 여기에는 단순히 하나 이상의 요소를 사용해서 교체 희생자를 선택하는 키 기반 정책이 있고, 여러 요소를 조화롭게 결합하여 교체 희생자를 선택하는 함수 기반 정책이 있다.

키 기반 정책으로는 LRU-MIN[6], LRU-threshold[6], LOG2-SIZE[6], Hyper-G[7]가 있다. LRU-MIN, LRU-threshold, LOG2-SIZE 방법은 LRU 방법에 객체 크기의 요소를 고려한 것이다. Hyper-G는 캐쉬에서 객체의 적중 빈도와 최근 접근 시간을 고려한 것이다.

함수 기반 정책으로는 GreedyDual Algorithm[18], Size Adjusted LRU (SLRU) [11], Least Relative Value (LRV)[12], Least Normalized Cost Replacement for the Web (LNC-R-W3)[13], Least Unified Value (LUV)[14], Hybrid[15], Mix[16] 등이 있다. GreedyDual Algorithm에는 GreedyDual Size (GDS)[8], Popularly-Aware GreedyDual Size (PGDS)[9], GreedyDual* (GD*)[10]가 있다. GDS는 객체를 사용한 시간과 객체의 크기를 고려한 것이다. PGDS는 GDS를 확장한 것이고 GD*는 PGDS를 확장한 것으로 객체를 사용한 시간과 객체의 크기, 객체의 사용 빈도를 고려한 것이다. SLRU는 LRU를 기본으로 하고 객체의 크기를 고려한 것이다. LRV는 광범위한 통계적 해석을 수행하는 함수이고, LNC-R-W3은 LRU-K[20](최근 K번째 참조된 시간에 의해서 가치를 평가하는 방법) 방법을 기준으로 하여 캐쉬

교체와 일관성을 통합한 것이며, LUV는 LRU의 장점과 LFU의 장점을 가지는 방법이고, Mix는 Hybrid 방법을 확장한 것으로 객체에 최근에 접근한 시간의 의미를 추가한 것으로, 이는 객체를 사용한 시간과 객체의 크기, 객체의 사용 빈도를 고려한 것이다. 또한 Hybrid는 접근 대기 시간을 줄이는 데 초점을 둔 것으로 객체의 크기와 객체의 사용 빈도를 고려한 것이다.

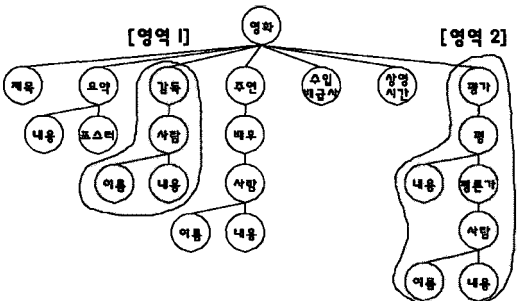
[1,3,4]에서 나타나는 문제를 해결하기 위해 본 논문에서 XML 질의 결과 내의 모든 엘리먼트를 교체 단위로 하고, 이에 적합한 교체 함수를 사용하는 교체 기법을 제시한다. XML 질의 결과 내의 모든 경로를 교체 단위로 할 때보다 XML 질의 결과 내의 모든 엘리먼트를 교체 단위로 하면서 교체 단위의 개수는 더욱 많아지고 교체 단위의 크기는 더욱 작아지게 된다. 그리고 교체 단위 크기의 차이가 작아서 캐쉬에 새로 추가할 데이터의 크기보다 과도하게 큰 교체 단위를 희생자로 선택하는 경우를 피한다. 또한 기존의 여러 가지 웹 캐쉬 교체 함수 및 [3,4]의 교체 함수는 XML 질의 결과 내의 모든 엘리먼트를 교체 단위로 하는 엘리먼트 교체의 특징을 모두 반영하지 못한다. 기존의 웹 캐쉬 교체 함수를 적용시키기 위해서는 엘리먼트를 사용한 시간과 사용 빈도, 인출 지연 시간 외에 엘리먼트의 크기만을 사용할 수 있다. 그러나 XML 시맨틱 영역의 크기를 교체 함수에서 고려하면 더 좋은 효율을 얻을 수 있다. 왜냐하면 희생자 엘리먼트를 선택할 때 선택된 엘리먼트가 자주 사용될 수 있는 XML 시맨틱 영역에 포함되는 경우를 줄여주기 때문이다. 따라서 본 논문에서는 XML 시맨틱 영역의 크기와 영역 내의 엘리먼트 크기를 모두 적용하는 XML 시맨틱 캐쉬 교체 함수인 LRFF-LS를 제시한다. LRFF-LS은 객체 사용 빈도, 최근 접근 시간, 인출 지연 시간, XML 시맨틱 영역의 크기, XML 시맨틱 영역 내의 엘리먼트 크기를 사용하여 희생자를 선택하기 위한 것이다.

## 3. XML 시맨틱 영역

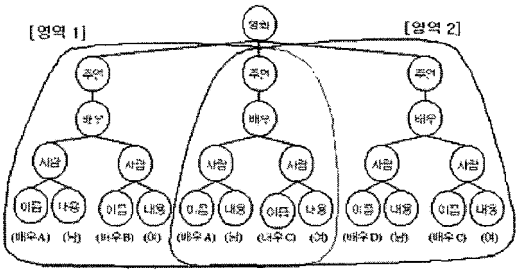
XML 시맨틱 영역이란 XPath 등의 XML 질의어로 표현되는 질의 결과의 하위 경로까지 모두 포함한 영역을 말한다. 본 논문에서 캐쉬에 저장되는 단위는 XML 시맨틱 영역이다. <그림 2>와 <그림 3>은 XML 시맨틱 영역의 예이다. 여기서 사용된 문서는 영화 문서로 <그림 2>에서는 경로에 따른 XML 시맨틱 영역을 보여준다. 영역 1은 '/영화/감독'으로 정의된 XML 시맨틱 영역이고, 영역 2는 '/영화/평가'로 정의된 XML 시맨틱 영역이다. 각 영역은 XML 시맨틱 영역 정의의 하위 경로(subtree)를 포함한다. <그림 3>은 인스턴스에 따른 XML 시맨틱 영역을 보여준다. 영역 1은 '/영화/주연[/배우/사람/이름="배우 A"]'로 정의된 XML 시맨틱 영역이고, 영역 2는 '/영화/주연[/배우/사람/이름="배우 C"]'로 정의된 XML 시맨틱 영역이다. 각 XML 시맨틱 영역은 하위 경로가 같더라도 최종 노드의 인스턴스에 따라 서로 다른 영역이 된다. 그리고 서

로 다르게 정의된 XML 시맨틱 영역이라도 〈그림 3〉의 경우처럼 중복되는 영역이 발생할 수 있다.

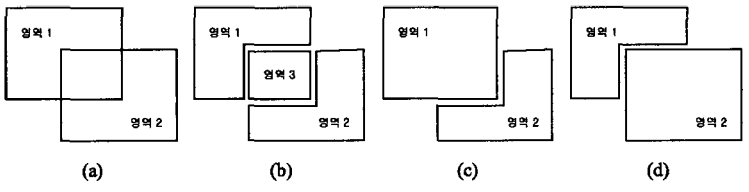
XML 시맨틱 캐쉬에 이미 존재하는 XML 시맨틱 영역 이외에 캐쉬 미스되어 하부 소스로부터 제공되는 XML 시맨틱 영역은 XML 시맨틱 캐쉬에서 재사용할 수 있도록 추가된다. 이때 추가될 XML 시맨틱 영역은 기존의 XML 시맨틱 영역과 겹쳐지는 부분이 있을 수 있고 전혀 다른 XML 시맨틱 영역일 수도 있다. 전혀 다른 XML 시맨틱 영역이 추가될 경우에는 기존의 XML 시맨틱 영역에 추가될 XML 시맨틱 영역을 추가하여 최종 결과를 만든다. 그러나 추가될 XML 시맨틱 영역이 기존의 XML 시맨틱 영역과 겹치는 경우에는 겹치는 부분을 처리해야 한다. 즉, 겹치는 부분을 어느 XML 시맨틱 영역에 포함시켜야 하는지의 여부에 따라 XML 시맨틱 영역의 분할 방법이 분류된다. 〈그림 4〉는 세가지 XML 시맨틱 영역 분할 방법을 보여준다. 〈그림 4〉의 (a)는 기존에 영역 1이 존재할 때 영역 2가 추가되는 상황을 나타낸다. 〈그림 4〉의 (b)는 추가될 XML 시맨틱 영역과 겹쳐지는 영역을 독립적인 영역으로 하는 방법으로, 새로운 영역이 추가되면 겹쳐지는 영역은 영역 3이 되고, 나머지 영역은 영역 2가 된다. 〈그림 4〉의 (c)는 추가될 XML 시




(경로에 따른 영역)
〈그림 2〉 XML 시맨틱 영역의 예




(인스턴스에 따른 영역)
〈그림 3〉 XML 시맨틱 영역의 예

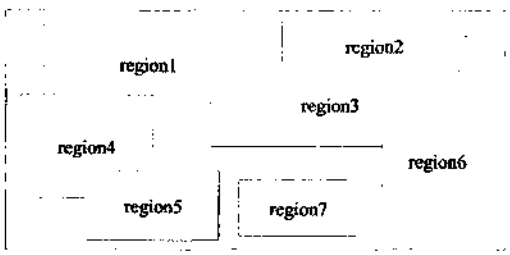



〈그림 4〉 XML 시맨틱 영역 추가 시 분할 방법

맨틱 영역에서 기존 XML 시맨틱 영역과 겹치지 않는 영역만을 추가하는 방법으로, 새로운 영역이 추가되면 겹치지 않는 부분만 영역 2가 된다. 〈그림 4〉의 (d)는 기존 XML 시맨틱 영역에서 추가될 XML 시맨틱 영역과 겹치는 영역을 제거하고 추가될 XML 시맨틱 영역은 원형을 추가하는 방법으로, 새로운 영역이 추가되면 추가되는 부분이 영역 2가 되고, 기존 영역 1에서 겹치는 부분을 제외한 영역이 영역 1이 된다.

## 4. XML 시맨틱 캐쉬 교체

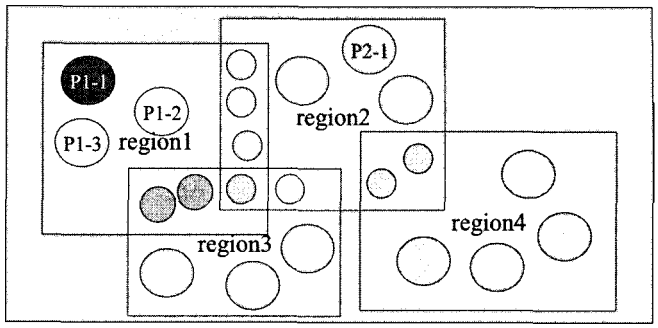
XML 시맨틱 캐쉬의 효율적인 교체를 위해서는 어떠한 교체 단위를 사용할 것이고, 그에 따라 어떠한 교체 함수를 적용할 것인지를 고려해야 한다. 교체 단위로는 기존 연구[3,4]에서 XQuery 결과 전체를 캐쉬 교체 단위로 사용하는 전체 교체(Total Replacement, TR)와 XQuery 결과가 가지는 모든 경로를 캐쉬 교체 단위로 사용하는 부분 교체(Partial Replacement)가 제시되었다. 이들 중 부분 교체는 XML 질의 결과의 구분 가능한 경로를 교체 단위로 하는 것이므로 경로 교체(Path Replacement)라고 명명해도 된다. 본 절에서는 기존 기법의 문제점을 해결하기 위해 엘리먼트 교체(Element Replacement, ER)를 제시한다. 그리고 엘리먼트 교체를 사용할 경우 캐쉬된 데이터의 적절한 통계치를 이용한 교체 함수로 LRFF-LS를 제시한다.




〈그림 5〉 전체 교체에서 교체 단위의 예

### 4.1. 전체 교체와 경로 교체

전체 교체는 XQuery 결과 전체를 캐쉬의 교체 단위로 하는 것이다. 이 방법은 교체 단위의 개수가 적고 그 크기가 크다. 따라서 교체 희생자를 선택하기 위해 모든 교체 단위에서 교체 희생자 선택을 위한 교체 함수를 적용해야 하는데, 교체 단위의 개수가 적기 때문에 교체 희생자를 선택하기 위한 연산을 짧은 시간에 수행할 수 있다. 〈그림 5〉는 전체 교체에서 교체 단위의 예를 보여준다. Region 1~7은 시맨틱 영역이다. 캐쉬 내에서 교체가 필요할 때 이 영역 중에서 하나를 희생자로 선택한다. 여기서 시맨틱 영역을 추가하려고 하는데 캐쉬의 여유 공간이 부족하다고 하자. 캐쉬 내에서 하나의 시맨틱 영역이 교체될 것이다. 교체될 시맨틱 영역의 크기가 추가될 시맨틱 영역의 크기보다 과도하게 크다면 교체 후 캐쉬 내에는 캐쉬의 미사용 공간이 많이 생기게 된다. 또한 크기가 큰 시맨틱 영역이 교체됨으로써 질의되는 시맨틱 영역과 부분적으로 적중될 수 있는 경우가 적어지게 된다. 즉, 교체 단위의 크기가 다양하고 크므로 캐쉬 공간의




〈그림 6〉 경로 교체에서 교체 단위의 예

활용률이 낮아진다.

전체 교체의 단점을 해결하는 방법으로 XQuery의 질의 결과 내의 모든 경로를 교체 단위로 하는 경로 교체가 제시되었다. 이는 전체 교체와 비교하여 상대적으로 교체 단위의 개수가 많아지고 크기는 작아지게 된다. 〈그림 6〉은 전체 교체에서 교체 단위의 개수가 적고 크기가 크므로 인해서 나타나는 문제점을 해결한 경로 교체에서 교체 단위의 예를 보여준다. Region 1~4는 시맨틱 영역이다. 그리고 시맨틱 영역 내의 원형 객체는 시맨틱 영역 내의 모든 가능한 경로를 나타낸다. 캐쉬 내에서 교체가 필요할 때 각 시맨틱 영역 내의 모든 경로들 중에서 하나의 희생자를 선택한다. 희생자의 크기가 작으므로 교체가 일어나더라도 추가될 시맨틱 영역의 크기보다 과도하게 큰 객체가 교체되지 않아 캐쉬 공간의 활용률을 높이게 된다. 그러나 교체 단위 크기의 차이가 커서 캐쉬의 효율을 높이는 데 한계가 있다. 이러

〈표 1〉 XML 질의 결과가 포함하는 구분 가능한 모든 경로와 크기 정보

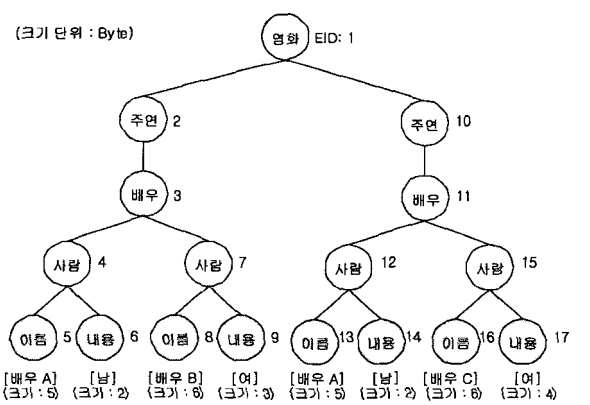
| XML 질의: '/PLAY/ACT/SCENE[/SPEECH/SPEAKER = "CLARENCE"]' | |
|---|---|
| XML 질의 결과의 크기: 165,956 Byte | |
| XML 질의 결과 내의 구분 가능한 경로 | 경로가 포함하는 데이터의 크기 (Byte) |
| /PLAY/ACT/SCENE/TITLE | 922 |
| /PLAY/ACT/SCENE/SPEECH/SPEAKER | 24,018 |
| /PLAY/ACT/SCENE/SPEECH/LINE/STAGEDIR | 1,162 |
| /PLAY/ACT/SCENE/SPEECH/LINE | 133,765 |
| /PLAY/ACT/SCENE/SPEECH/STAGEDIR | 1,364 |
| /PLAY/ACT/SCENE/STAGEDIR | 4,725 |

한 경우가 발생하는 상황을 예를 들어서 살펴보자. 〈표 1〉은 세익스피어 희곡 문서를 원본 데이터로 하는 XML 질의 결과의 구분 가능한 경로 크기를 나타낸 것이다. 이 결과 중에서 가장 크기가 큰 경로의 크기는 133,765 Byte이다. 이 경로가 가지는 크기는 전체 XML 질의 결과의 크기 중 80% 이상을 차지한다. 반면 가장 작은 크기를 가지는 경로의 크기는 922Byte로 XML 질의 결과 중 0.55%에 불과하다. 즉, 경로가 가지는 크기가 최대 145배 정도의 차이가 있다. 이는 캐쉬 회생자를 선택할 때 새로 추가될 데이터의 크기보다 과도하게 큰 회생자가 선택될 수 있다는 것을 보여준다.

### 4.2. 엘리먼트 교체

본 절에서는 전체 교체와 경로 교체의 문제점을 해결하기 위해 XPath 등의 XML 질의어로 표현되는 질의 결과의 XML 시맨틱 영역 내의 모든 엘리먼트를 교체 단위로 하는 교체 기법인 엘리먼트 교체를 제시한다. 〈그림 7〉은 엘리먼트 교체에서 교체 단위의 예를 보여준다. 이는 〈그림 3〉의 영역 1인 '/영화/주연[/배우/사람/이름="배우 A"]'로 정의된 XML 시맨틱 영역이다. 이 XML 시맨틱 영역에서 구분 가능한 경로는 '/영화/주연/배우/사람/이름'과 '/영화/주연/배우/사람/내용' 2개로 나뉘고, 각 경로의 크기는 22Byte, 11Byte가 된다. 이에 반해 XML 시맨틱 영역의 엘리먼트는 트리의 단말 엘리먼트(leaf element)인 'EID(Element ID) 5, 6, 8, 9, 13, 14, 16, 17'의 8개로 나뉘고, 크기 또한 각각 5, 2, 6, 3, 5, 2, 6, 4 Byte가 된다. 즉, 교체 단위의 개수가 늘어나고, 교체 단위의 크기가 상당히 줄어들게 된다. 즉, 필요 이상으로 크기가 큰 교체 단위가 희생되면서 발생하는 효율의 저하를 방지할 수 있다.

〈표 2〉는 세익스피어 희곡 문서를 원본 데이터로 하는 XML 질의 결과가 포함하는 모든 엘리먼트 중 최대 엘리먼트 크기와 최소 엘리먼트 크기를 보여준다. 여기서 엘리먼트의 최대 크기는 164 Byte이고 최소 크기는 35 Byte이다. 이에 반해, XML 시맨틱 영역의 모든 경로를 교체 단위로 할 때 경로의 최대 크기는 〈표 1〉에서 보았듯이 133,765Byte이고 최소 크기는 922Byte이다. 즉, XML 시맨틱 영역 내의 엘리먼트를 교체 단위로 함으로써 교체 단위의 크기가 작을 뿐 아니라 최대 크기와 최소 크기의 차이도 5배 이내가 되므로 필요 이상으로 크기가 큰 희생자가 발생하지 않고, 교체 후에도 캐




〈그림 7〉 엘리먼트 교체에서 교체 단위의 예

〈표 2〉 XML 질의 결과가 포함하는 엘리먼트의 최대/최소 크기

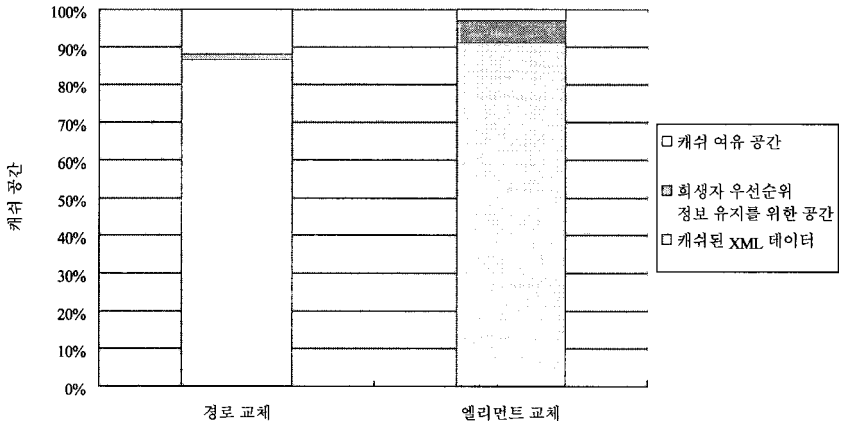
| XML 질의: '/PLAY/ACT/SCENE[/SPEECH/SPEAKER = "CLARENCE"]' | |
|---|---|
| XML 질의 결과의 크기: 165,956 Byte | |
| XML 질의 결과 내의 엘리먼트 최대크기 | 164 Byte |
| XML 질의 결과 내의 엘리먼트 최소크기 | 35 Byte |

〈표 3〉 경로 교체와 엘리먼트 교체의 개수와 평균 크기

| XML 질의: '/PLAY/ACT/SCENE[/SPEECH/SPEAKER = "CLARENCE"]' | | |
|---|---|---|
| XML 질의 결과의 크기: 165956 Byte | | |
| 경로 교체 | 교체 단위 개수 | 6 개 |
| | 교체 단위 평균 크기 | 27,660 Byte |
| 엘리먼트 교체 | 교체 단위 개수 | 1,660 개 |
| | 교체 단위 평균 크기 | 100 Byte |

쉬 내의 미사용 공간도 작아지게 된다.

엘리먼트 교체와 같이 개수가 많고 크기가 작은 교체 단위를 사용하면서 나타나는 부담으로 다음 두가지를 들 수 있다. 첫째는 교체 희생자를 선택하기 위한 연산을 수행하는 시간이 많이 걸린다는 것이다. 예를 들어 경로 교체와 엘리먼트 교체에서 희생자 선택을 위한 시간을 고려하자. 〈표 3〉은 각 교체 기법에서 사용하는 교체 단위의 개수와 평균 크기를 나타낸 것이다. 먼저 경로 교체에서 20,000Byte의 여유 공간을 확보하기 위한 희생자 제거 시간을 계산하면 다음과 같다. 6개의 교체 단위 중 1개를 선택하고, 선택된 희생자 1개만 희생되어도 충분한 여유 공간을 확보할 수 있으므로 이 1개의 희생자를 교체하면 된다. 즉, 6개 중 1개를 선택하고 이를 캐쉬에서 삭제하면 된다. 엘리먼트 교체에서는 좀 더 복잡하다. 20,000 Byte의 여유 공간을 확보하기 위해서는 적어도 200개 이상의 희생자를 선택해야 한다. 그리고 선택해야 하는 대상도 경로 교체에 비해 270배 정도 많다. 즉 1,660개의 교체 대상 중 200개를 찾아야 하는 시간이 경로 교체에 비해 매우 많이 든다. 한편 선택된 희생자를 캐쉬에서 제거하는 시간은 경로 교체에서의 희생자 제거 시간과 비슷하다. 즉, 엘리먼트 교체에서는 많은 교체 단위 중에서 여러 개를 선택해야 하므로 교체 희생자를 찾고 제거하는 데 많은 시간이 걸린다. 그러나 경로 교체보다 엘리먼트 교체에서 교체 희생자를 선택하고 제거하는 시간이 더 많이 걸리지만, 전체 질의 처리 시간에서




(그림 8) 교체 기법 간 캐쉬 공간 활용률 비교

는 엘리먼트 교체가 더 좋은 성능을 보인다. 그 이유는 엘리먼트 교체로 인한 캐쉬 적중률이 경로 교체로 인한 캐쉬 적중률보다 더욱 향상되고, 캐쉬 적중률 향상으로 인한 질의 응답 시간이 단축되는 시간이 캐쉬 교체를 위해 희생자를 선택하고 제거하는 시간보다 훨씬 더 크기 때문이다.

둘째 부담은 엘리먼트 교체를 사용하면서 교체 단위인 각 엘리먼트의 희생자 우선 순위 정보를 유지하기 위하여 캐쉬 저장 공간의 일부를 필요로 한다는 것이다. 경로 교체에서 각 경로의 교체 희생자 우선순위 정보를 유지하기 위해 캐쉬 저장 공간의 1.5%를 사용하지만, 엘리먼트 교체에서는 캐쉬 공간의 6%를 사용한다. 그러나 엘리먼트 교체를 위한 캐쉬 저장 공간의 일부를 부담하여도 전체적인 캐쉬 성능을 향상시킬 수 있다. 그 이유는 〈그림 8〉로 알 수 있다. 〈그림 8〉에서는 캐쉬 크기가 1,000 KB인 XML 캐쉬에서 캐쉬 공간의 활용률을 보여준다. 경로 교체를 사용하였을 때 경로의 희생자 우선 순위 정보를 유지하기 위해 1.5%의 공간을 필요로 하지만 교체를 수행하는 동안 캐쉬 여유 공간은 12% 정도가 된다. 즉, 캐쉬 내에 순수 XML 데이터를 86.5% 포함하고 있다. 반면에 엘리먼트 교체를 사용하면 엘리먼트의 희생자 우선 순위 정보를 유지하기 위해 6%의 공간을 필요로 하지만 교체를 수행하는 동안 캐쉬 여유 공간은 3%에 불과하다. 따라서 캐쉬 내의 순수 XML 데이터를 91% 포함하게 된다. 즉, 엘리먼트 교체를 사용할 경우 캐쉬 활용률이 높다는 것을 알 수 있다.

### 4.3. XML 시맨틱 캐쉬 교체 함수

본 절에서는 엘리먼트를 교체 단위로 하는 XML 시맨틱 캐쉬 교체에서 적절한 희생자를 선택하기 위한 교체 함수를 살펴본다. 2절에서는 교체 희생자를 선택하기 위한 교체 함수로 웹 캐쉬 교체 함수들의 특징과 장단점을 살펴보았다. 웹 캐쉬의 교체에서 희생자를 선택하기 위한 교체 함수로 객체

$$\text{ReplacementFunction} : \frac{\text{HitCount} \times \text{LastAccessTime} \times \text{FetchTime}}{\text{XMLSemanticeRegionSize} \times \text{ElementSize}} \quad \text{식 (1)}$$

의 사용 빈도, 최근 사용 시간, 크기 중에서 몇 가지를 반영하는 여러 가지 방법이 있다.

그러나 XML 질의 결과 내의 모든 엘리먼트를 교체 단위로 사용할 때 XML 시맨틱 영역의 크기와 그 영역 내의 엘리먼트의 크기를 고려 사항으로 사용할 수 있는데, 기존의 웹 캐쉬 교체 함수에서는 이를 반영하는 데 적절하지 못하다. 즉, 기존의 웹 캐쉬 교체 함수를 적용시키기 위해서는 엘리먼트를 사용한 시간과 사용 빈도, 인출 지연 시간 외에 엘리먼트의 크기만을 사용할 수 있다. 그러나 희생자 엘리먼트를 선택할 때 선택된 엘리먼트가 자주 사용될 수 있는 XML 시맨틱 영역에 포함되는 경우를 줄여주기 위해서 XML 시맨틱 영역의 크기를 교체 함수에서 고려하면 더 좋은 효율을 얻을 수 있다. 따라서 본 논문에서는 XML 시맨틱 영역의 크기와 영역 내의 엘리먼트 크기를 모두 적용하는 XML 시맨틱 캐쉬 교체 함수인 LRFF-LS(Least Recently Frequently Fetch-Large Size) 교체 함수를 제시한다. 이 교체 함수는 적중 빈도(Hit Count), 인출 지연 시간(Fetch Time), 최근 접근 시간(Last Access Time), XML 시맨틱 영역 크기(XML Semantic Region Size), XML 시맨틱 영역 내의 엘리먼트 크기(Element Size)를 고려해서 희생 우선 순위를 결정한다. 여기서 적중 빈도란 캐쉬의 XML 시맨틱 영역 또는 XML 시맨틱 영역 내의 모든 엘리먼트가 얼마나 많이 재사용되었는지를 나타낸다 (XML 질의 결과 중 재사용된 엘리먼트의 개수 / XML 질의 결과 전체가 포함하는 엘리먼트의 개수). 인출 지연시간이란 캐쉬에 적중되지 않는 데이터가 하부 XML 데이터에 질의를 요청하는 시점부터 그 결과가 캐쉬에 도달하는 데 까지 걸리는 시간이다 (데이터 서버에서의 질의 수행 시간 + 네트워크 전송 시간). 최근 접근 시간은 캐쉬의 객체에 가장 마지막으로 사용했을 때의 시간이고, 객체 크기는 캐쉬의 XML 시맨틱 영역의 크기와 각 엘리먼트의 크기를 말한다.

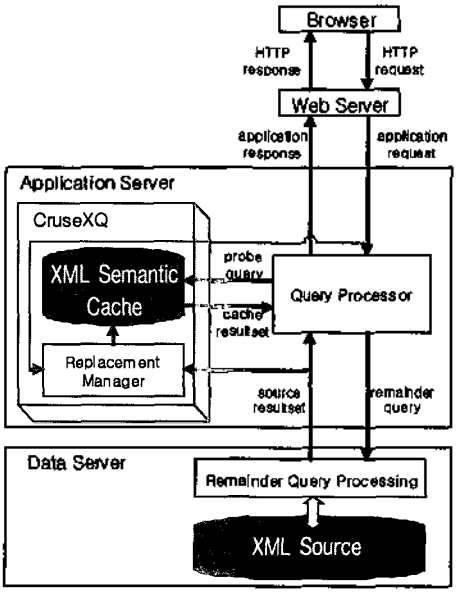
LRFF-LS는 교체 희생자 우선 순위를 설정하기 위해 식 (1)을 사용한다. HitCount는 캐쉬에 해당 객체의 적중 빈도를 의미하며 캐쉬에서 해당 캐쉬가 제거되면 HitCount는 '0'이 된다. LastAccessTime은 최근 접근 시간이며, 한 객체가 HitCount가 높지만 최근 접근 시간이 오래되었을 경우 희생자로 선택되기 위하여 사용한다. LastAccessTime은 현재 시간에서 현재 시간과 최근 접근 시간의 차이를 나눈 값으로 최근 접근 시간이 오래 되지 않을수록 이 값은 커진다. FetchTime은 인출 지연 시간으로 1/1,000초 단위로 측정한 값이다.

XMLSemanticRegionSize는 하부 XML 데이터에서 질의 응답된 XML 시맨틱 영역의 크기를 제곱근한 값이며, ElementSize는 XML 시맨틱 영역 내의 각각의 엘리먼트의 크기이다. 즉, 이는 적중 빈도가 낮고 사용

한지 오래된 객체는 다음에 재사용될 확률이 낮아 빨리 희생자로 선택된다. 그리고 인출 지연 시간이 큰 객체 (예: 하부 XML 데이터에서 질의 수행 시간이 많이 걸리는 객체)를 먼저 희생자로 선택하지 않게 해서 전체 질의 처리 시간을 줄인다. 또한 XML 시맨틱 영역의 크기가 크고, 그 중에서도 XML 시맨틱 영역 내의 엘리먼트 크기가 큰 객체를 먼저 희생자로 선택하게 하여 크기가 작은 객체의 많은 수를 캐쉬에 오래

```
int[i] selectVictim(Cached_Object co, int required_size) {
    int[i] victim;
    int victim_size = 0;
    object[i] victim_order;
    victim_order ← ReplacementFunction(co);   // 식 (1) 참조
    for i=0부터 캐쉬 여유 공간이 확보될 때 까지 i는 1씩 증가 do
        victim[i] = victim_order[i].element_id;
        victim_size =+ victim_order[i].size;
        if victim_size ≥ required_size then
            break;
        end if
    end for
    return victim;
}
```

〈그림 9〉 LRFF-LS 교체 함수를 이용하는 희생자 선택 알고리즘

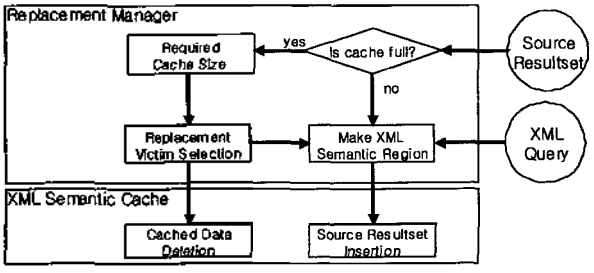



(그림 10) CruseXQ를 적용한 XML 시맨틱 캐쉬 시스템의 구조

남기도록 한다.

본 논문에서 제시한 LRFF-LS 교체 함수와 기존의 웹 캐쉬 교체 함수와의 차이점은 교체 단위의 크기를 고려한 것뿐만 아니라 교체 단위가 속한 XML 시맨틱 영역의 크기까지 고려한 것이다. 즉, 교체 단위가 속한 영역을 고려하여 상대적으로 크기가 큰 영역의 객체를 희생자로 선택한다. 따라서 크기가 작은 XML 시맨틱 영역의 데이터를 캐쉬에 오래 남겨 두어 데이터 간의 관련성을 높인다.

<그림 9>는 LRFF-LS 교체 함수를 이용하여 여러 희생자를 선택하는 알고리즘이다. 캐쉬된 객체 '∞'를 교체 함수에 적용하여 희생자 우선순위 'victim_order'를 만든다. 'victim_order'에서 희생자 우선 순위를 따라 하나씩 희생자의 식별(victim_order.element_id)를 희생자 배열(victim)에 추가하고, 각각의 희생자의 크기(victim_order.size)를 전체 희생자 크기(victim_size)에 누적한다. 그리고 캐쉬에 요구되는 크기(required_size)와 전체 희생자의 크기를 비교하여 전체 희생자의 크기가 캐쉬에 요구되는 크기보다 크면 희생자 선택을 중단하고 선택된 희생자 배열을 반환하게 된다.




(그림 11) CruseXQ의 내부 구조

## 5. 구현 및 성능 평가

본 절에서는 본 논문에서 제시한 기법을 구현한 CruseXQ와 이를 적용한 XML 시맨틱 캐쉬 시스템 프로토타입의 구현과 이를 실제 LAN 환경에서 적용하여 본 논문에서 제시한 교체 기법의 성능을 기존의 기법들과 실험을 통하여 비교 평가한 결과를 기술한다.

### 5.1. CruseXQ

<그림 10>은 CruseXQ가 적용된 XML 시맨틱 캐쉬 시스템의 구조도이다. 응용 서버(Application Server)에 XML 질의가 요청되면 XML 질의는 CruseXQ의 교체 관리자(Replacement Manager)에 전달된다. 질의 처리기(Query Processor)는 XML 질의를 주 질의(probe query)와 잔여 질의(remainder query)로 구분하고, 주 질의는 캐쉬에서 처리하고 잔여 질의는 데이터 서버(Data Server)에서 처리하여 사용자에게 질의의 결과를 전달한다. 이때 잔여 질의의 결과인 소스 결과셋(source resultset)은 CruseXQ의 교체 관리자에 전달되고, 이전에 전달받은 XML 질의 정보와 소스 결과셋을 조합하여 캐쉬에 추가하게 된다.

<그림 11>은 CruseXQ의 내부 구조를 보여준다. XML 질의가 요청되면 XML 질의는

CruseXQ에서 XML 시맨틱 영역을 만드는데 사용되고, 잔여 질의의 결과인 소스 결과셋은 교체 관리자로 전달된다. 교체 관리자는 소스 결과셋이 캐쉬에 추가될 수 있는지 판단한다. 캐쉬의 여유 공간이 충분하면 소스 결과셋과 XML 질의를 이용하여 XML 시맨틱 영역의 정의와 내용을 XML 시맨틱 캐쉬(XML Semantic Cache)에 추가한다. 캐쉬의 공간이 충분하지 않으면 캐쉬에서 희생되어야 할 크기를 결정하고, 교체 함수와 교체 기법에 따라 교체 희생자를 선택하여 XML 시맨틱 캐쉬에서 희생자를 삭제한다. 그리고 소스 결과셋과 XML 질의를 이용하여 XML 시맨틱 영역의 정의와 내용을 XML 시맨틱 캐쉬에 추가한다.

### 5.2. 구현 환경 및 실험 데이터

CruseXQ를 적용한 프로토타입 시스템에서는 소스 데이터 및 캐쉬된 데이터를 RDBMS에 저장한다. 그리고 XML 캐쉬를 사용함으로써 네트워크 비용의 절감 효과를 측정하기 위해 데이터 서버와 응용 서버를 실제 LAN 환경에서 작동하도록 설정하였다. <그림 1>은 이러한 환경을 나타낸 것이다. CruseXQ를 적용한 프로토타입 시스템에서의 실험 목적은 XML 시맨틱 캐쉬 교체 시스템에서 적중률과 캐쉬 교체 연산 및 교체 수행의 부담과 평균 질의 처리 시간을 측정하는 것이다. 구현 및 실험 환경은 다음과 같다. 데이터 서버는 CPU는 Intel Celeron 2.4 GHz, Memory는 2GB, OS는 Windows 2000 Server, 개발 도구는 Java 1.4.1, 데이터베이스는 Oracle 10g를 사용하였으며, 응용 서버는 CPU는 Intel Pentium4 3.6 GHz이고 나머지는 데이터 서버의 환경과 같다. 본 실험에서 사용한 데이터는 "세익스피어 희곡" 문서 37개[5] (문서들의 총 엘리먼트 개수 179689개, 문서들의 총 바이트 크기 7.57 MB)이다. XML 질의는 XPath로 표현되었다.

### 5.3. 성능 평가 실험

본 실험에서는 실제 LAN 환경에서 전체 교체, 경로 교체 및 엘리먼트 교체를 포함하는 XML 시맨틱 캐쉬 교체 기법 간의 성능을 비교 평가하였다. 즉, ACE-XQ[3, 4]의 교체 함수를 이용하는 전체 교체(TR/ACEXQ) 및 경로 교체(PR/ACEXQ)와 LRFF-LS 교체 함수를 이용하는 본 논문의 엘리먼트 교체(ER/LRFF-LS) 3자 간의 성능을 비교하였다. 성능 평가의 척도로는 Hit Count Ratio (HCR), Hit Byte Ratio(HBR), 캐쉬 교체 수행 시 희생자가 제거되는 시간(Victim Selection and Purge Time, VSPT), 평균 질의 응답 시간(Average Query Response Time, AQRT)을 사용하였다. HCR는 요청되는 XML 질의의 모든 엘리먼트들 중에서 캐쉬에서 응답되는 엘리먼트의 비율을 나타낸 것이다. HBR는 요청되는 XML 질의 결과 전체 크기 중에서 캐쉬에서 응답되는 부분의 크기의 비율을 나타낸 것이다. VSPT는 캐쉬 교체가 발생할 때 충분한 캐쉬 저장 공간을 확보하기 위해 교체 희생자를 찾는데 소요되는 시간과 교체 희생자를 삭제하는데 소요되는 시간이다. 그리고 AQRT는

〈표 4〉 성능 척도의 수식

| 수식 |
|---|
| $HitCountRatio(HCR) = \frac{1}{n}\sum_{i=1}^{n}\frac{C_i \cdot elemnum}{Q_i \cdot elemnum}$ |
| $HitByteRatio(HBR) = \frac{1}{n}\sum_{i=1}^{n}\frac{C_i \cdot byte}{Q_i \cdot byte}$ |
| $VictimSelectAndPurgeTime(VSPT) = \frac{1}{r}\sum_{i=1}^{n} C_i \cdot ReplacementTime$ |
| $AverageQueryResponseTime(AQRT) = \frac{1}{r}\sum_{i=1}^{n} QueryResponseTime$ |

응용 서버에 수많은 질의가 요청되고 XML 캐쉬에서 응답되거나 데이터 서버에서 응답되어 질의 결과가 최종 응답 될 때까지의 시간을 평균값으로 측정한 결과이다.

〈표 4〉는 위의 다섯가지 결과 값을 얻기 위한 수식을 나타낸다. 여기서 'Qi'는 i번째 XML 질의를 나타내고, 'Ci'는 i번째 XML 질의 중 캐쉬에서 응답되는 부분을 의미한다. 성능 평가에서 XML 질의의 총 수행 횟수는 'n'이고, XML 질의 수행 중 캐쉬 교체가 발생한 횟수는 'r'이다. 'elemnum'은 단말 엘리먼트의 개수, 'byte'는 단말 엘리먼트의 크기, 'ReplacementTime'은 교체를 수행하는 시간(희생자 선택 및 제거)을 의미한다. 또한 'QueryResponseTime'은 응용 서버에 XML 질의가 주어지고 XML 질의 결과가 최종 응답될 때까지의 시간으로, 질의에 따라 캐쉬에서 응답되거나 데이터 서버에서 응답될 수 있다.

5.3.1. 실험 Ⅰ

본 실험에 사용된 XML 질의는 〈표 5〉의 10가지이고, 에 나타내었다. 이들 질의는 3개의 트리와 그 서브 트리로 이루어져 있다. Q1, Q2, Q3은 서로 각각의 트리이다. 그리고 Q4, Q7, Q10은 Q1의 서브 트리이다(Q1-Q4-Q7-Q10). Q5, Q8은 Q2의 서브 트리이다(Q2-Q5-Q8). Q6, Q9는 Q3의 서브 트리이다(Q3-Q6-Q9). 즉 Q1~Q10 순서로 질의가 수행되면 Q1~Q3은 하부 데이터에서 결과를 얻고 그 결과를 캐쉬에 저장한다. 그 후 Q4~Q10은 캐쉬에 저장된 Q1~Q3에 완전 포함(contained query)되므로 캐쉬에서 모두 응답된다. 반대로 Q10~Q1 순서로 질의가 수행되면 처음 Q10~Q8은 하부 데이터의 값을 얻어 캐쉬에 저장하고, 그 후 Q7~Q1은 이전 질의에 부분적으로 중복(overlap query)되므로 일부 캐쉬의 데이터를 사용한다.

● 완전 포함 질의

〈그림 12〉~〈그림 15〉는 XML 질의를 Q1~Q10 순서대로 수행하였을 때(완전 포함 질의 수행)의 HCR, HBR, VSPT, AQRT

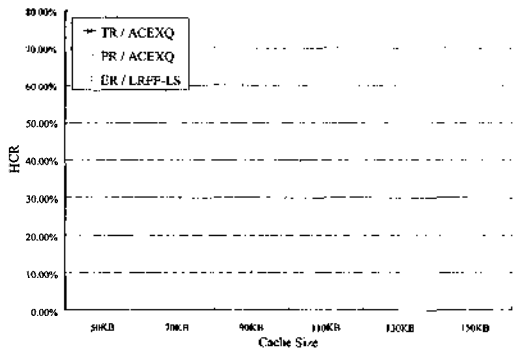
〈표 5〉 성능 실험 I의 XML 질의

| | XML 질의 종류 | 결과 크기 (Byte) |
|---|---|---|
| Q1 | /PLAY/INDUCT | 28,008 |
| Q2 | /PLAY/PERSONAE | 53,636 |
| Q3 | /PLAT/ACT/PROLOGUE | 35,141 |
| Q4 | /PLAY/INDUCT/SCENE | 25,009 |
| Q5 | /PLAY/PERSONAE/PGROUP | 15,113 |
| Q6 | /PLAT/ACT/PROLOGUE/SPEECH | 33,322 |
| Q7 | /PLAY/INDUCT/SCENE/SPEECH | 24,281 |
| Q8 | /PLAY/PERSONAE/PGROUP/GRPDESCR | 4,322 |
| Q9 | /PLAT/ACT/PROLOGUE/SPEECH/LINE | 31,679 |
| Q10 | /PLAY/INDUCT/SCENE/SPEECH/LINE | 19,480 |

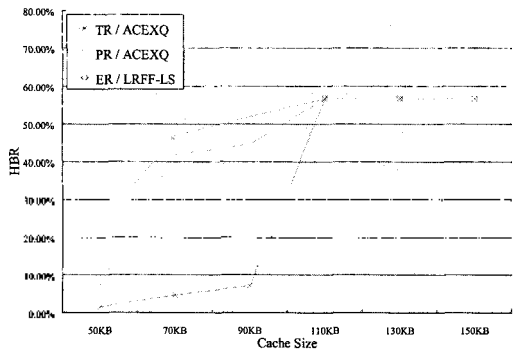
을 나타낸 것이다. 캐쉬 공간의 크기가 90 KB 이하에서 XML 질의가 수행되면서 교체가 발생한다. 이 경우 전체 교체 성능이 상당히 낮은 것을 확인할 수 있다. 이는 처음에 캐쉬에 저장되는 XML 시맨틱 영역의 크기가 크기 때문이다. 캐쉬의 크기가 110K 이상이면 교체가 발생하지 않음으로 가장 좋은 성능을 보이면서 교체 기법 간의 성능 차이는 없다.

● 부분 중복 질의

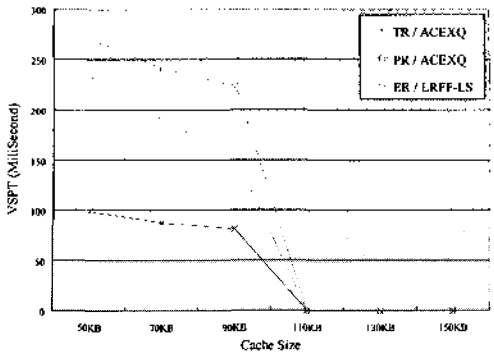
〈그림 16〉~〈그림 19〉는 XML 질의를 Q10~Q1 순서대로 수행하였을 때(부분 중복 질의 수행)의 HCR, HBR, VSPT, AQRT이다. 역시 캐쉬 공간의 크기가 90 KB 이하에서 XML 질의가 수행 되면서 교체가 발생한다. 그러나 이 경우 각 기법의 성능 차이는 크지 않았다. 이는 전체 교체에서 처음에 XML 캐쉬에 저장되는



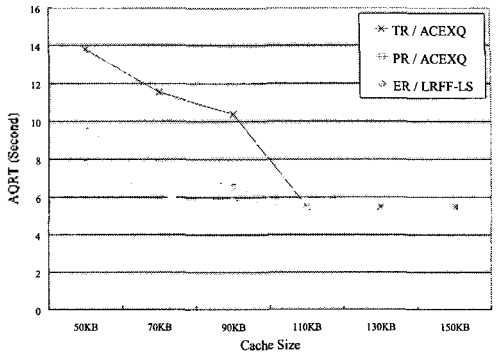

〈그림 12〉 성능 실험 I -contained query(HCR)



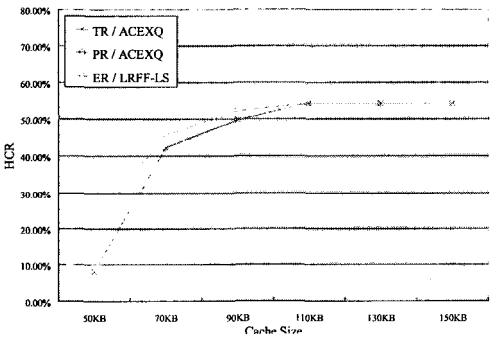

〈그림 13〉 성능 실험 I -contained query(HBR)



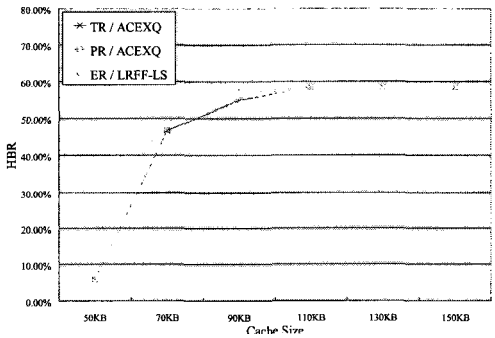

〈그림 14〉 성능 실험 I -contained query(VSPT)



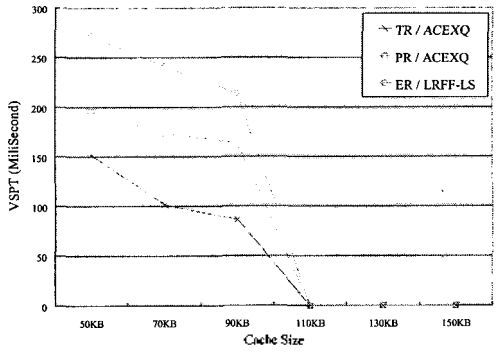

〈그림 15〉 성능 실험 I -contained query(AQRT)



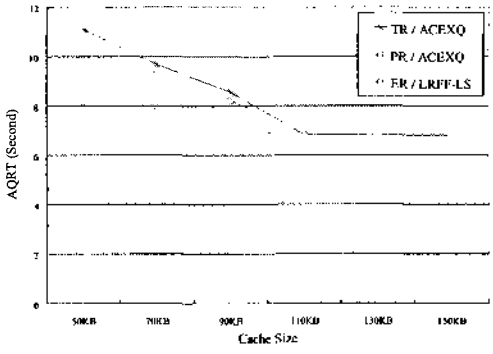

〈그림 16〉 성능 실험 I- overlap query(HCR)




〈그림 17〉 성능 실험 I - overlap query(HBR)




〈그림 18〉 성능 실험 I - overlap query(VSPT)




〈그림 19〉 성능 실험 I - overlap query(AQRT)

〈표 6〉 실험 II의 XML 질의

| | XML 질의 종류 | 요청 빈도(%) | 결과 크기(Byte) |
|---|---|---|---|
| Q1 | /PLAY/ACT/SCENE/SPEECH/LINE/STAGEDIR | 45 | 28,073 |
| Q2 | //SCENE/TITLE | 25 | 52,052 |
| Q3 | /PLAY/ACT//TITLE | 15 | 59,660 |
| Q4 | /PLAY/ACT/SCENE/SPEECH[/SPEAKER = "CURIO"] | 10 | 593 |
| Q5 | /PLAY/ACT/SCENE[/SPEECH/SPEAKER = "Steward"] | 5 | 25,299 |

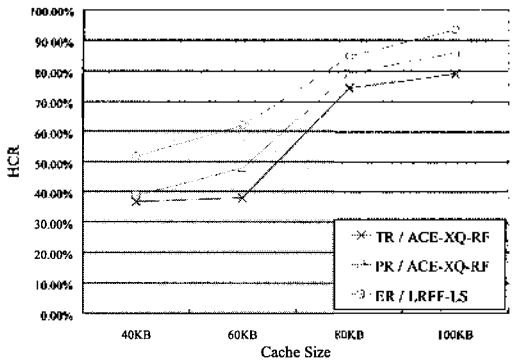
XML 시맨틱 영역의 크기가 작기 때문이다. 완전 포함 질의를 사용할 때와 마찬가지로 캐쉬의 크기가 110K 이상이면 교체가 발생하지 않으므로 가장 좋은 성능을 보이면서 교체 기법 간의 성능 차이는 없다. 그러나 평균 응답 시간에서 완전 포함 질의를 사용할 때보다 부분 중복 질의를 사용할 때 응답 시간이 약간 느린 것을 확인할 수 있다. 이는 부분 중복 질의는 질의가 요청될 때 마다 항상 하부 데이터 서버에 어느 정도의 데이터를 요청하기 때문이다.

### 5.3.2. 실험 II

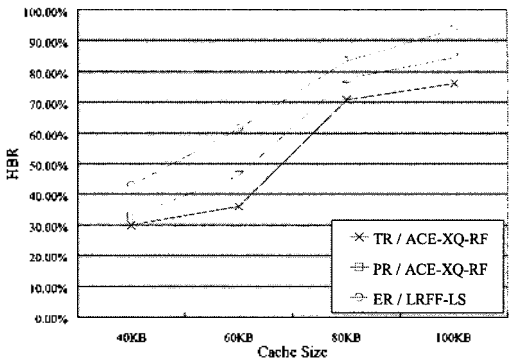
본 실험에서는 〈표 9〉의 XML 질의 5 종류를 사용하였다. 각 질의마다 요청 빈도의 차이를 두었으며, 캐쉬의 크기가 40 KB에서 100KB까지 20KB씩 증가할 때 각각 200회의 질의를 수행한다. 〈그림 20〉, 〈그림 21〉은 세 가지 교체 기법의 HCR, HBR를 비교한 것으로 엘리먼트 교체가 가장 좋은 성능을 보인다. 〈그림 22〉에서는 캐쉬 교체 수행 시 희생자가 제거되는 시간이 엘리먼트 교체에서 가장 오래 걸린다는 것을 확인할 수 있다. 마지막으로 엘리먼트 교체에서 교체 수행 시간이 경로 교체에서의 그것보다 더 많이 걸리지만 전체 질의 응답 시간에서는 엘리먼트 교체의 질의 응답 시간이 가장 좋다는 것을 〈그림 23〉에서 보여준다.

## 6. 결론 및 향후 연구

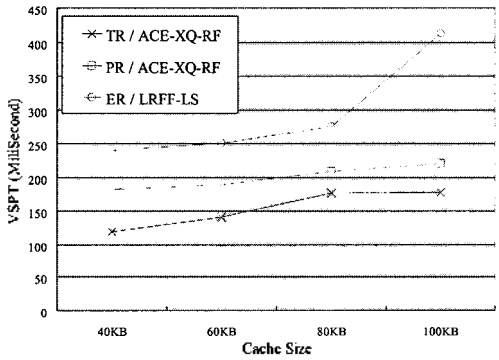
전자 거래를 지원하기 위한 데이터 관리 핵심 기술의 하나로 XML 데이터에 대한 질의 처리 성능의 향상을 들 수 있다. 이를 위해 XML 시맨틱 캐쉬 기술이 주목받고 있으나 이 기술의 완전한 활용을 위해서는 효율적인 캐쉬 교체 기술이 필요하다. 본 논문에서는 기존의 교체 기법 연구에서 나타난 문제점을 해결하기 위하여 캐쉬 교체 단위로 XML 시맨틱 영역 내의 모든 엘리먼트를 교체 단위로 사용하는 엘리먼트 교체를 제시하였다. 그리고 엘리먼트 교체에 적합한 캐쉬 희생자를 선택하기 위한 교체 함수로 각 교체 단위의 적중 빈도, 최근 접근 시간, 인출 지연 시간과 XML 시맨틱 영역



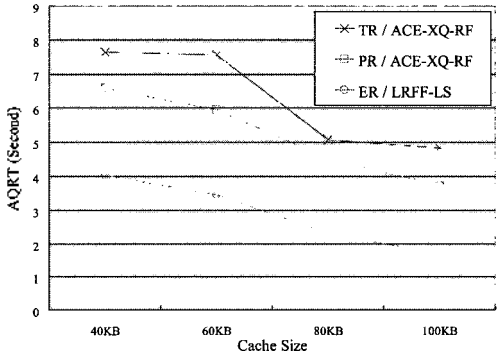

〈그림 20〉 성능 실험 II(HCR)




〈그림 21〉 성능 실험 II(HBR)




〈그림 22〉 성능 실험 II(VSPT)




〈그림 23〉 성능 실험 II(AQRT)

의 크기, 그 영역 내의 모든 엘리먼트의 크기를 이용하여 희생자를 선택하는 교체 함수 LRFF-LS를 제시하였다. 제시한 교체 기법을 적용하여 XML 시맨틱 캐쉬 시스템 프로토타입을 구현하여 실제 LAN 환경에서 데이터 서버와 응용 서버를 구동하여 성능 실험을 수행하였다. 실험 결과 본 논문에서 제시한 엘리먼트 교체를 사용함으로써 기존의 기법들에 비해 캐쉬 교체 희생자 선택을 위한 부담은 늘지만 캐쉬의 적중률을 향상시킴으로서 전체적인 질의 처리 시간이 크게 줄어드는 것을 확인할 수 있었다.

향후 연구과제는 다음과 같다. 첫째, 본 논문에서 제시한 기법을 실제 전자 거래 응용에 적용하는 사례 연구가 필요하다. 전자 거래 응용에서도 XML 데이터를 엘리먼트 단위로 접근하고 처리하는 것이 핵심 연산 중의 하나이므로 본 논문에서 제시한 엘리먼트 기반의 캐쉬 교체 기법의 적용은 효율적일 것으로 예상된다. 둘째, 전자 거래의 소스 데이터가 XML로 기술되어 있지 않고 기존의 RDB에 저장되어 있는 경우에 전자

거래 응용이 요구하는 XML 데이터를 RDB로부터 XML 뷰로 출판하게 되는데 이때의 XML 뷰를 실체뷰로 저장하면 시맨틱 캐쉬에 해당된다. 이 경우의 시맨틱 캐쉬 교체에 본 논문에서 제시한 기법을 적용하기 위한 것으로, RDB로부터 XML 실체뷰를 구성할 때 수행되는 엘리먼트 관련 처리로부터 캐쉬 교체를 위한 통계 정보를 추출하여 활용하는 연구가 필요하다.

## 참 고 문 헌


[1] V. Hristidis and M. Petropoulos. "Semantic Caching of XML Databases". Proc. Workshop on the Web and Databases, 2002.

[2] L. Chen and E. Rundensteiner. "ACE-XQ: A CachE-aware XQuery Answering System". Proc. Workshop on the Web and Databases, 2002.

[3] L. Chen, S. Wang and E. A. Rundensteiner. "A Fine-Grained Replace-ment Strategy for XML Query Cache," Proc. 4th Intl. Workshop on Web Information and Data Management (WIDM'02), pp. 76-83, 2002.

[4] L. Chen, S. Wang and E. A. Rundensteiner, "Replacement Strategies for XQuery Caching Systems," Proc. Elsevier Science Publishers B. V., 2004.

[5] http://metalab.unc.edu/bosak/xml/eg/shaks200.zip

[6] M. Abrams et al., "Caching Proxies: Limitations and Potentials," Proc. 4th Int'l. World Wide Web Conf., 1995.

[7] S. Wiliams et al., "Removal Policies in Network Caches for World Wide Web Documents", Proc. ACM SIGCOMM Conf., Stanford University, Aug. pp. 293-305, 1996.

[8] P. Cao and S. Irani, "Cost-Aware WWW Proxy Caching Algorithms", Proc. 1997 USENIX Symp. Internet Tech. and Sys., pp. 193-206, 1997.

[9] S. Jin and A. Bestavros, "Popularity-Aware Greedy-dual Size Web Proxy Caching Algorithms," Proc. IEEE Int'l. Conf. Distributed Computing Systems (ICDCS), Taiwan, May pp. 254-61, 2000.

[10] S. Jin and A. Bestavros, "Greedy-dual* Web Caching Algorithm", Int'l. J. Comp. Commun., Vol. 24, No. 2, Feb. pp. 174-83, 2001.

[11] C. Aggarwal, J. Wolf, and P. Fellow, "Caching on the World Wide Web", IEEE Trans. Knowledge and Data Eng., Vol. 11, No. 1, Jan./Feb. pp. 94-107, 1999.

[12] L. Rizzo and L. Vicisano, "Replacement Policies for a Proxy Cache", Proc. IEEE/ACM Transactions on Networking (TON), pp. 158-170, 2000.

[13] J. Shim, P. Scheuermann, and R. Vingralek, "Proxy Cache Algorithms:

Design, Implementation, and Performance", IEEE Trans. Knowledge and Data Engineering, Vol. 11, No. 4, July/Aug. pp. 549-62, 1999.

[14] H. Bahn et al, "Efficient Replacement of Nonuniform Objects in Web Caches," IEEE Comp., pp. 65-7, 2002.

[15] R. Wooster and M. Abrams, "Proxy Caching that Estimates Page Load Delays", Proc. 6th Int'l. World Wide Web Conf., Santa Clara, CA, Apr. pp. 325-34, 1997.

[16] N. Niclausse, Z. Liu, and P. Nain, "A New Efficient Caching Policy for the World Wide Web", Proc. Internet Server Perf. Wksp. (WISP '98), Madison, WI, USA, June pp. 119-28, 1998.

[17] S. Dar, M. J. Franklin and B. Jonsson, "Semantic Database Caching and Replacement", Proc. 22nd VLDB, pp. 330-341, 1996.

[18] N. Young, "The k-server Dual and Loose Competitiveness for Paging," Algorithmica, vol. 11, no. 6, June pp. 525-41, 1994.

[19] A. Balamash and M. Krunz, "An Overview of Web Caching Replacement Algorithms", IEEE Communications Surveys & Tutorials, pp. 44-56, 2004.

[20] E. O'Neil, P. O'Neil and G. Weikum, "The LRU-K page replacement algorithm for database disk buffering", Proc. ACM SIGMOD Int'l. Conf. Management of Data, Washington, D.C., USA, May pp. 297-306, 1993.

## 저 자 소 개

홍정우 (E-mail : jwhong@dblab.cse.cau.ac.kr)
2004. 중앙대학교 기계공학부 졸업(학사)
2006. 중앙대학교 대학원 컴퓨터공학과 졸업(석사)
관심분야 XML 데이터베이스, 웹 캐쉬

강현철 (E-mail : hckang@cau.ac.kr)
1983. 서울대학교 컴퓨터공학과 졸업(학사)
1985. U. of Maryland at College Park, Computer Science(M.S.)
1987. U. of Maryland at College Park, Computer Science(Ph.D.)
1988 ~ 현재 중앙대학교 컴퓨터공학부 교수
관심분야 XML 데이터베이스, 웹 데이터베이스, 스트림 데이터 관리, 센서네트워크 데이타베이스