세그먼트 기반의 XML 문서 필터링
(XML Document Filtering based on Segments)

권 준 호 [†] Praveen Rao [††] 문 봉 기 [†††] 이 석 호 [††††]
(Joonho Kwon) (Praveen Rao) (Bongki Moon) (Sukho Lee)

요 약 최근 XML 문서 필터링에 기반한 출판-구독(publish-subscribe) 시스템이 많은 관심을 받고 있다. 전형적인 출판-구독 시스템에서, 구독자들은 XPath 언어로 명세된 프로파일로 자신들의 관심을 표현하고, 새로운 내용들은 사용자 프로파일에 대하여 매칭 여부를 판단하여 관심을 가지고 있는 사용자들에게만 배달된다. 구독자의 수와 그들의 프로파일이 증가할수록, 시스템의 확장성이 출판-구독 시스템의 중요한 성공 요소가 된다. 이 논문에서는 FiST 시스템을 확장한 세그먼트 기반의 XML 문서 필터링 시스템인 SFiST 시스템을 제안한다. SFiST 시스템은 XML 문서 필터링에서 중복된 처리를 없애기 위해서 가지형 패턴의 사용자 프로파일에서 세그먼트를 추출하여 해시 기반의 세그먼트 테이블에 저장하고 유지한다. 이 세그먼트는 사용자 프로파일을 터스 시퀀스 형태로 표현하는데 이용되고, 효율적인 필터링을 위한 컴팩트 시퀀스 인덱스에도 사용된다. 실험을 통하여 세그먼트 기반의 SFiST 시스템이 이전의 연구인 FiST 시스템보다 좋은 성능을 가지고 있음을 보였다.

키워드 : XML 필터링, 세그먼트, 가지형 패턴, Prüfer 시퀀스

Abstract In recent years, publish-subscribe (pub-sub) systems based on XML document filtering have received much attention. In a typical pub-sub system, subscribed users specify their interest in profiles expressed in the XPath language, and each new content is matched against the user profiles so that the content is delivered to only the interested subscribers. As the number of subscribed users and their profiles can grow very large, the scalability of the system is critical to the success of pub-sub services. In this paper, we propose a fast and scalable XML filtering system called SFiST which is an extension of the FiST system. Sharable segments are extracted from twig patterns and stored into the hash-based Segment Table in SFiST system. Segments are used to represent user profiles as Terse Sequences and stored in the Compact Segment Index during filtering. Our experimental study shows that SFiST system has better performance than FiST system in terms of filtering time and memory usage.

Key words : XML filtering, segment, twig pattern, Prüfer sequence

· 본 연구는 지식경제부 및 정보통신연구진흥원의 대학 IT연구센터 육성·지원사업(IITA-2008-C1090-0801-0031)의 연구결과로 수행되었음

† 학생회원 : 서울대학교 전기컴퓨터공학부
joonho@db.snu.ac.kr
†† 비 회 원 : 미주리캔사스대학교 CSEE 교수
raopr@umkc.edu
††† 비 회 원 : 아리조나대학교 전산학과 교수
bkmoon@cs.arizona.edu
†††† 정 회 원 : 서울대학교 전기컴퓨터공학부 교수
shlee@snu.ac.kr

논문접수 : 2008년 1월 9일
심사완료 : 2008년 4월 2일

정보과학회논문지: 데이타베이스 제35권 제4호(2008.8)

1. 서 론

출판-등록(publish-subscribe) 시스템은 선택적 정보 전송(selective dissemination of information) 덕분에 e-commerce와 인터넷 애플리케이션에서 중요한 역할을 하고 있다. 전형적인 출판-등록 시스템은 새로운 내용이 생성될 때마다 그 내용을 관심을 가지고 있는 구독자들에게만 전송한다. 이러한 서비스들은 수백만의 등록된 사용자들에 대하여 많은 수의 내용들을 효율적으로 매칭하는 소프트웨어 시스템을 필요로 한다.

정보 교환을 위한 표준으로 자리 잡은 XML(extensible markup language)의 인기는 정보 전달(information dissemination)을 위한 XML 필터링 시스템을 개발하려는 다양한 연구를 촉진시켰다. 일반적으로 XML

필터링 시스템에서는 사용자들이 XPath 언어[1]를 사용하여 사용자 프로파일(user profile)[1]을 명세한다. 이 논문에서는 가지형 패턴(twig pattern)으로 표현되는 사용자 프로파일을 다룬다. 가지형 패턴은 자식(child)과 후손(descendant) XPath 축(axis)을 포함하는 2개 이상의 선형 경로(linear path)로 구성된다. 다음과 같은 XPath 식이 가지형 패턴이다.

book[author//name="Jonh"]/title

XML 필터링 문제는 XML 문서에서 모든 가지형 패턴의 출현을 발견하는 문제와는 다른 문제이다. XML 문서와 가지형 패턴의 역할이 정반대로 작동하기 때문이다. 이 논문에서 풀고자 하는 필터링 문제는 다음과 같이 표현할 수 있다.

XPath 식이 주어졌을 때, 주어진 XML 문서에 나타나는 XPath 식을 찾아라.

FiST 시스템[2][3]은 XML 문서와 가지형 패턴(사용자 프로파일)들을 Prüfer 시퀀스[4]로 변환하여 전체 가지 패턴 매칭(holistic twig pattern)을 수행한다. FiST의 매칭 알고리즘은 점진적인 서브시퀀스 매칭 단계와 브랜치 노드 검증 단계의 두 단계로 구성된다. 그러나 FiST 시스템은 사용자들 사이의 유사하거나 동일한 가지형 패턴들을 공유하여 처리하지는 못하였다.

이 논문에서는 FiST 시스템을 확장한 SFiST 시스템을 제안한다. SFiST 시스템은 XML 문서 필터링에서 중복된 처리를 없애기 위해서 사용자 프로파일에서 공통으로 나타나는 부분을 세그먼트로 추출한 후, 이를 해시 기반의 세그먼트 테이블에 저장하고 유지한다. 또한 이 논문에서는 세그먼트를 이용한 사용자 프로파일의 터스 시퀀스 표현과 컴팩트 시퀀스 인덱스를 제안하고, 세그먼트 기반의 XML 문서 필터링 알고리즘도 제안한다. 실험을 통하여 SFiST 시스템이 이전의 연구인 FiST 시스템보다 좋은 성능을 가지고 있음을 보였다.

이 논문의 중요한 특징은 다음과 같이 요약할 수 있다.

첫째, 세그먼트 기반의 XML 필터링 시스템인 SFiST 시스템을 제안하였다는 것이다. 사용자 프로파일에서 공통적으로 나타나는 부분을 세그먼트로 추출하여 인덱싱과 필터링 알고리즘에서 이용한다.

둘째, 세그먼트를 기반으로 한 공유 처리를 하는 SFiST 시스템이 기존의 연구인 FiST 시스템보다 빠른 처리 시간을 보이고, 메모리를 효율적으로 사용한다는 것을 실험을 통하여 보였다.

이 논문의 구성은 다음과 같다. 2장에서 XML 필터링과 관련된 기존 연구들을 설명한다. 3장에서는 SFIST 시스템의 이해를 위한 배경 지식을 설명한다. 4장에서는 SFiST 시스템의 구조와 핵심 알고리즘에 대하여 설명한다. 5장에서는 실험 결과에 대하여 분석하고, 마지막으로 6장에서는 연구 결과를 정리하고 결론을 맺는다.

2. 관련연구

정보 교환의 표준으로서 XML의 인기는 확장성 있는 XML 필터링 시스템을 작성하려는 많은 연구들을 촉진시켰다. 관련 연구들은 크게 오토마타 기반의 연구, 인덱스 기반의 연구와 기타 연구의 세가지 범주로 분류할 수 있다.

대다수의 이전 연구들은 사용자 질의로부터 오토마타를 구성하여 XML 필터링 시스템에서 이용한다. XFilter[5]는 XML 필터링 문제에서 첫 번째 연구로서 선형 경로(linear path)만 존재하는 XPath 질의들과 입력 XML 문서와의 매칭 여부를 판단하였다. 이를 위하여 선형 경로들을 하나의 유한 상태 기계(FSM, finite state machine)으로 변환하였다. 후속 연구인 YFilter[6]는 XPath 질의의 공유 처리 모든 XPath 질의를 하나의 NFA(non-deterministic finite automata)로 구성하였다. YFilter는 가지형 패턴을 지원하기 위하여 다음과 같은 과정을 거친다. 가지형 패턴을 여러 개의 선행 경로들로 분해하여 NFA에 추가하고, 각각의 경로들에 대한 매칭 여부를 계산한다. 그런 다음 후처리 과정을 통하여 가지형 패턴의 매칭 여부를 결정한다. 오토마타에 나타나는 상태의 개수를 줄이기 위하여 YFilter는 NFA를 사용하였으나, lazy-DFA[7,8]는 DFA를 이용하지만 상태의 개수를 줄이기 위하여 늦게 오토마타를 구축하는 방법을 이용한다. XML 필터링 동작의 캐시 성능 향상을 위하여 YFilter 기반에 캐시 관리 기법을 추가한 캐시 고려 기법[9,10]도 제안되었다. 이런 기법에서 XML 문서 필터링 과정은 루트 노드에서 리프 노드 방향(하향식)으로 처리된다.

오토마타 이외에 추가적인 버퍼를 사용한 여러 기법들도 존재한다. XSM[11]은 트랜스듀서 기반의 방법을 사용하였으며, XQuery[12] 언어의 일부분을 지원하기 위하여 추가적인 내부 버퍼를 사용한다. 그렇지만 XSM은 '//' 연산자를 지원하지 않는다는 단점이 있다. XPush [13]는 프리디킷을 처리하기 위해서 변형된 결정적 푸시다운 오토마타(deterministic pushdown automata)를 사용하였다. XPush 시스템의 단점은 새로운 질의의 추가나 기존 질의의 삭제를 하기가 어렵다는 것이다. XSQ[14]는 계층 구조를 가지는 푸시다운 오토마타를 사용하여 여러 개의 프리디킷과 집계 연산을 가지는 XPath 질의를 처리하는 방법을 제안하였다. 그렇지만 XSQ 시스템은 한번에 하나의 XPath 식만을 처리한다.

1) 이 논문에서 사용자 프로파일, 가지형 패턴, 사용자 질의는 같은 의미를 가진다.

인덱스 기반의 연구로서 XTrie[15]와 IndexFilter[16] 연구가 있다. XTrie[15]는 가지형 패턴의 필터링을 제공하기 위하여, 선형 경로 분해하여 매칭을 수행한 트라이 기반의 알고리즘을 제안하였다. XTrie는 가지형 패턴에서 '//'와 '*'가 나타나면 '/'만 포함하는 선형 경로로 분해하기 때문에 가지형 패턴에서 '//'의 비율이 높을수록 분해로 인해 생기는 선형 경로의 수가 많아진다는 단점이 있다. IndexFilter의 연구[16]는 인덱스-기반과 탐색 기반 XML multi-query 처리를 연구하였으며, 그 각각이 장단점이 있음을 보였지만 선형 경로로 나타나는 질의들만 고려하였다는 단점이 있다.

기타 기법을 사용하는 분야에도 다양한 연구들이 존재한다. Tian 등은 관계 데이타베이스 시스템에 기반한 XML 기반의 출판-구독 시스템[17]을 제안하였다. 이 연구에서는 XML 필터링 문제를 조인 문제로 변환하여 풀었으나, 질의의 수가 증가함에 따라 조인의 수도 증가하는 단점을 가지고 있다. 블룸 필터(bloom filter)를 사용한 XML 필터링 시스템[18]에서는 XPath 질의를 문자열로 간주하고 질의 문자열들을 해시 함수를 사용하여 블룸 필터에 매핑한다. AFilter[19] 시스템은 XPath 질의들에 나타나는 공통된 접두어(prefix)와 접미어(suffix)를 이용하여 필터링에 사용하지만, 가지형 패턴을 지원하지 않는다는 단점이 있다.

연구동기

오타마타 기반의 대다수의 기존 연구들은 하향식 기반으로 동작을 수행하거나 가지형 패턴을 처리할 수 없는 단점이 있다. 이 논문의 기존연구인 FiST 시스템[2,3]은 상향식(가지 노드에서 루트 노드 방향으로)으로 동작하며, 전체 가지형 패턴 매칭(holistic twig pattern matching)을 수행한다. 이를 위하여 가지형 패턴과 입력 문서를 Prüfer 시퀀스로 변환한다. FiST의 매칭 알고리즘은 점진적인 서브시퀀스 매칭 단계와 브랜치 노드 검증 단계의 두 단계의 과정으로 구성된다. 그렇지만 FiST 시스템은 사용자들 간의 유사하거나 동일한 프로파일을 공유하여 효율적으로 처리하지는 못하였다. 이 논문에서는 YFilter 시스템과 같이 사용자 프로파일 의 공유 처리를 통한 성능 향상을 위하여 FiST를 확장한 세그먼트 기반의 XML 문서 필터링 SFiST 시스템을 제안한다.

3. 배경 지식

이 장에서는 SFiST 시스템을 이해하기 위한 기본 지식들을 설명한다. 먼저 XPath 언어와 XML 데이타에 대한 모델을 설명하고, 다음으로 이전 연구인 FiST 시스템에서 사용자 프로파일과 가지형 패턴을 Prüfer 시퀀스로 변환하는 방법을 기술한다. 그리고 이 논문에서 풀고자 하는 XML 필터링 문제를 정형화한다. 마지막으로 SFiST 시스템의 구조에 대하여 설명한다.

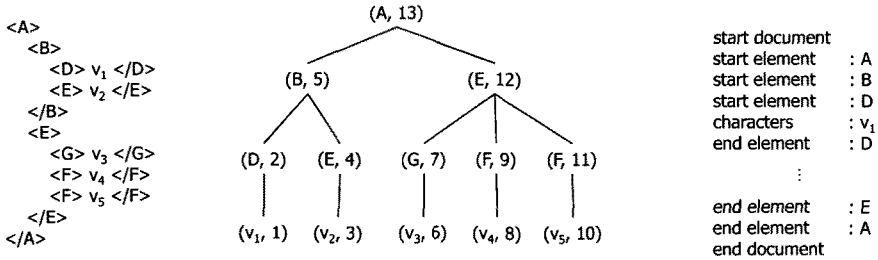
3.1 XPath와 데이타 모델

XPath 언어[1]는 XML 문서의 일부분을 표현하기 위하여 사용되며, XSLT[20]와 XQuery[12] 언어를 이루는 한 구성요소로서 활용된다. SFiST 시스템은 가지형 패턴을 표현하기 위하여 XPath 언어의 일부분을 이용한다. 그림 1은 SFiST 시스템이 처리할 수 있는 XPath 언어의 문법을 보여준다. 문법은 엘리먼트, 애트리뷰트, 와일드카드(wildcard)와 자식(child)과 후손(descendant) 축을 포함한다.

XML 문서는 순서를 가진 레이블된 트리로 모델링할 수 있다. 그림 2(a)의 XML 문서는 그림 2(b)처럼 순서를 가진 레이블된 트리로 표현할 수 있다. 트리에 있는 각각의 노드는 XML 문서의 엘리먼트나 값(value)에 해

Path	:=	RelLocationPath | AbsLocationPath
AbsLocationPath	:=	'/' RelLocationPath
RelLocationPath	:=	Step '/' RelLocationPath | Step
Step	:=	Axis NodeTest | Step '[' Predicate ']'
Axis	:=	'/' | '//' | '@'
NodeTest	:=	String | *
Predicate	:=	Expression | Expression '=' Expression
Expression	:=	String | Path

그림 1 지원하는 XPath 문법



(a) XML 문서 (b) 트리 표현 (c) SAX 파서의 이벤트

그림 2 샘플 XML 문서

당한다. 값들은 문자 데이타(CDATA, PCDATA)로 표현되고 리프 노드에 나타난다. 트리의 간선은 2개의 엘리먼트 사이의 관계나 엘리먼트와 값 사이의 관계를 표현한다. 각각의 엘리먼트는 (애트리뷰트, 값) 쌍을 가질 수 있다. 애트리뷰트는 엘리먼트와 같은 방식으로 처리할 수 있으므로, 이 논문에서는 애트리뷰트와 엘리먼트를 따로 구분하지 않는다.

3.2 XML 문서의 SAX 파싱

시스템에 입력으로 들어오는 XML 문서는 SAX 파서[21]를 이용하여 처리한다. SAX 파서는 각 엘리먼트의 열림 태그와 닫힘 태그를 만날 때 마다 "start element"와 "end element" 이벤트를 호출한다. 문서의 시작과 끝을 만날 때는 "start document"와 "end document" 이벤트를 호출한다. "characters" 이벤트는 열림 태그와 닫힘 태그 사이에 있는 문자열을 반환한다. 그림 2(c)는 그림 2(a)의 문서를 SAX 파싱할 때 발생하는 이벤트의 일부분을 보여준다.

3.3 XML 문서와 가지형 패턴의 Prüfer 시퀀스 변환

Prüfer(1918)는 트리에서 한 번에 하나의 노드를 삭제하여 레이블된 트리와 시퀀스 사이의 일대일 대응을 이루는 방법을 제안하였다[4]. 1부터 n을 레이블로 가진 n개의 노드를 가지고 있는 트리 T_n에서 시퀀스를 생성하는 알고리즘은 다음과 같다. T_n에서 가장 작은 번호를 가진 리프 노드를 삭제하고, 이를 T_{n-1}이라 한다. 삭제된 노드의 부모 노드가 가진 레이블을 a_1이라고 하자. T_{n-1}에서 가장 작은 번호를 가진 리프 노드를 삭제하여 T_{n-2}를 얻고, 그 노드의 부모 노드가 가진 레이블 a_2라 하자. 하나의 간선으로 연결된 2개의 노드가 남을 때까지 이 동작을 반복한다. 추출된 시퀀스 $<a_1, a_2, a_3, ..., a_{n-2}>$가 트리 T_n의 Prüfer 시퀀스이다. 추출된 시퀀스로부터 원래의 트리 T_n을 재구성할 수 있다. 트리 T_n의 Prüfer 시퀀스의 길이는 n-2이다. FiST 시스템은 문서 트리의 리프 노드에 더미 자식 노드를 추가하여 원래 트리의 리프 노드 레이블도 나타나는 확장 Prüfer 시퀀스를 사용한다.

SAX 파서가 XML 문서를 파싱하여 이벤트를 생성하는 과정에서 Prüfer 시퀀스로 변환할 수 있다. XML 문서의 경우 필터링 과정에서 시퀀스의 레이블을 차례차례 이용하게 된다. XPath 파서가 가지형 패턴을 Prüfer 시퀀스 기반의 프로파일 시퀀스로 변환한다. 필터링 과정에서 시퀀스의 레이블뿐만 아니라 XPath 트리의 구조 정보가 더 필요하기 때문에 프로파일 시퀀스에 포함하여 사용한다. 그림 3은 FiST[2,3] 시스템에서 가지형 패턴(사용자 프로파일)을 Prüfer 시퀀스 기반의 프로파일 시퀀스로 변환한 표현을 보여준다. 프로파일 시퀀스의 레이블 속성은 Q_1의 Prüfer 시퀀스 "D B A G E F E A"중 하나의 레이블을 기록하고, Qid 속성은 질의 식별자 번호를 저장하고, sym 속성은 자식 또는 후손 축(axis)을 비롯한 가지형 패턴의 구조 정보를 기록한다.

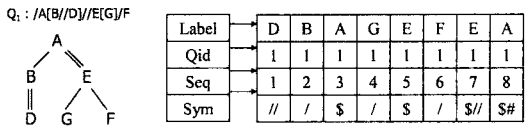


그림 3 가지형 패턴과 프로파일 시퀀스

가지형 패턴에서 자식을 둘 이상 가지는 노드를 브랜치 노드라 하며, 이 노드들은 프로파일 시퀀스에서 자식의 수만큼 등장하는 특징을 가진다. 처음이나 중간에 나타나는 브랜치 노드들은 내부 브랜치 노드라고 하고, 제일 끝에 나타나는 브랜치 노드를 마지막 브랜치 노드라 한다. 그림 3(b)에서 3, 5, 7, 8번 노드가 브랜치 노드들이며, 3번과 5번 노드들이 내부 브랜치 노드이며 7번과 8번 노드들은 마지막 브랜치 노드이다. 8번 노드는 마지막 브랜치 노드이면서 가장 마지막에 나타나는 루트 노드이다. 브랜치 노드들은 사용자 프로파일에서 세그먼트를 추출할 때 사용되며, 4.2절에서 자세히 설명한다.

3.4 XML 문서 필터링 시스템

이 논문에서 풀고자 하는 XML 문서 필터링 문제는 XML 데이타베이스에서 모든 가지형 패턴을 찾는 것과는 다른 동작을 필요로 한다. XML 필터링 문제에서는 입력 문서에 어떤 가지형 패턴이 나타나는지 여부를 빨리 결정하기 위해서 가지형 패턴들을 인덱싱한다. XML 문서 필터링 문제를 다음과 같이 정의할 수 있다.

> 가지형 패턴들의 집합 Q와 XML 문서 D가 주어질 때, 모든 $q \in Q'$가 문서 D와 매칭하는 서브셋 $Q' \subset Q$을 찾아라.

4. SFiST 시스템의 구조와 알고리즘

이 장에서는 SFiST 시스템의 전체적인 구조, 공유처리를 위한 자료 구조와 핵심 알고리즘을 설명한다.

4.1 SFiST 시스템의 구조

그림 4는 SFiST 시스템의 핵심 구성 요소와 전체적인 구조를 보여준다. XPath로 명세한 사용자 프로파일을 XPath 파서가 파싱하여 Prüfer 시퀀스로 변경할 때, 세그먼트(segment)를 추출한다. 이 부분은 4.2절에서 설명한다. 세그먼트들은 효율적인 관리를 위해 세그먼트 테이블(Segment Table)에 저장하여 관리한다. 각각의 사용자 프로파일은 세그먼트를 이용한 터스 시퀀스(Terse Sequence) 형태로 표현한다. 또한 각 프로파일의 문서 매칭 여부를 효율적으로 판단하기 위해서 세그먼트를 기반으로 한 컴팩트 시퀀스 인덱스(Compact

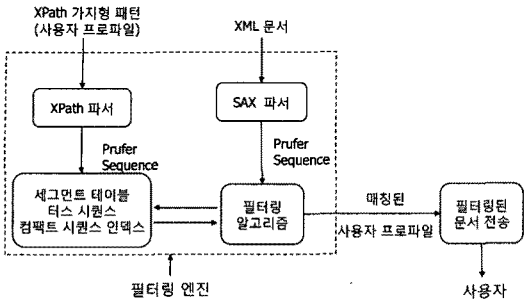


그림 4 SFiST 시스템 구조

Sequence Index)를 사용한다. 세그먼트, 세그먼트 테이블, 컴팩트 시퀀스 인덱스가 SFiST 필터링 엔진의 핵심 자료 구조이며, 자세한 내용은 4.3절에서 설명한다. SAX 파서가 입력으로 주어지는 XML 문서를 파싱한다. 필터링 엔진은 점진적으로 문서로부터 Prüfer 시퀀스를 구성하면서 SAX 파서의 이벤트마다 특정한 동작을 수행한다.

FiST 시스템[2][3]을 확장한 SFiST의 필터링 알고리즘도 점진적인 서브시퀀스 매칭 단계와 브랜치 노드 검증 단계의 두 단계로 구성된다. 첫 번째 단계는 입력 XML 문서와 매칭되는 가지형 패턴들의 수퍼셋(super-set)을 판별하고, 두 번째 단계는 가지형 패턴에서 나타나는 브랜치 노드에 대한 후처리 과정을 통해 잘못된 결과들을 추려낸다. 두 번째 단계는 FiST 시스템과 동일한 알고리즘을 사용하며, 첫 번째 단계는 세그먼트 기반의 자료구조들을 고려한 알고리즘으로 변경되었으며 4.4절에서 이를 설명한다.

4.2 세그먼트(segment)

이 절에서는 가지형 패턴에서 어떤 부분이 공유 가능한지를 정의하고, SFiST 시스템에 어떻게 적용하는지를 설명한다. Prüfer 방식에 따라 가지형 패턴을 시퀀스로 구성할 때, 리프 노드에서 가장 가까운 조상으로 위치한 부모 브랜치 노드에서 뽑을 수 있는 노드 레이블들이 이어져서 프로파일 시퀀스에 나타난다. 가지형 패턴(사용자 프로파일)에서 공유 가능한 부분을 세그먼트(segment)라 하는데, 세그먼트는 아래의 정의 1에 의해 구할 수 있다.

정의 1. 가지형 패턴을 변환한 프로파일 시퀀스의 첫 시퀀스 노드에서 시작하여 각각의 노드를 검사하면서 공유 세그먼트의 끝인지 여부를 검사한다. 다음 공유 가능한 세그먼트는 끝으로 표시한 시퀀스 노드의 다음 시퀀스 노드부터 시작한다. 세그먼트의 끝 여부는 아래의 두 가지 조건으로 결정할 수 있다.

(1) 시퀀스 노드가 내부 브랜치 노드이면, 이 노드는 세그먼트의 끝을 의미한다.

(2) 시퀀스 노드가 프로파일 시퀀스의 마지막 노드이면, 이 노드는 세그먼트의 끝을 의미한다.

3.3절에서, 프로파일 시퀀스에서 나타나는 두 종류의 브랜치 노드인 내부 브랜치 노드와 마지막 브랜치 노드에 대하여 설명하였다. 내부 브랜치 노드와 그 다음의 노드(보통 리프 노드로부터 생성됨) 사이에는 아무런 관계가 존재하지 않는다. 따라서 우리는 내부 브랜치 노드를 세그먼트의 끝으로 표시를 한다. 이와 반대로, 마지막 브랜치 노드와 다음 노드(보통 중간 노드로부터 생성됨) 사이에는 부모-자식 관계나 조상-후손 관계가 성립한다. 따라서 우리는 마지막 브랜치 노드라고 해서 세그먼트의 끝으로 간주하지 않고 다른 조건인 (2)도 만족하는 경우 세그먼트의 끝으로 표시한다.

주의 1. 하나의 프로파일 시퀀스에서 공유 가능한 각각의 세그먼트는 가지형 패턴에서 하나의 가지를 나타낸다. (정의 1)

주의 2. 각각의 공유 가능한 세그먼트들을 필터링 과정에서 한번의 FindSubseqeunce() 함수 호출을 통해 처리할 수 있다.

주의 1과 2에 따르면, 가지형 패턴들에서 공유 가능한 세그먼트를 판별한 후, 중복된 세그먼트 중 하나만 인덱싱하여 SFiST 시스템에서 사용하기 때문에 필터링 과정에서 중복된 처리를 피할 수 있다. 따라서 세그먼트 기반으로 XML 문서 필터링을 수행하면 처리 비용과 저장 비용의 감소를 달성할 수 있다.

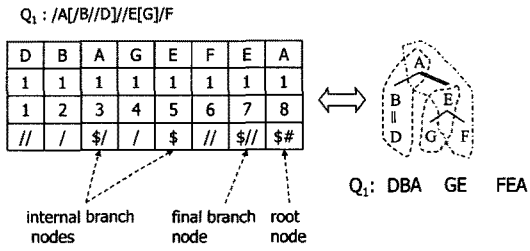
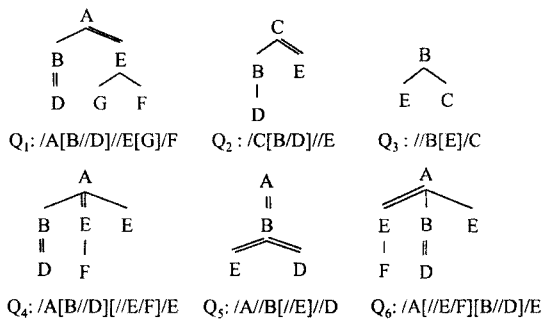


그림 5 가지형 패턴의 공유 가능 세그먼트

예제 1. 그림 5의 가지형 패턴 Q_1을 고려하자. 정의 1의 과정을 적용하면, 3개의 공유 가능한 세그먼트를 얻을 수 있다. 3번과 5번 노드들은 내부 브랜치 노드이므로 세그먼트의 끝으로 표시할 수 있다. 7번 노드는 마지막 브랜치 노드이지만, 8번 노드와 조상-후손 관계를 가지므로 세그먼트의 끝이 될 수 없다. 8번째 노드는 마지막 브랜치 노드이자만 루트 노드이므로 세그먼트의 끝으로 표시할 수 있다. 따라서 Q_1에서 3개의 공유 가능한 세그먼트를 얻을 수 있다.

그림 6은 6개의 가지형 패턴과 그 패턴들에서 얻을



(a) 가지형 패턴 질의

Q_1:	DBA	GE	FEA
	// / $/	/ $	/ $/ $#
Q_2:	DBC	EC	
	/ / $/	// $#	
Q_3:	EB	CB	
	/ $//	/ $#	
Q_4:	DBA	FEA	EA
	// / $/	/ // $#	/ $#
Q_5:	EB	DBA	
	// $	// $// $#	
Q_6:	FEA	DBA	EA
	/ // $/	// / $/	/ $#

(b) 세그먼트

그림 6 가지형 패턴 질의들과 세그먼트들

수 있는 세그먼트들을 보여준다. 예를 들어 선형 경로 "/A[B//D]"는 세 개의 가지형 패턴 Q_1, Q_4, Q_6에 공통적으로 나타남을 알 수 있다.

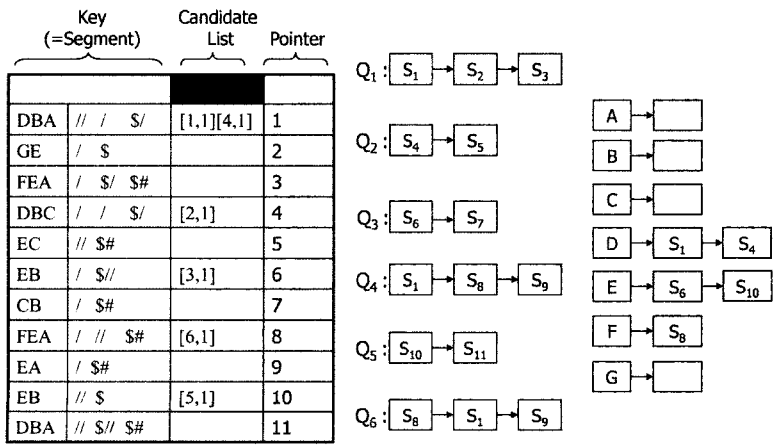
4.3 세그먼트를 이용한 인덱스 구조

이 절에서는 가지형 패턴 질의에서 추출한 세그먼트를 SFiST 시스템이 어떻게 인덱싱하는지를 설명한다.

4.3.1 세그먼트 테이블과 터스 시퀀스

프로파일 시퀀스에서 시퀀스 ID(seqID)와 세그먼트 위치(segPos)로 공유 가능한 세그먼트를 유일하게 식별할 수 있다. seqID는 사용자 프로파일을 식별하는 정수이고, segPos는 프로파일 시퀀스에서 세그먼트의 순서를 결정하는 정수이다. 프로파일 시퀀스의 집합은 해시 테이블 기반의 세그먼트 테이블을 사용하여 터스 시퀀스 형태로 저장된다. 세그먼트 테이블은 공유 가능한 세그먼트들을 키로 사용하고, 후보 리스트(candidate list)를 값으로 하는 해시 테이블이다. 후보 리스트는 [seqId, segPos] 쌍의 집합인데, [seqId, segPos] 쌍은 프로파일 시퀀스에서 입력 문서에 매칭 여부를 판단할 다음 세그먼트를 나타낸다. 이 후보 리스트에 다음 [seqId, segPos] 쌍을 추가하는 것으로서 상태 전이를 표현한다. 초기 상태에는 각 시퀀스의 첫 번째 세그먼트, 즉 segPos 값이 1인 세그먼트들만이 후보 리스트에 저장된다. 사용자 프로파일은 세그먼트 테이블에 있는 세그먼트를 가리키는 포인터들의 순서 리스트 형태인 터스 시퀀스로 표현된다.

예제 2. 그림 7(a)는 그림 6에 있는 6개의 가지형 패턴에 대한 세그먼트 테이블을 나타내고, 그림 7(b)는 터스 시퀀스로 표현된 프로파일 시퀀스를 보여준다. 터스 시퀀스에 있는 S_i는 세그먼트 테이블의 i번째 행을 나타



(a) 세그먼트 테이블 (b) 터스 시퀀스 (c) 컴팩트 시퀀스 인덱스

그림 7 프로파일 시퀀스의 저장과 인덱싱

낸다. 후보 리스트는 프로파일 시퀀스의 첫 번째 세그먼트로 초기화 되어 있다. 예를 들어, 세그먼트 테이블의 첫 행의 후보 리스트는 Q_1과 Q_4의 첫 번째 세그먼트를 나타내고 있다.

그림 7(b)에서 Q_1은 세 개의 세그먼트 S_1, S_2, S_3으로 구성되어 있고, Q_5는 세 개의 세그먼트 S_1, S_8, S_9로 구성되어 있음을 표현한다.

4.3.2 컴팩트 세그먼트 인덱스(compact segment index)

프로파일 시퀀스들은 해시 기반의 컴팩트 세그먼트 인덱스에 저장된다. 이 인덱스에는 공유 세그먼트의 첫 번째 레이블 값이 키 역할을 하고, 세그먼트 테이블에 있는 엔트리에 대한 포인터들이 값 역할을 한다.

예제 3. 그림 7(c)의 컴팩트 시퀀스는 그림 7(b)의 터스 시퀀스로 초기화되어 있다. Q_1과 Q_4의 첫 번째 세그먼트는 동일한 세그먼트이고, 첫 레이블은 'D'이다. 따라서 컴팩트 시퀀스 인덱스의 해시 키 'D'에 세그먼트 테이블의 첫 행을 가리키는 포인터(S_1)이 저장되어 있다. Q_2의 첫 번째 세그먼트 테이블도 첫 레이블로 'D'를 가지기 때문에 포인터 S_4도 해시 키 'D'에 저장된다. 같은 방식으로 해시 키 'E'에는 S_6과 S_{10}이 저장되고, 'F'에는 S_8이 저장된다.

4.4 세그먼트 처리 알고리즘

세그먼트 테이블에 저장된 세그먼트는 FiST 시스템의 프로파일 시퀀스 노드와 같이 label과 sym 속성을 가지고 있기 때문에, 실행 시간 스택을 이용하여 필터링 과정을 진행한다. FiST 시스템의 SAX 파서의 "end element" 이벤트에 구현된 FindSubsequence 알고리즘을 세그먼트를 이용한 방식의 알고리즘 1로 대체하면 XML 문서 필터링을 수행하는 SFiST 시스템이 된다.

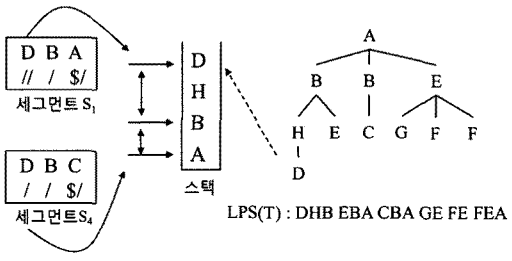
알고리즘 1. 세그먼트 기반의 점진적인 서브시퀀스 매칭

```
Input: L - Prüfer 시퀀스 레이블
procedure FindSubsequence(L)
1  CurrentSegments ← CompactSequenceIndex[L];
2  foreach Segment s in CurrentSegments do
3      if extendedStackTest(s) is true then
4          foreach pair [seqId, segPos] in Candidate List of s do
5              Segment seg ← (segPos+1)th segment of Q_seqId;
6              segL ← the first Prüfer sequence label of seg;
7              Add the pair [seqId, segPos+1] into the Candidate List of seg;
8              Copy seg into the CompactSequenceIndex[segL];
           endfch
       endif
   endfch
End
```

FindSubsequence에서는 SAX 파서에서 얻어진 레이블 'L'을 이용을 하여 컴팩트 시퀀스 인덱스에 저장되어 있는 세그먼트들을 찾는다(1행). 세그먼트가 가지고 있는 label과 sym 속성을 XML 문서 파싱 시 유지하는 실행 시간 스택과 비교를 한다(4행). 실행 시간 스택은 XML 문서를 파싱하면서 나타난 엘리먼트들을 저장하는데, 열린 태그와 닫힌 태그를 만날 때 마다 엘리먼트들을 저장하거나 삭제한다. 실행 시간 스택 테스트를 통과한 세그먼트들의 후보 리스트에 유지되고 있는 쌍(pair)은 사용자 프로파일의 현재 세그먼트대한 정보이므로, 다음 세그먼트의 후보 리스트에 새로운 쌍(pair)을 추가한다(5-7행).

예제 3. 그림 8(a)의 XML 문서와 그림 8(b)의 세그먼트 테이블을 고려하자. 주어진 가지형 패턴들로부터 추출한 세그먼트, 세그먼트 테이블, 컴팩트 세그먼트 시퀀스의 초기상태는 그림 7과 같다.

FindSubsequence(D)과 호출될 때, 스택의 상태는 그림 8(a)에 나타나 있다.

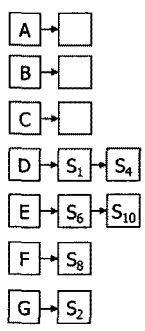


(a) 세그먼트와 스택의 비교

세그먼트 테이블

DBA	// / $/	[1,1][4,1]	1
GE	/ $	[1,2]	2
FEA	/ $/ $#		3
DBC	/ / $/	[2,1]	4
EC	// $#		5
EB	/ $//	[3,1]	6
CB	/ $#		7
FEA	/ // $#	[6,1][4,2]	8
EA	/ $#		9
EB	// $	[5,1]	10
DBA	// $// $#		11

컴팩트 시퀀스 인덱스



(b) 세그먼트 테이블과 시퀀스 인덱스의 변화

그림 8 FindSubsequence(D)가 호출 시

레이블 'D'를 가지고 FindSubsequence가 호출 되었으므로, 컴팩트 시퀀스 인덱스에서 해시 키 'D'에 저장된 세그먼트 S_1과 S_4를 가져온다. S_1은 스택 검사를 통과하지만, S_4 스택 검사에 실패한다. S_1의 후보 리스트(candidate list)에는 두 개의 쌍 [1,1]과 [4,1]이 저장되어 있으므로, Q_1의 두번째 세그먼트 S_2와 Q_4의 두 번째 세그먼트 S_8의 후보 리스트에 쌍 [1,2]와 [4,2]를 각각 저장한다. 또한 S_2와 S_4를 컴팩트 세그먼트 인덱스에 삽입한다. 세그먼트 테이블과 컴팩트 시퀀스 인덱스의 이런변경은 그림 8(b)에 표현되어 있다.

5. 실험 결과

이 논문에서는 FiST 시스템과 SFiST 시스템을 비교하였다. FiST와 SFiST 시스템의 성능을 다양한 크기의 XML 문서와 사용자 프로파일(가지형 패턴)을 사용하여 측정하였다.

5.1 실험 환경

512MB의 메모리를 가지고 있고 리눅스가 설치되어 있는 Intel Pentium IV 2.4 GHz CPU 기계에서 모든 실험을 수행하였다. FiST와 SFiST 시스템은 Xerces XML Parser 2.5.0 버전[22]을 사용하여 c++로 구현되었으며, GNU g++ 3.3.2를 사용하여 컴파일 하였다.

실험을 위하여 DBLP[23]와 TreeBank[24] DTD를 사용하여 사용자 프로파일과 XML 문서를 사용하였다. XML Generator[25]를 사용하여 XML 문서들을 1000개 생성하였다. 생성된 문서는 크기에 따라 [5KB, 6KB], [10KB, 20KB), [20KB, 30KB)로 분류되고, 이 논문에서는 편의상 이 데이타셋을 5k, 10k, 20k라 한다. 각 그룹의 속한 1000개의 문서에 대한 필터링을 수행한 후 구한 수행 시간의 평균을 실험 결과로 사용하였다.

사용자 프로파일은 YFilter[6]에서 배포하는 XPath 생성기를 사용하여 50,000부터 250,000개까지 생성하여 실험에 사용하였다.

5.2 중복도의 증가

출판/등록 시스템에서, 많은 사용자들은 공통의 관심을 공유할 수 있으며 이는 중복 또는 유사한 사용자 프로파일로 표현된다. 이 실험에서는 중복된 사용자 프로파일이 문서 필터링 시간에 어떤 영향을 주는지 알아본다.

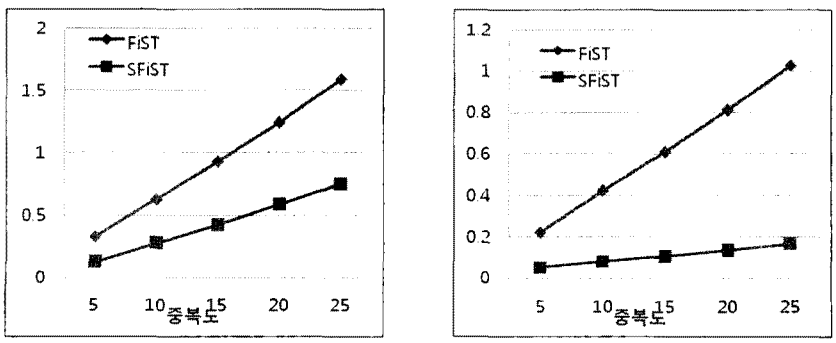
실험 데이타셋을 위하여, 먼저 중복을 없앤 사용자 프로파일을 생성한 후 이를 여러 번 반복한다. 반복의 횟수를 중복도(degree of duplicate)라 칭한다. 예를 들어, 처음에 10,000개의 서로 다른 사용자 프로파일을 생성한 후 이를 25번 반복하여 250,000개의 사용자 프로파일 데이타셋을 만들었다면 중복도는 25이다.

그림 9는 서로 다른 10,000개의 데이타를 생성한 후 중복도를 5에서 25까지 변화시켰을 때의 FiST와 SFiST의 실행시간을 보여준다. 그림 9(a)와 (b)는 각각 DBLP DTD와 Treebank DTD를 사용한 실험이다. X축은 중복도를 의미하고 Y축은 초 단위의 경과된 시간을 의미한다.

중복도가 증가함에 가지형 패턴의 수도 많아지기 때문에 필터링 시간도 증가함을 알 수 있다. 그렇지만 중복도가 증가함에 따라 FiST 시스템과 SFiST 시스템의 시간 차이가 점점 더 벌어지는 경향을 볼 수 있다. 이는 세그먼트를 사용한 공유 처리 기법의 효율성을 의미한다. Treebank 데이타셋에서 SFiST의 필터링 시간 증가량이 DBLP 데이타셋의 경우보다 훨씬 적음을 알 수 있다. 이것은 DBLP보다 Treebank의 구조가 복잡하여 Treebank 데이타셋에서 FindSubsequence 함수 수행 과정에서 매치로 판별되는 질의의 개수가 DBLP 데이타셋에서 나타나는 경우보다 훨씬 적기 때문이다. DBLP의 경우 SFiST가 최대 52% 정도 빨랐으며, Treebank의 경우 최대 83% 빨랐다.

5.3 사용자 프로파일 수의 증가

FiST 시스템과 SFiST 시스템의 확장성(scalability)를 살펴보기 위하여 사용자 프로파일의 수를 다양하게 변경하면서 성능을 측정하였다. 사용자 프로파일의 중복도는



(a) DBLP, 10K (b) Treebank, 10K

그림 9 중복도의 영향

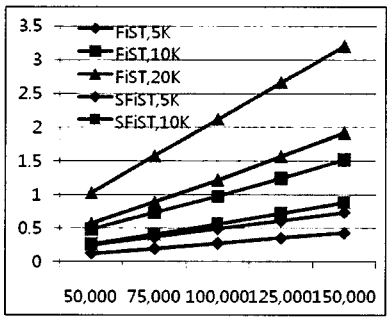
15이고, 사용자 프로파일의 브랜치 수는 2로 고정하였다.

그림 10은 사용자 프로파일의 증가(X축)에 따른 필터링 시간(Y축)을 보여준다. 사용자 프로파일이 증가함에 따라 필터링 시간도 증가한다. 그러나 SFiST 시스템은 FiST 시스템보다 덜 증가한다. 이는 SFiST 시스템의 확장성이 더 뛰어남을 보여준다. 특히 Treebank 데이타셋에서 SFiST 시스템은 좋은 확장성을 보여준다.

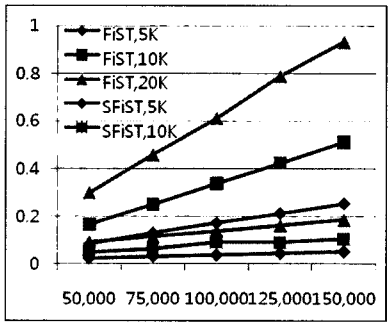
5.4 메모리 사용량

XML 문서 필터링 시스템이 사용하는 메모리의 양 또한 중요한 척도이다. 이번 실험에서는 총 사용자 프로파일의 수를 75,000개로 고정하고 중복도를 1에서 25로 변화 시키면서 필터링 시스템이 사용하는 메모리의 사용량을 측정하였다. 중복도에 따라 서로 다른 사용자 프로파일의 수는 3,000개에서 75,000개 변화한다.

그림 11은 FiST와 SFiST 시스템이 사용하는 메모리 사용량을 보여준다. X축은 중복도를 의미하고, Y축은 메모리 사용량(MB)을 보여준다. 우선 중복도가 증가할수록 FiST와 SFiST 시스템의 메모리 사용량도 감소하는 것을 볼 수 있다. 이것은 표 1에 나타난 것처럼 처리해야 할 총 시퀀스 노드의 수와 세그먼트의 수가 감소함에 따라 사용량도 감소한다고 해석할 수 있다. DBLP 데이타셋도 같은 경향을 보여주기 때문에, Treebank 데이타셋만 표시하였다. 또한 공통으로 나타나는 세그먼트를 공유하여 처리하는 SFiST 시스템이 FiST보다 적은 양의 메모리를 사용하는 것을 알 수 있다. 중복도가 25인 경우 SFiST가 FiST 보다 40% 정도 적은 메모리를 사용한다

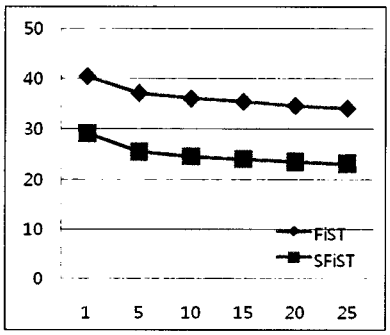


(a) DBLP, 가지수=2

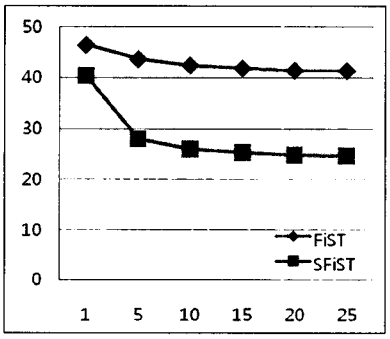


(b) Treebank, 가지수=2

그림 10 사용자 프로파일의 증가



(a) DBLP, 5K



(b) Treebank, 5K

그림 11 메모리 사용량

표 1 사용자 프로파일에 대한 통계 정보

DTD	중복도	총 시퀀스 노드의 수 (FiST)	총 세그먼트의 수 (SFiST)	세그먼트 테이블에 저장된 세그먼트의 수
Treebank	1	769,815	177,044	93,164
	5	712,625	169,510	20,296
	10	686,620	166,140	10,407
	15	673,635	165,000	7,119
	20	665,880	163,500	5,347
	25	665,775	163,350	4,407

그림 11에서 중복도가 증가함에 따라서 SFiST의 메모리 사용량도 감소하지만 그 감소폭은 줄어드는 경향을 볼 수 있다. 이것은 4.3절에서 설명한 세그먼트 테이블의 구조가 원인이다. 비록 세그먼트 테이블에 저장되는 세그먼트의 수는 크게 감소하지만, 필터링 과정에서 세그먼트가 어떤 질의의 몇 번째 부분이지 기록하는 세그먼트 테이블의 후보 리스트는 공유할 수 없는 부분이므로 질의의 수가 고정되면 크게 감소하지 않는다. 이런 이유로 중복도가 증가해도 메모리가 크게 감소하지는 않는다.

그림 12는 입력 XML 문서의 변화에 따른 메모리 사용량을 보여준다. X축은 실험에 사용한 XML 문서의 크기를 Y축은 메모리 사용량(MB)을 나타낸다. DBLP 데이타셋도 같은 경향을 보여주기 때문에, Treebank 데이타셋을 사용한 결과만을 표시하였다. FiST와 SFiST 시스템 둘 다 입력 XML 문서의 크기가 5k에서 20k로 변화할 때의 메모리 크기는 거의 증가하지 않음을 알 수 있다. 이는 FiST와 SFiST에서 사용하는 실행 시간 스택의 크기가 XML 문서의 최대 깊이만큼의 원소들만을 저장하기 때문이다. 실제로 입력으로 들어온 Treebank 데이타셋 XML 문서의 최대 깊이의 평균은 5k의 경우 18.59, 10k인 경우 20.91, 20k인 경우 21.00으로 미세하게 증가한다.

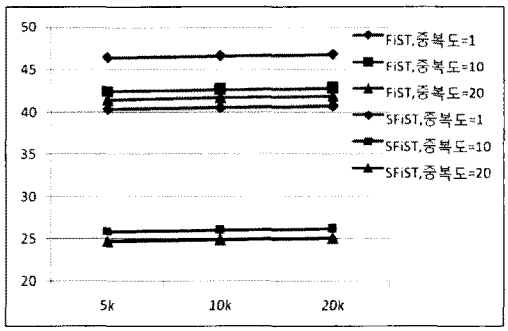


그림 12 Treebank 데이타 처리시 메모리 사용량

그림 11과 그림 12에서 보았던 것처럼 모든 경우에서 SFiST가 FiST 보다 최대 40% 정도 적은 양의 메모리를 사용함을 알 수 있다.

5.5 실험 요약

이 논문에서는 다른 속성을 가진 데이타를 이용한 SFiST 시스템과 FiST 시스템의 실험을 통한 결과는 다음과 같이 요약할 수 있다.

> 가지형 패턴(사용자 질의)에서 공통적으로 나타나는 부분을 공유하여 처리함으로써 상당한 성능 향상을 달성할 수 있다. 많은 수의 중복이 있는 경우, SFiST 시스템은 FiST 시스템보다 40% 적은 메모리를 사용하면서 대략 48% 빠른 처리 시간을 보여주었다.

6. 결 론

이 논문에서는 XML 문서 필터링 과정에서 중복된 처리를 없애기 위해서 세그먼트를 사용한 효율적인 XML 필터링 시스템인 SFiST 시스템을 제안하였다. SFiST 시스템은 가지형 패턴의 사용자 프로파일에서 세그먼트를 추출하여 해시 기반의 세그먼트 테이블에 저장하고 유지한다. 이 세그먼트는 사용자 프로파일을 터스 시퀀스 형태로 표현하는데 이용되고, 효율적인 필터링 위한 컴팩트 시퀀스 인덱스에도 사용된다. 실험을 통하여 세그먼트 기반의 SFiST 시스템이 이전의 연구인 FiST 시스템보다 좋은 성능을 가지고 있음을 보였다.

참 고 문 헌

[1] James Clark, Steve DeRose, "XML Path Language (XPath) version 1.0," http://www.w3.org/ TR/ xpath/ (Nov. 1999).

[2] Joonho Kwon, Praveen Rao, Bongki Moon, Sukho Lee, "FiST: Scalable XML Document Filtering by Sequencing Twig Patterns," In Proceeding of the 31st VLDB Conference, pp. 217-228, 2005.

[3] 권준호, Praveen Rao, 문봉기, 이석호, "가지형 패턴의 시퀀스화를 이용한 XML 문서 필터링", 정보과학회논문지:데이타베이스, 제33권, 제4호, pp. 423-436, 2006.

[4] H. Prüfer, "Neuer Beweis eines Satzes über Permutationen," Archiv fur Mathematik und Physik, 27: 142-144, 1998.

[5] Mehmet Altinel, Michael J. Franklin, "Efficient Filtering of XML Documents for Selective Dissemination of Information," In Proceeding of the 26th VLDB Conference, pp. 53-64, Cairo, Egypt, September 2000.

[6] Yanlei Diao, Mehmet Altinel, Michael J. Franklin, Hao Zhang, Peter Fischer, "Path sharing and predicate evaluation for high-performance XML filtering," ACM Trans. Database Syst, 28(4) : 467-516, 2003.

[7] Todd J. Green, Gerome Miklau, Makoto Onizuka, Dan Suciu, "Processing XML Streams with Deterministic Automata," In Proceedings of the 9th International Conference on Database Theory. Siena, Italy, 2003, pp. 173-189.

[8] Todd J. Green, Gerome Miklau, Makoto Onizuka, Dan Suciu, "Processing XML streams with Deterministic Automata and Stream Indexes," ACM Trans. Database Syst., Vol.29, No.4, pp.752-788, 2004.

[9] Bingsheng He, Qiong Luo, Byron Choi, "Cache-Conscious Automata for XML Filtering," In Proceedings of the 21st IEEE International Conference on Data Engineering. Tokyo, Japan, 2005, pp. 878-889.

[10] Bingsheng He, Qiong Luo, Byron Choi, "Cache-

Conscious Automata for XML Filtering," IEEE Trans. Knowl. Data Eng, Vol.18, No.12, pp. 1629-1644, 2006.
[11] Bertram Ludäscher, Pratik Mukhopadhyay, Yannis Papakonstantinou, "A transducer-based XML query processor," In Proceedings of the 28th VLDB Conference, Hong Kong, China, 2002, pp. 227-238.
[12] Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robiem, Jérôme Siméon, "XQuery 1.0: An XML Query Language," http://www.w3.org/TR/xquery/.
[13] Ashish Kumar Gupta and Dan Suciu, "Stream processing of XPath queries with predicates," In Proceeding of the 2003 ACM-SIGMOD conference, pp. 419-430, San Diego, CA, June 2003.
[14] Feng Peng and Sudarshan S. Chawathe, "XPath queries on streaming data," In Proceeding of the 2003 ACM-SIGMOD Conference, pp. 431-442, San Diego, CA, June 2003.
[15] Chee Yong Chan, Pascal Felber, Minos N. Garofalakis, Rajeev Rastogi, "Efficient Filtering of XML Documents with XPath Expressions," In Proceedings of the 18th IEEE International Conference on Data Engineering, pp. 235-244, San Jose, CA, February 2002.
[16] Nicolas Bruno, Luis Gravano, Nick Koudas, Divesh Srivastava, "Navigation- vs. Index-Based XML Multi-Query Processing," In Proceedings of the 19th IEEE International Conference on Data Engineering, pp. 139-150, Bangalore, India, March 2003.
[17] Feng Tian, Berthold Reinwald, Hamid Pirahesh, Tobias Mayr, Jussi Myllymaki, "Implementing a Scalable XML Publish/Subscribe System Using a Relational Database System," In Proceeding of the 2004 ACM-SIGMOD Conference, pp. 479-490, Paris, France, June 2004.
[18] Xueqing Gong, Ying Yan, Weining Qian, Aoying Zhou, "Bloom Filter-based XML Packets Filtering for Millions of Path Queries," In Proceedings of the 21st IEEE International Conference on Data Engineering. Tokyo, Japan, 2005, pp. 890-901.
[19] K. Selçuk Candan, Wang-Pin Hsiung, Songting Chen, Jun'ichi Tatemura and Divyakant Agrawal, "AFilter: adaptable XML filtering with prefix-caching suffix-clustering," In Proceedings of the 32nd VLDB Conference, Seoul, Korea, 2006, pp. 559-570.
[20] James Clark, "XSL Transformations (XSLT) Version 1.0," http://www.w3.org/TR/xslt/ (Nov. 1999).
[21] David Megginson, Simple API for XML, http://sax.sourceforge.net/
[22] Apache Xerces C++ Parser. http://xml.apache.org/xerces-c/
[23] Michael Ley, DBLP Bibliography. http://www.informatik.uni-trier.de/~ley/db/
[24] The Penn Treebank Project, http://www.cis.upenn.edu/~ treebank/
[25] Angel Luis Diaz and Douglas Lovell, XML Generator. http://www.alphaworks.ibm.com/ tech/xml-generator

권 준 호
1999년 서울대학교 컴퓨터공학과 졸업 2001년 서울대학교 전기컴퓨터공학부 석사학위 취득. 2001년~현재 서울대학교 전기컴퓨터공학부 박사과정. 관심분야는 XML 인덱싱, XML 문서 필터링, 웹 서비스등

Praveen Rao
1999년 인도 University of Pune 컴퓨터공학과 졸업. 2001년 미국 University of Arizona 전산학과 석사학위 취득 2007년 미국 University of Arizona 전산학과 박사학위 취득. 2007년~현재 미국 Missouri-Kansas city 대학 CSEE 조교수. 관심분야는 XML indexing and query processing, filtering /aggregation over XML, XML & Peer-to-Peer (P2P) systems 등

문 봉 기
1996년 미국 메릴랜드 대학 박사학위 취득. 현 University of Arizona, Computer Science 부교수. ACM SIGMOD (2003), EDBT (2002), ISDB (2002), WWW (2002), VLDB (2001), ACM SIGMOD (2001) 위원회 위원 역임. 2002년 ACM SIGMOD Proceedings Chair 역임. 관심분야는 XML indexing, data mining, data warehousing과 parallel & distributed processing 등

이 석 호
1964년 연세대학교 정치외교학과 졸업 1975년, 1979년 미국 텍사스대학교 전산학 석사와 박사학위 취득. 1979년~1982년 한국과학원 전산학과 조교수. 1982년~1986년 한국정보과학회 논문 편집위원장. 1986년~1989년 미국 IBM T.J. Watson 연구소 객원교수. 1988년~1990년 데이타베이스연구회 운영위원장. 1989~1991년 서울대학교 중앙교육연구전산원 원장. 1994년 한국정보과학회 회장. 1997년~1999년 한국학술진흥재단부설 첨단학술정보센터 소장. 1982년~현재 서울대학교 컴퓨터공학부 교수