공 학 박 사 학 위 논 문

XML 데이터베이스 질의를 위한 새로운 인덱싱 기법에 관한 연구

2006년 2월

부 경 대 학 교 대 학 원

컴 퓨 터 공 학 과

박 희 숙

공 학 박 사 학 위 논 문

XML 데이터베이스 질의를 위한 새로운 인덱싱 기법에 관한 연구

지도교수 조 우 현

이 논문을 공학박사 학위논문으로 제출함.

2006년 2월

부 경 대 학 교 대 학 원

컴 퓨 터 공 학 과

박 희 숙

박희숙의 공학박사 학위논문을 인준함.

2006년 2월 14일

주 심 공학박사 정 목 동 인

위 원 이학박사 윤 성 대 인

위 원 공학박사 류 시 국 인

위 원 공학박사 권 오 흠 인

위 원 공학박사 조 우 현 인

목 차

표 목 차

그 림 목 차

A Study on New Indexing Techniques for XML Database Queries

Hee-Sook Park

Department of Computer Engineering, The Graduate School,
Pukyong National University

Abstract

In this dissertation, we propose two new indexing techniques to improve the searching performance of path-oriented queries against XML document stored in relational databases.

In the first proposed indexing technique, an abbreviation path file to perform path-oriented queries efficiently is generated which is able to use its hash-code value to index keys. Also this technique can be further enhanced by combining the Extendible Hashing technique with the abbreviation path file for improving performance of retrieval. We will be named this technique, as EHP(Extendible Hashing with abbreviation Path)-indexing method.

In the other technique, we propose a Parallel Match Indexing Fabric to speed up evaluation of path-oriented query using path signature and design the parallel match algorithm to perform match process between a path signature of input query and path signatures of elements stored in the database. To construct structure of the parallel match indexing, we first make the binary trie for all path signatures of elements on an XML document and then which trie is transformed to the Parallel Match Indexing Fabric. Also we use a Parallel Match Indexing Fabric and a parallel match algorithm for executing search operation of a path-oriented query. The technique is termed

PMP(Parallel Match with Path signature)-indexing method.

When the number of path signatures in path signature file is small, although an existing method can gain better performance than proposed method, we find that proposed approaches in this dissertation can accomplish further advancement of performance as increasing the number of path signatures in the path signature file. Also, the existing method reveals a difference of performance depending on how many bits are set to 1 within a path signature and how many number of path signatures are included in the path signature file. However, the performance of the proposed methods here is not influenced by neither the number of path signatures(N) nor the number of bits set to 1, but only influenced by the length of a path signature. Accordingly, proposed approaches can achieve to a significant improvement of evaluation performance of the path-oriented query to XML documents stored in database.

When we use proposed approaches in great than medium volume of database, we will get more benefits than using existing approach. In our proposed approaches, each time complexity of algorithms is $O(k)$ and $O(h)$ in the worst case, where k represents the number of slots a bucket and h is the length of path signature.

1. 서 론

최근 인터넷의 폭발적인 성장과 함께 인터넷상에서 정보를 효과적으로 교환하는 문제는 그 중요성이 더욱 증가되고 있다. W3C(World Wide Web Consortium)에 의해 제안된 XML(eXtensible Markup Language)은 인터넷상에서 데이터를 교환하기 위한 표준 수단으로서 그 위치를 확고히 하고 있다. 또한, XML은 네트워크상에서 문서를 교환하고 조작하기 위해 사용되고 있으며, 이것은 반-구조적(semi-structured)인 데이터의 표현이 가능한 문법이다. XML의 잠재성은 무한하며 XML을 이용한 많은 애플리케이션들이 현재 개발되고 있다 [18, 31, 33, 35, 70].

이와 같은 시대적 배경에 따라 관계형 데이터베이스에 저장된 대규모 XML 문서를 대상으로 사용자가 입력한 경로-지향 질의(Path-oriented Query)를 평가하는 데 있어서 평가 속도의 효과적인 개선에 관한 문제는 오늘날 중요한 연구과제가 되고 있다. 따라서 XML문서를 저장하고 있는 데이터베이스로부터 경로-지향 질의를 수행 할 때 질의어의 평가 성능을 높이기 위한 효과적인 인덱싱 알고리즘에 대한 연구가 필요하다.

지금까지 경로-지향 질의어에 대한 평가 성능을 개선하기 위한 몇 가지 순차적 매치 인덱싱 기법들이 제안되어 왔었다. 그 대표적인 사례들로는 Chen에 의해 제안되었던 경로서명 파일과 패트리샤 트리(Patricia Tree: PT) 기술을 결합한 기법이 있다. 또 다른 기법으로는 엘리먼트들에 대한 넘버링 스키마를 기반으로 한 XML문서 저장과 인덱싱을 위한 기법이 소개되었다 [1, 51].

병렬 매치 인덱싱 기법에 관한 기존의 연구는 주로 텍스트 문서 검색을 위

한 기법들이 주류였으며, 그들 중에서 가장 대표적인 것으로는 Stanfill과 Kahle에 의해 연구되었던 것으로 수평적 서명 분할법과 SIMD 컴퓨터구조를 사용하는 기법이 있다 [64].

기존의 제안된 기법들 중에서 Chen이 제안하였던 기법은 본 논문에서 제안한 기법들과 같이 XML문서 데이터베이스를 대상으로 한 경로-지향 질의어를 평가하고 있다. 따라서 본 논문에서 제안한 기법들에 대한 검색 성능을 비교하기 위한 대상으로 이 기법을 활용한다.

기존의 기법은 패트리샤 트리 구조에서 외부노드에 한 개의 정보만을 저장하는 방식을 사용하고 있다. 따라서 데이터의 양이 증가함에 따라 패트리샤 트리의 구조 확장이 요구된다. 또한, 검색을 위한 매치 연산을 수행할 때 이 기법은 사용자에 의해 입력된 경로-지향 질의어의 경로서명값이 가지는 이진 비트 패턴의 형태에 따라서 키의 검색 성능이 크게 영향을 받는다는 문제점을 가진다.

본 논문에서는 기존의 기법이 가진 문제점들을 보완하고 경로-지향 질의어의 평가 성능을 향상하기 위한 두 가지 인덱싱 기법을 제안한다.

첫 번째 인덱싱 기법은 각 엘리먼트의 단축경로(Abbreviation Path)를 생성하고 이것을 단축경로 파일에 저장한 다음 인덱스 키로 사용한다. 여기서 단축경로를 만드는 방법은 XML문서의 경로(path)상에 존재하는 각 엘리먼트(Element)들의 첫 번째 문자들을 연결하여 생성한 다음 단축경로 파일에 저장한다. 파일에 저장된 단축경로들은 인덱스 키로 사용되며 질의 평가속도를 개선하기 위해 확장성 해싱(Extendible Hashing)기술과 결합한다. 본 논문에서는 이 인덱싱 기법을 EHP(Extendible Hashing with abbreviation Path)-indexing 기법이라 부른다. EHP-indexing 기법은 기존의 제안된 인덱싱 기법보다 질의

평가속도와 요구되는 기억 공간측면에서 더욱 향상된 결과를 나타낸다. 그 이유는 확장성 해싱은 키의 삽입이나 삭제 연산에서 기억공간의 요구량이 동적으로 변하기 때문이다. 이 기법은 디렉토리가 메모리 내부에 있다면 1회만 접근하면 되고, 만약 디렉토리가 페이지화 되어있다면 2회의 메모리 접근을 필요로 하기 때문이다. 따라서 메모리 접근 수행시간은 많아야 2회 이하로 수행이 가능하기 때문에 *O(1)*의 시간 복잡도를 가진다.

기존의 기법은 최악의 경우에 있어서 검색을 위해 키의 최대 비교 횟수로 O(N/2^l)의 시간 복잡도를 요구하고 있다. 여기서 *N*은 경로서명 파일의 최대 엔트리의 수를 뜻하고 *l*은 경로서명에서 1로 설정된 비트의 수를 뜻한다. 그러나 EHP-indexing 기법은 키의 검색을 위한 비교횟수로 최악의 경우에도 각 버킷당 슬롯의 수인 k만큼만을 요구한다. 따라서 EHP-indexing 기법의 시간 복잡도는 *O(k)*이다.

두 번째 인덱싱 기법은 경로서명(Path Signature)과 이진 트라이(binary trie)구조를 개선한 형태인 병렬 매치 인덱싱 구조(Parallel Match Indexing Fabric: PMIF)를 함께 사용하는 기법이다. 이 인덱싱 기법을 PMP(Parallel Match with Path signature)-indexing 기법이라 부른다.

PMP-indexing 기법은 다음과 같은 순서로 실행한다.

첫째, 생성된 경로서명 파일을 이용하여 이진 트라이 구조를 구성한다.

둘째, 구성된 이진 트라이 구조를 프로세스 엘리먼트(Process Element: PE)들로 구성된 병렬 매치 인덱싱 구조로 변환한다.

셋째, 평가를 위해 입력된 경로-지향 질의어에 포함된 엘리먼트들의 단축경로를 생성하고 연속적인 이진 비트 패턴(binary bit pattern)들로 이루어진 이들의 경로서명값을 비트단위로 루트에서부터 아래쪽으로 연결된 각 PE들로

입력된 값을 순차적으로 전달한다.

넷째, 병렬 매치를 수행하기 위해 루트 PE에 매치 처리 수행 시작 신호를 입력한다.

최악의 경우에 있어서 기존의 기법은 검색을 위해 키의 최대 비교횟수로 시간 복잡도 $O(N/2^{l})$을 요구하고 있다. 그러나 PMP-indexing 기법은 검색을 위해 키의 비교횟수로 경로서명의 길이 h만큼만을 요구한다. 따라서 PMP-indexing 기법의 시간 복잡도는 $O(h)$이다.

본 논문에서 제안한 기법들은 경로서명들의 개수(N) 뿐만 아니라 경로서명에서 1로 설정된 비트의 수에 관계없이 오직 버킷의 슬롯 수(k)와 경로서명의 길이(h)에만 영향을 받기 때문에 일정한 검색 성능을 유지할 수 있으며, 소규모의 데이터베이스에서 사용하는 것보다는 중간규모 이상의 데이터베이스에서 사용하는 경우에 더 많은 기대효과를 얻을 수 있다.

본 논문은 다음과 같이 구성된다. 2장에서는 새로운 인덱싱 기법들의 연구에 필요한 관련 이론들의 서술과 함께 기존의 연구사례 소개 및 문제점을 제시한다. 3장에서는 본 논문에서 제안하고 있는 새로운 인덱싱 기법들에 대한 시스템 설계 및 실험 고찰 결과를 기술한다. 4장에서는 기존의 기법과 제안된 기법들에 대한 실험결과에 대한 고찰 및 그 성능을 비교 분석한 결과에 대한 내용을 기술한다. 마지막으로 5장에서는 결론 및 향후과제에 대하여 논의한다.

2. 인덱싱 기법 제안을 위한 관련 이론

본 장에서는 XML관련 기술들의 개략적인 설명과 함께 경로-지향 언어들의 종류와 특성, 서명과 경로서명, 트라이(Trie)와 확장성 해싱 기술 그리고 기존의 연구사례에 대한 소개 및 문제점에 관하여 기술한다.

2.1 XML관련 기술

XML은 SGML(Standard Generalized Markup Language)의 부분집합으로 설계되었으며 W3C에 의해 권장되고 있다. XML은 단순하고 매우 유연한 텍스트 형식을 가지고 있으며 이것의 기본 구문은 HTML과 유사하지만 그 목적은 다르다.

XML의 가장 큰 특징은 XML이 메타언어(Meta Language)라는 것이다. 이것은 XML이 HTML과 같이 어떤 문서를 기술하는 문서유형을 제공하는 것이 아니라 문서유형을 만드는 역할을 하기 때문이다. 즉, 문서유형에 사용될 요소를 선언하고 이 요소들 간의 관계를 기술하는 역할을 하는 언어라는 것이다 [18, 60].

XML은 원래 대규모 전자출판 문제를 해결하기 위한 목적으로 설계되었으나 현재 e-비지니스/전자상거래와 관련한 마크업 언어, 수학 공식, 화학 분자구조, 그래픽, 회계 데이터등과 같은 데이터를 마크업 하기 위해 사용되고 있다.

XML문서는 DTD(Document Type Descriptor) 또는 스키마(Schema)에 의

해 미리 기술된 구조와 태그 규칙을 따라야만 한다.

DTD는 특정 XML문서의 구조를 명시적으로 선언하여 제공함으로서 엘리먼트들 사이의 관계를 분명하게 나타내는 역할을 한다. 그러나 DTD는 XML이 아니므로 XML에 대해서 할 수 있는 것과 같은 방식으로 DTD를 다룰 수 없기 때문에 프로그램에서 필요한 기능들을 충분히 만족하지 못한다. 따라서 최근에는 DTD보다는 XML문서 구문을 사용하며 엘리먼트의 콘텐츠를 함께 표현하는 스키마를 XML문서 구조를 표현하는 주요 수단으로 사용하고 있는 추세이다. 두 가지 주요 모델로는 W3C XML 스키마와 마이크로소프트 XML 스키마가 있다.

문법에 맞게 작성된 XML문서를 '적정 문서(well-formed document)'라 하며, XML문서가 문법에 맞으면서 DTD 또는 스키마에 맞게 작성된 문서를 '유효한 문서(valid document)'라하고 이것을 체크하기 위한 프로그램을 파서(parser)라 한다. 그림 1은 마이크로소프트 XML 스키마 모델을 사용하는 유효한 XML문서의 코드를 보여주고 있다.

```
<?xml version="1.0" ?>
<CustomerList xmlns = "x-schema:customerlist-schema.xml">
   <Member type="normal">
     <Person>
       <name>Sally</name>
       <birthday> 30th January </birthday>
       <gender> Female </gender>
       <address> Pusan </address>
     </Person>
   </Member>
   <hobbies>
     <artist>Bach</artist>
     <sports> Baseball </sports>
     <music> Jazz</music>
     <favorite> Dancing </favorite>
    </hobbies>
</CustomerList>
```

그림 1. 유효한 예제 XML문서 코드
Fig. 1. Valid an example XML document code

대표적인 파서들로는 DOM(Document Object Model)-기반의 파서와 SAX(Simple API for XML)-기반의 파서가 있다.

DOM기반의 파서는 W3C의 표준 권고안으로서 객체-기반(Object-based)의 인터페이스를 가지고 있다. 따라서 메모리 상주 트리를 이용하기 때문에 응용 프로그램이 간단하며, XML문서를 트리(tree)와 같이 표현할 수 있다. 트리에서 노드의 형태는 각각 엘리먼트(Element)와 애트리뷰트(Attribute) 그리고 텍스트(Text)중에 한 가지이다 [18, 23, 73].

본 논문에서는 DOM파서를 사용한다. 그림 2는 DOM파서를 이용하여 그림 1의 예제 XML문서에 대한 파싱 결과를 트리 구조로 표현한 것이다.

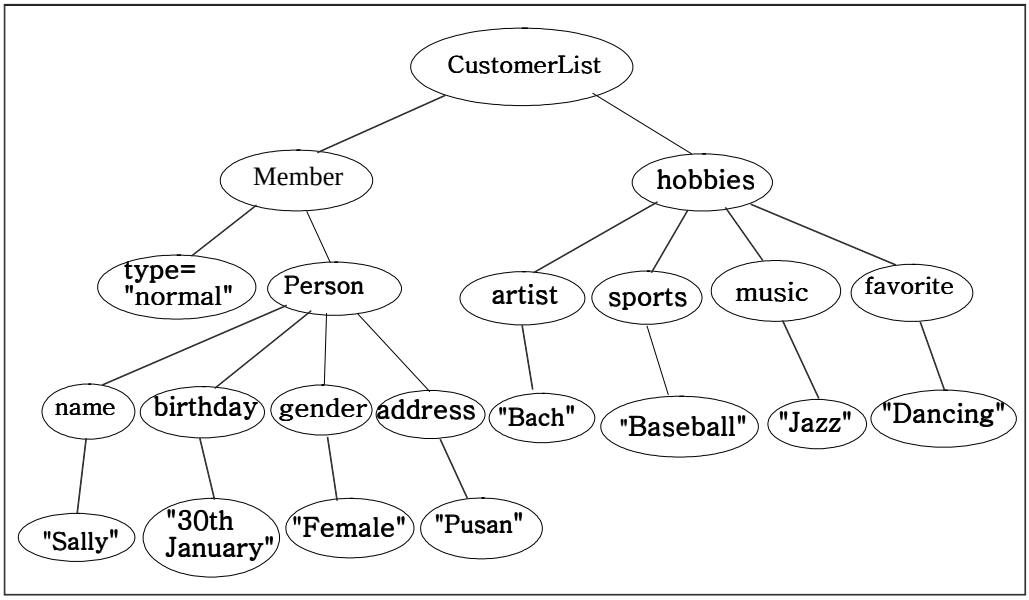


그림 2. XML문서의 트리구조

Fig. 2. Tree structure of an example XML document

반면에 SAX 기반의 파서는 W3C의 표준 권고안은 아니며 이벤트 기반

(Event-driven)의 인터페이스를 사용하고 있다. 이것은 XML문서를 처리하는 과정에서 태그, 텍스트, 주석 등을 만나게 되면 이벤트를 발생하기 때문에 DOM파싱보다 더 빠르다. SAX파서는 그 구조는 간단하지만 이벤트 핸들러를 처리하기 위한 응용 프로그램은 비교적 복잡하다 [18, 23, 16].

2.2 경로-지향 언어

트리 구조로 표현 가능한 XML문서를 질의하기 위하여 XQuery, XML-QL, XPath등과 같은 몇 가지 경로-지향 언어들이 제안되었다.

(1) XQuery

XQuery는 Quilt라 불리는 XML 질의 언어로부터 유도되었으며, XPath, XQL, XML-QL, SQL, OQL등과 같은 몇 가지 언어들로부터 그들의 특성들을 차용하였다. XQuery는 ORACLE을 비롯한 현재 모든 주요 데이터베이스 엔진들이 지원하고 있다.

XQuery는 명령라인으로 표현되어지며 경로 연산자(path operator)를 사용하여 엘리먼트 타입들을 연결한다. 경로 연산자에는 '/'와 '//'가 있다. 여기서 '/'는 자식노드를 구별하기 위한 연산자이며 '//'는 후손노드를 구별하는 연산자이다. 또한, 심볼 '@'를 사용할 수 있으며 이것은 애트리뷰트의 이름 앞에 사용한다. XQuery의 단순경로는 다음과 같이 Backus-Naur Form 문법으로 기술한다 [18, 69, 72].

그림 3은 XQuery의 문법구조를 표현한 것이다.

```
<Simple Path> ::= <PathOP><SimplePathUnit>|
                  <PathOP><SimplePathUnit> '@'<AttName>
<PathOP> ::= '/' | '//'
<SimplePathUnit> ::= <ElementType>|
<ElementType><PathOP><SimplePathUnit>
```

그림 3. XQuery의 문법구조
Fig. 3. Grammatical construction of XQuery

다음은 XQuery을 이용한 단순경로 질의어의 한 가지 예이다.

<SimplePathUnit> :
/CustomerList/Member/Person[gender$contains$'Female']

여기서 /CustomerList/Member/Person은 단순경로를 뜻하는 것이며, [gender$contains$'Female']는 술어(predicate)이다. 즉, 지정한 경로를 가진 엘리먼트 “gender”가 단어 “Female”을 포함하는지 아닌지를 질의한다.

(2) XML-QL

XML-QL은 질의어를 표현할 수 있으며, 본래는 XML문서들의 스타일과 레이아웃을 명세하기 위해 의도된 것이다. 또한, 이것은 XML문서들로부터 데이터의 조각들을 추출할 수 있으며, XML데이터와 DTD들 사이의 사상(mapping)과 다른 소스들로부터 XML데이터를 통합할 수 있다. XML-QL은 “construct” 라는 명령을 통해 새로운 XML문서를 만들어 낼 수 있다.

XML-QL은 조인과 같은 데이터 조작을 위한 특정 연산들을 가지고 있으며 XML데이터 변환을 지원한다. 즉, 새로운 문서생성과 문서의 부분을 추출하기 위한 트리-브라우징과 트리-변환 연산자를 제공한다. 그러나 XML-QL의 문

법은 아직까지도 개발 중에 있으며 현재의 버전도 완전한 것은 아니다. XML-QL의 문법 구조는 그림 4와 같다.

```
XML-QL ::= (Function | Query) <EOF>
Function ::= 'FUNCTION' <FUN-ID>'(' (<VAR>(':' <DTD>)?)* ')'
             (':' <DTD>)? Query 'END'
Query ::= Element | Literal |  <VAR> | QueryBlock
Element ::=  StartTag  Query  EndTag
StartTag ::= '<'(<ID>|<VAR>) SkolemID? Attribute* '>'
SkolemID ::= <ID> '(' <VAR> (',' <VAR>)* ')'
Attribute ::= <ID> '='  ('"' <STRING> '"'  |  <VAR> )
EndTag ::= '<' / <ID>? '>'
Literal ::= <STRING>
QueryBlock ::=  Where  Construct ('{' QueryBlock '}')*
Where ::= 'WHERE' Condition (',' Condition)*
Construct ::= OrderedBy? 'CONSTRUCT'  Query
Condition ::= Pattern BindingAs* 'IN' DataSource | Predicate
Pattern ::= StartTagPattern Pattern* EndTag
StartTagPattern ::= '<' RegularExpression Attribute* '>'
RegularExpression ::= RegularExpression '*' |RegularExpression '+'|
                      RegularExpression '.' RegularExpression |
                      RegularExpression '|' RegularExpression |
                      <VAR> | <ID>
BindingAs ::= 'ELEMENT_AS' <VAR>  | 'CONTENT_AS' <VAR>
Predicate ::= Expression OpRel Expression
Expression ::=  <VAR> | <CONSTANT>
OpRel ::= '<' | '<=' | '>' | '>=' | '=' | '!='
OrderedBy ::= 'ORDERED-BY' <VAR>+
DataSource ::= <VAR>|<URI>|<FUN-ID>(DataSource(',' DataSource)*)
```

그림 4. XML-QL의 문법구조

Fig. 4. Grammatical construction of XML-QL

XML-QL 표현을 사용한 질의어의 예는 다음과 같다.

```
WHERE <CustomerList><hobbies>
    <music> $a </>
    </> IN "예제.xml"
CONSTRUCT <검색결과> $a </>
```

위의 예제 질의어는 “예제.xml”이라는 XML문서로부터 “CustomerList” 엘리먼트의 자식 엘리먼트로서 “hobbies”를 가지고 있고, 그의 자식 노드인 “music” 엘리먼트의 값을 찾은 후 결과로 “검색결과”라는 엘리먼트 구조를 만들어 출력하라는 의미이다. 여기서 $a는 변수를 뜻한다. 결과로는 “<검색결과> Jazz </검색결과>”가 만들어진다.

(3) XPath

XPath의 기본 목적은 XML문서의 부분들을 다루기 위한 것이다. 또한, 문자열과 숫자, 불리언들을 조작하기 위한 기초 기능들을 제공한다. XPath는 XML문서내의 네비게이션을 명세하기 위한 언어이다. 주어진 XML문서에 대한 XPath 표현의 평가 결과는 문서의 순서에 따라 정렬된 노드들의 집합이다. XPath 표현은 그림 5와 같은 문법 구조를 사용한다.

```
LocationPath ::= RelativeLocationPath|AbsoluteLocationPath
AbsoluteLocationPath ::= '/' RelativeLocationPath?|
AbbreviatedAbsoluteLocationPath
RelativeLocationPath ::= Step| RelativeLocationPath
 '/' Step | AbbreviatedRelativeLocationPath
Path ::= /Step1/Step2/..../Stepn
Step ::= AxisSpecifier NodeTest Predicate* | AbbreviatedStep
```

그림 5. XPath의 문법 구조
Fig. 5. Grammatical construction of XPath

XPath 표현은 왼쪽 스텝부터 순차적으로 평가한다. XPath의 스텝은 단일노드에 적용된 결과노드들의 집합을 선택한다. 각각의 결과노드 집합은 다음 스텝 평가를 위한 컨텍스트 노드(Context node)로 사용된다. XPath 표현의 평가 결과는 마지막 스텝에서 선택된 노드집합들의 합집합이다. 스텝내의 AxisSpecifier는 문서에서 네비게이트 되어야 하는 방향을 나타내며 XPath는 child, descendant, parent, ancestor, following-sibling등 12가지의 방향을 제공한다. 축(Axes)에 의해 선택된 노드들은 노드 테스트에 의해 걸러진다. 가장 공통적으로 사용되는 노드 테스트는 노드의 이름이다. XPath는 대괄호([ ])로 둘러싸인 술어를 포함할 수 있으며, 술어는 위치경로의 일부로서 검색한 노드들을 걸러내는 부울식을 말한다.

XPath 표현을 사용한 질의어의 예는 다음과 같다 [18, 70].

/CustormerList/hobbies/activity[.='Dancing']/@class

여기서 /CustormerList/hobbies/activity는 스텝들이며 [.='Dancing']은 술어를 나타낸다. 또한, @class는 애트리뷰트이다. 즉, 지정한 경로를 가진 엘리먼트 activity의 애트리뷰트 class가 애트리뷰트값으로 'Dancing'을 가지고 있는지 아닌지를 질의한다.

2.3 서명과 경로서명

서명파일(Signature file)은 부정확한 여과장치(Inexact filter)의 개념을 기

반으로 하고 있다. 따라서 그들은 많은 부적합한 값들을 버리는 빠른 테스트를 제공한다. 검색조건에 적합한 값들은 확실히 테스트를 통과한다. 그러나 몇 개의 값들은 실제로는 검색 요구 조건을 만족하지 않는다 할지라도 우연히 여과장치를 통과 하게 되는 경우가 발생할 수 있으며 이들을 허위드롭(False drop)이라 한다.

그림 6은 허위드롭(False drop)의 예를 보여주고 있다.

문헌텍스트 : .... Management ..... database ..... information ....

대표단어들	대표단어의 서명	질의어	질의어의 서명	매치 결과
Management	010000100110	Management	010000100110	match
database	100010010100	XML	011000100100	no match
information	010100011000	informatik	110100100000	false drop
목표 서명	110110111110			

그림 6. 경로서명 생성과 허위드롭(False drop)의 예
Fig. 6. Signature generation and an example of false drop

위의 그림 6에서 보는 것과 같이 'informatik'이란 질의어는 실제로 문서 텍스트 내부에 존재하지 않는 단어라 할지라도 우연히 목표서명값과 매치되어 질의어의 결과값으로 나타난다. 따라서 이것은 허위드롭(False drop)이 된다.

허위드롭(False drop)의 가능성을 최소화하기 위해서는 각 서명값에서 1로 설정된 비트의 수가 거의 50%를 포함하는 경우인 것으로 드러났다. 따라서 최적의 설계 하에서 다음과 같은 식(1.1)과 식(1.2)를 얻을 수 있다.

$$F_d=2^{-m} \quad (1.1)$$

$$Fln2=mD \quad (1.2)$$

위의 식에서 F_d는 허위드롭(False drop)의 가능성을 나타내며 F는 해시-코드화 된 서명의 비트 길이를 그리고 m은 1로 설정된 비트의 수, D는 비일상 용어의 수를 나타내고 있다.

서명은 1로 설정된 n개의 비트를 가진 길이 h인 해시-코드화된 이진 비트 패턴들이며 서명파일 안에 저장된다. 이들 서명을 생성하기 위한 방법으로는 전형적으로 중첩 기호법(Superimposed ORing)을 사용한다.

어떤 경로-지향 질의어가 도착했을 때 먼저 질의어 안에 포함된 경로에 대한 경로서명을 생성한 후 이것을 포함하고 있는 모든 엘리먼트들을 경로서명 파일로부터 검색한다. 이 과정에서 많은 부적합한 엘리먼트들은 버려지게 되고 적합한 엘리먼트들은 허위드롭(False drop)을 제거하기 위해 체크되어지거나 사용자에게 검색결과로서 되돌려 준다.

이 서명파일의 사용 목적은 대부분의 부적합한 엘리먼트들을 가려내는 것이다. 즉, 질의어의 서명과 매치되지 않는 서명은 동등한 엘리먼트가 무시 되어질 수 있다는 것을 보증한다. 따라서 불필요한 엘리먼트들의 액세스를 금지하게 된다. 또한, 서명파일은 역 인덱스 방법보다 훨씬 더 낮은 공간 오버헤드와 단순한 파일구조를 가진다.

위의 필터링의 아이디어는 유사한 방법으로 경로서명(Path Signature)을 설정하는 것으로 경로-지향 질의어를 지원하기 위해 활용될 수 있다 [18, 35]. 그림 1의 예제 XML문서에 대한 트리구조 표현에서 엘리먼트 “Person”의 경로서명값을 생성하기 위한 예는 그림 7과 같다.

Element node	Signature
CustomerList	010 000 100 110
Member	100 010 010 100
Person	∨ 010 100 011 000
Path Signature	110 110 111 110

그림 7. 경로서명 생성 예제

Fig. 7. An example of path signature generation

2.4 트라이와 확장성 해싱 기술

트라이는 스트링(string)들의 집합을 저장하기 위한 디지털 트리(digital tree)이며, “retrieval”이란 단어의 중간 부분을 인용한 것이다. 이것은 키 검색을 위해 키값을 직접적으로 표현하지 않고 키를 구성하는 문자나 숫자의 순서로 키값을 표현한다. 그러므로 트라이의 차수는 키값 표현에 사용하는 문자의 수인 기수(radix)에 의해 결정된다. 따라서 트라이는 기수 검색(radix searching)이라고도 불리는데, 이는 탐색 트리의 분기 계수(branching factor)가 키의 각 위치에 나타날 수 있는 서로 다른 기호들의 개수와 같기 때문이다. 트라이는 외부노드인 정보노드(information node)와 중간노드인 분기노드(branch node)로 구성된다. 모든 정보는 외부노드에만 저장되며 분기노드는 정보는 없고 링크만을 가진다.

다른 검색 트리들이 입력되는 노드의 순서에 따라 모양이 변하는 것과 달리 트라이는 입력순서에 관계없이 항상 같은 모양을 나타낸다. 트라이의 정보노드는 서로 구별할 수 있는 자리수 만큼의 깊이(depth)에 위치를 하고 있다 [2, 13].

확장성 해싱에서의 아이디어는 기존의 해싱을 트라이라는 검색방법과 결합하는 것이다. 확장성 해싱의 기본적인 접근방법은 기수가 2인 트라이를 이용한다. 검색 결정은 비트 대 비트 단위로 일어난다 [2, 13].

트라이가 외부노드에 한 개의 정보만을 저장하는데 비해 확장성 해싱의 외부노드는 다수의 정보를 저장할 수 있는 버킷을 가진다.

버킷은 다수의 정보를 저장하기 위해 각 버킷마다 고정된 수의 슬롯(slot)을 가진다. 슬롯의 수에 따라 한 개의 버킷에 저장될 수 있는 최대 정보의 수가 결정된다.

확장성 해싱에서 각 외부노드는 실제 데이터의 검색키가 저장된 버킷의 주소를 저장한다. 확장성 해싱의 구성요소는 디렉토리와 버킷(directory, bucket)의 쌍으로 구성되며 디렉토리의 확장이나 축소 그리고 페이지 분할, 병합을 통하여 동적인 검색 구조를 형성한다.

확장성 해싱을 구성하기 위한 과정은 다음과 같다.

첫째, 각 키에 대한 해시코드를 구한다.

둘째, 구해진 해시코드를 이용하여 이진 트라이를 구성한다.

셋째, 이진 트라이를 완전 이진 트리(complete binary tree)로 변형한다.

넷째, 완전 이진 트리를 디렉토리로 변환한다.

어떤 해싱 시스템에서 핵심 이슈는 버킷에 오버플로가 일어났을 때 어떻게 처리하는가에 대한 문제이다. 확장성 해싱 기술은 오버플로가 발생한다면 오버플로와 관련한 주소 공간을 증가시키는 방법을 사용한다. 즉, 오버플로가 발생하는 경우에 이것을 처리하기 위해 버킷을 분할하거나 디렉토리를 확장하는 방법들 중에서 한 가지를 사용한다.

2.5 기존 연구사례 소개 및 문제점 제시

관계형 데이터베이스에 저장된 XML문서들을 대상으로 경로서명을 이용하여 경로-지향 질의어의 평가속도를 개선하기 위한 기존의 대표적인 연구사례로는 Chen [18]에 의해 연구된 기법이 있다. 이 기법은 경로-지향 질의에 대한 평가 성능을 향상시키기 위하여 경로서명 파일과 패트리샤 트리 기술을 결합하는 방법을 제안하고 있다.

패트리샤 트리는 이진 디지털 트리(binary digital tree)로서 내부노드에는 분기를 위한 각 비트의 위치를 결정하는 값을 저장하고 각 외부노드에 한 개의 정보만을 저장한다 [18].

그림 8 (a)와 (b)는 예제 경로서명 파일과 이에 대한 패트리샤 트리의 구성을 나타낸 것이다. 그림 8 (b)의 패트리샤 트리구조에서 내부노드에 표시된 숫자는 입력된 경로-지향 질의어의 경로서명값에서 검사되는 비트의 위치를 나타내고 있다. 그리고 외부노드의 값은 경로서명 파일에서 각 경로서명의 위치 주소를 가리킨다.

검색 연산을 수행하기 위한 매치 테스트는 패트리샤 트리를 운행하는 동안 다음과 같은 순서로 수행된다.

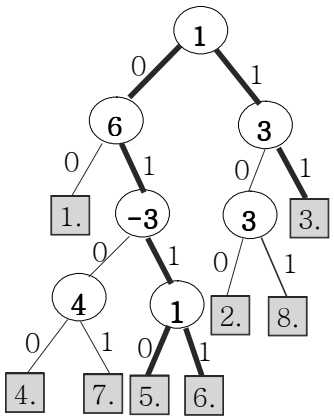
i) 패트리샤 트리의 각 노드의 값을 이용하여 검사할 비트의 위치(i-번째)를 구한다.
ii) 질의어 경로서명값의 i-번째 위치한 비트의 값이 1이면 오른쪽 자식노드를 따라 이동한다.

iii) 질의어 경로서명값의 i-번째 위치 비트의 값이 0이면 왼쪽과 오른쪽 자식노드 모두로 이동한다.

위치	경로	경로서명
1.	letter :	011 001 000 101
2.	letter-date :	111 011 001 111
3.	letter-greeting :	111 101 010 111
4.	letter-body :	011 001 101 111
5.	letter-closing :	011 101 110 101
6.	letter-sig :	011 111 110 101
7.	letter-body-para :	011 001 111 111
8.	letter-body-para-emph :	111 011 111 111

(a) 예제 경로서명 파일

(a) An example of path signature file



(b) 패트리샤 트리

(b) Patricia tree

그림 8. 예제 경로서명 파일과 패트리샤 트리

Fig. 8. An example of path signature file and patricia tree

예를 들어, 검색을 위해 입력되는 경로-지향 질의어의 경로서명값이 '000 100 100 000'이라고 가정하자. 검색을 위한 테스트는 그림 8의 (b)에서 굵은 선으로 표시된 경로를 따라서 이동하여 만나게 되는 각 단말노드의 값이 결과로 반환된다. 즉, 패트리샤 트리구조에서 질의어의 경로서명값의 첫 번째 위치의 비트값이 0이기 때문에 양쪽 자식노드를 따라 이동한다. 다음 조사 위치는 4번째(1+3=4)와 7번째(1+6) 위치이다. 4번째 위치의 비트값은 1이므로 오른쪽 자식노드를 따라 이동하여 단말노드에 도착하게 되어 이것을 검색 결과로 반환한다.

반면에, 7번째 위치의 비트값은 1이므로 오른쪽 자식 노드를 따라서 이동을 하게 되고, 이번에는 4번째(1+6-3=4) 위치의 비트값을 조사한다. 4번째 비트의 값이 1이기 때문에 오른쪽 자식노드를 따라서 이동하게 되고 이번에는 5번째(1+6-3+1=5) 위치의 비트값을 조사한다. 5번째 위치의 비트값이 0이므로 양쪽 자식노드를 따라 이동을 하면 마지막 단말노드에 도착하게 되고 단말노드값들이 검색 결과로 반환된다.

결과적으로 기존의 기법은 입력되는 질의어의 경로서명 해시-코드가 포함하는 1의 비트수와 경로서명 파일에 저장된 경로서명의 엔트리 수에 따라 검색 성능에 많은 영향을 미친다. 만약 1의 비트 수가 많아지면 검색을 위한 비교횟수는 적어지고 1의 비트의 수가 적어진다면 비교횟수는 상대적으로 많아진다. 또한, 입력되는 인덱스 키의 수가 많아짐에 따라 패트리샤 트리의 구조 확장이 요구된다.

최악의 경우에 있어서 기존의 기법은 검색연산을 위하여 키의 최대 비교횟수로 $O(N/2^{l})$ 요구하고 있다. 여기서 N은 경로서명 파일이 포함하는 경로서명의 수이며 l은 입력되는 질의어의 경로서명값에서 2진수 1로 설정된 비트들

의 개수이다. 따라서 N의 수가 커지면 키의 비교횟수도 증가하게 된다.

본 논문에서는 경로-지향 질의어 평가 성능을 개선하기 위한 2가지 새로운 인덱싱 기법들을 제안하고자 한다. 제안된 기법들은 각각 단축경로 파일과 확장성 해싱 인덱싱 구조를 결합하는 기법과 경로서명 파일과 병렬 매치 인덱싱 구조를 결합하는 기법들이다.

3. 새로운 인덱싱 기법 제안

본 장에서는 본 논문에서 제안하고자하는 두 가지 새로운 인덱싱 기법인 단축경로 파일과 확장성 해싱 인덱싱 구조를 결합한 EHP-indexing 기법과 경로서명과 병렬 매치 인덱싱 구조를 결합한 PMP-indexing 기법에 관한 내용을 기술한다.

3.1 EHP-indexing 기법

본 절에서는 EHP-indexing 기법에 대하여 기술한다.

3.1.1 EHP-indexing 기법을 위한 시스템의 설계

본 논문에서 제안하고 있는 첫 번째 인덱싱 기법은 단축경로와 확장성 해싱 인덱싱 구조를 결합한 EHP-indexing 기법이다. 그림 9는 EHP-indexing 기법을 수행하기 위해 필요한 논리적 시스템의 구성도를 나타낸 것이다.

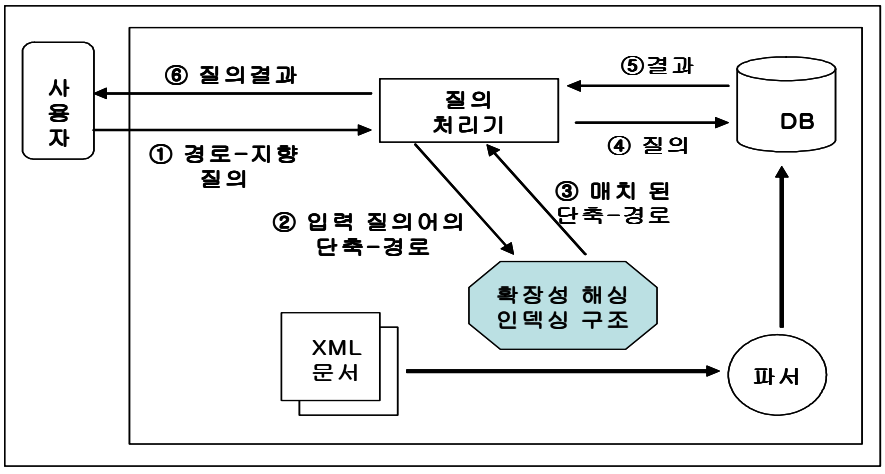


그림 9. EHP-indexing 기법의 논리적 시스템 구성도
Fig. 9. Logical architecture of the system for EHP-indexing method

그림 9의 논리적 시스템 구성도에서 질의처리기(Query processor)는 사용자가 입력한 경로-지향 질의어를 실행 가능한 관계형 질의어 형태로 변환하는 작업과 이들의 단축경로를 생성하는 작업 그리고 질의어 처리결과를 사용자에게 전달하기 위해 적절한 출력양식을 만들어 주는 기능을 수행한다.

확장성 해싱 인덱싱 구조는 입력 질의어의 단축경로와 매치되는 단축경로를 단축경로 파일로부터 검색하는 작업을 수행한다. 또한, 키의 삭제연산을 수행할 때 버킷으로부터 키의 삭제작업을 수행하게 되며, 필요에 따라서 버킷의 통합과 디렉토리를 축소하는 작업을 수행하기도 한다.

파서는 모든 XML문서를 파싱하여 관계형 데이터베이스에 엘리먼트, 텍스트, 애트리뷰트 테이블로 각각 분리 저장하는 역할을 한다.

엘리먼트 테이블의 구조는 {DocID:<Integer>, EID:<Integer>, EName:<string>, PID:<integer>, AbbPath:<string>}으로 구성된다. 여기서 DocID 필드는 문서의 식별자를, EID 필드는 엘리먼트의 식별자를 Ename 필드는 엘리먼트의 이름을, PID 필드는 각 엘리먼트의 부모 엘리먼트에 대한 식별자를, AbbPath 필드는 각 엘리먼트에 대한 단축경로값을 의미한다.

텍스트 테이블의 구조는 {DocID : <integer>, PID : <integer>, TextVal : <string>}으로 구성된다. 여기서 PID 필드는 엘리먼트 테이블에 저장된 엘리먼트들의 식별자를 위한 것이며, 이것은 원래의 XML문서 내에서 대응하는 텍스트값을 가진다. 또한, 텍스트는 항상 부모 노드로서 엘리먼트를 가진다는 것에 주목해만 한다. TextVal 필드는 텍스트 데이터값을 의미한다.

애트리뷰트 테이블의 구조는 {DocID:<integer>, PID:<integer>, AttName : <string>, AttVal:<string>}으로 구성된다. 여기서의 PID 필드도 텍스트 테이블에서와 같이 엘리먼트 테이블에 저장된 엘리먼트들의 식별자들이며 대응

하는 애트리뷰트값을 가진다. AttVal 필드는 애트리뷰트값을 의미한다.

그림 1의 예제 XML문서가 파서에 의해 파싱된 후 관계형 데이터베이스 내의 엘리먼트 테이블, 텍스트 테이블, 애트리뷰트 테이블에 각각 저장되는 형태는 다음의 표 1, 표 2, 표 3과 같다.

표 1. 엘리먼트 테이블

Table 1. Element table

DocID	EID	EName	PID	AbbPath
1	1	CustomerList	0	C
1	2	Member	1	CM
1	3	Person	2	CMP
1	4	name	3	CMPn
1	5	birthday	3	CMPb
1	6	gender	3	CMPg
1	7	address	3	CMPa
1	8	hobbies	2	Ch
1	9	artist	8	Cha
1	10	sports	8	Chs
1	11	music	8	Chm
1	12	favorite	8	Chf

표 2. 텍스트 테이블

Table 2. Text table

DocID	PID	TextVal
1	4	Sally
1	5	30th January
1	6	Female
1	7	Pusan
1	9	Bach
1	10	Baseball
1	11	Jazz
1	12	Dancing

표 3. 애트리뷰트 테이블
Table 3. Attribute table

DocID	PID	AttName	AttVal
1	2	type	"normal"

3.1.2 EHP-indexing 기법을 위한 구조 생성

본 절에서는 단축경로를 생성하는 방법과 단축경로 파일을 이용하여 EHP-indexing 기법을 위한 인덱싱 구조를 형성하는 방법에 대하여 설명한다.

단축경로는 XML문서의 파싱과정에서 각 엘리먼트들의 경로 정보를 고려하여 생성한다. 즉, 루트 엘리먼트에서 각 엘리먼트에 이르는 경로 상에 존재하는 모든 엘리먼트들의 첫 번째 머리글자만을 추출하고 이들을 결합(concatenation)하는 것으로 각 엘리먼트의 경로서명값을 생성한다. 본 논문에서는 이것을 단축경로라 부르고 이들을 단축경로 파일에 저장한다 [32]. 표 4는 그림 1의 예제 XML문서에 대한 단축경로 파일의 예를 보여주고 있다.

표 4. 단축경로 파일의 예
Table 4. An example of abbreviation path file

위치	Element들의 경로	단축경로	해시코드(16 bit)
1	CustomerList	C	0001101000101100
2	CustomerList/Member	CM	0001101001111001
3	CustomerList/Member/Person	CMP	0011100110111001
4	CustomerList/Member/Person/name	CMPn	0011101000100111
5	CustomerList/Member/Person/birthday	CMPb	0011101000011011
6	CustomerList/Member/Person/gender	CMPg	0011101000100000
7	CustomerList/Member/Person/address	CMPa	0011101000011010
8	CustomerList/hobbies	Ch	0001101010010100
9	CustomerList/hobbies/artist	Cha	0100000001111000
10	CustomerList/hobbies/sports	Chs	0100011110000000
11	CustomerList/hobbies/music	Chm	0100010100101000
12	CustomerList/hobbies/favorite	Chf	0010000001111000

예를 들어, 그림 1의 예제 문서에서 엘리먼트 Person의 경로는 CustomerList/Member/Person이다. 따라서 이 엘리먼트의 단축경로는 “CMP”가 생성된다. 경로서명의 길이를 16비트라 가정한다면 실제 인덱싱 구조를 형성할 때에는 이것의 해시 코드값 “0011 1001 1011 1001”을 인덱스 키로 사용한다.

그림 10은 EHP-indexing 기법의 인덱싱 구조를 구성하기 위한 첫 번째 중간단계로서 표 4의 단축경로 파일에 대한 이진 트라이 구조를 표현한 것이다.

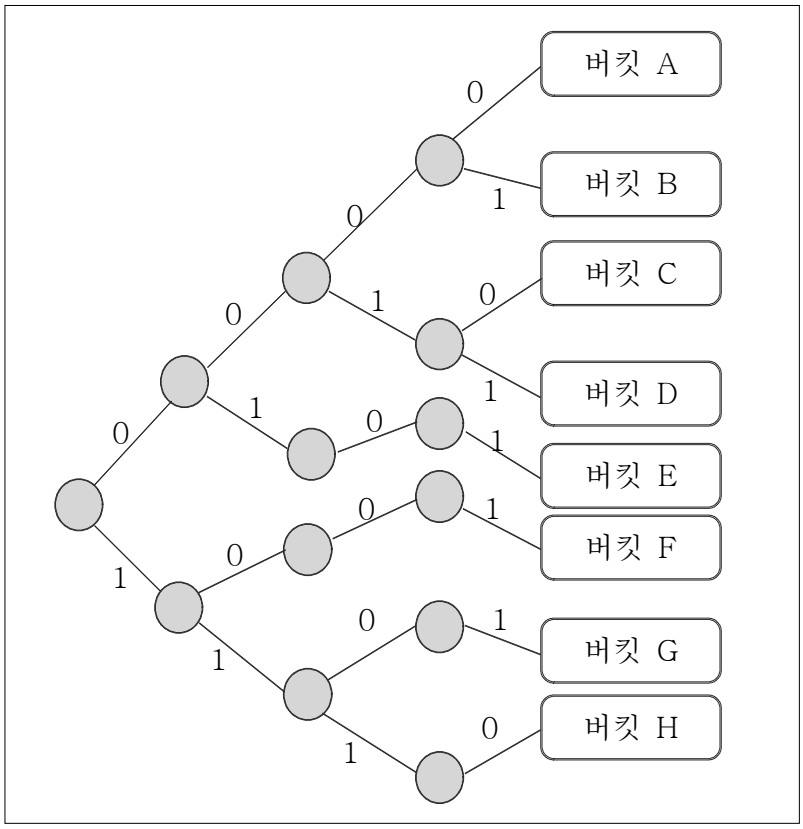


그림 10. 단축경로 파일에 대한 이진 트라이 구성
Fig. 10. Construction of binary trie for abbreviation path file

다음 그림 11은 확장성 해싱 인덱싱 구조를 구성하기 위한 두 번째 중간단계로서 그림 10의 이진 트라이 구조에 대한 완전 이진 트리 구조를 표현한 것이다.

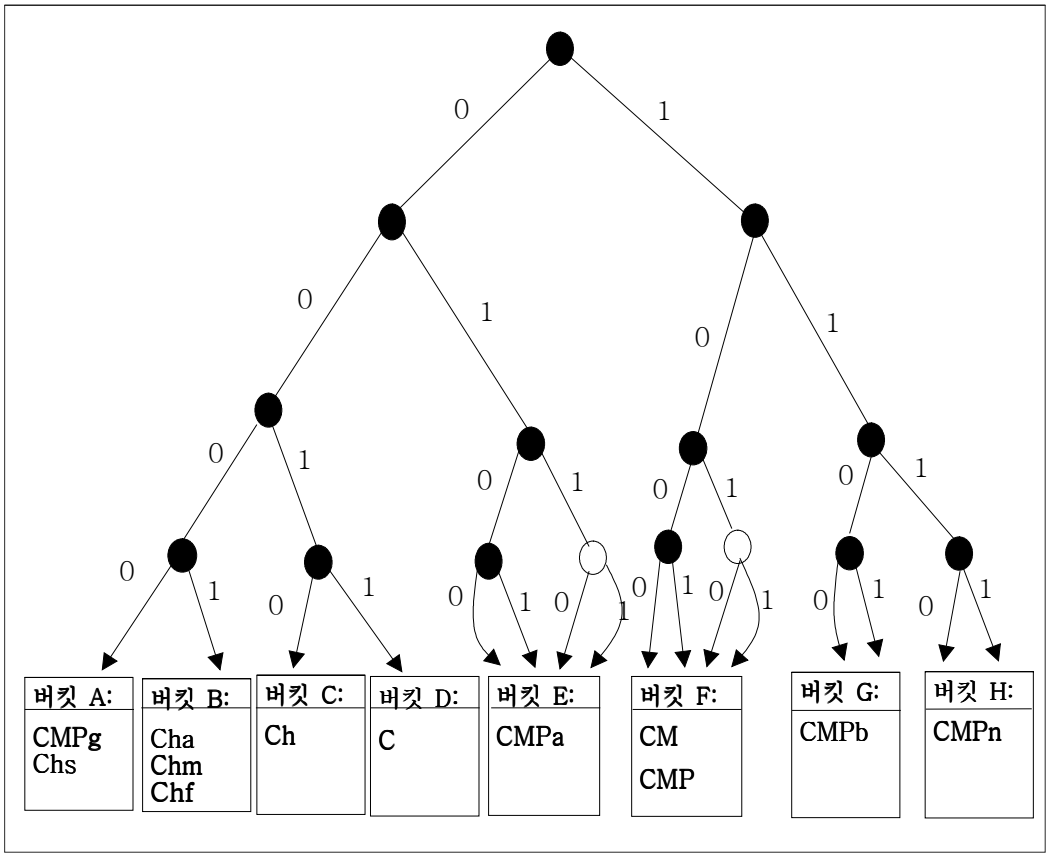


그림 11. 이진 트라이에 대한 완전 이진 트리 구조의 표현

Fig. 11. Representation of complete binary tree structure for binary tire

본 논문에서는 확장성 해싱 인덱싱 구조를 구성하기 위한 디렉토리 주소를 생성할 때 인덱스 키의 해시 코드를 오른쪽 비트부터 추출하여 역순(reversal order)으로 만들어 사용한다. 그 이유는 해시코드의 비트값 분포가 보다 다양해지므로 인덱스를 구성할 때 오버플로 횟수가 적어지기 때문이다.

그림 12는 그림 11의 완전 이진 트리를 확장성 해싱 구조로 변환한 것이다. 그림 12에서 우리는 한 개의 버킷에 최대 저장할 수 있는 키의 수를 3으로 한다.

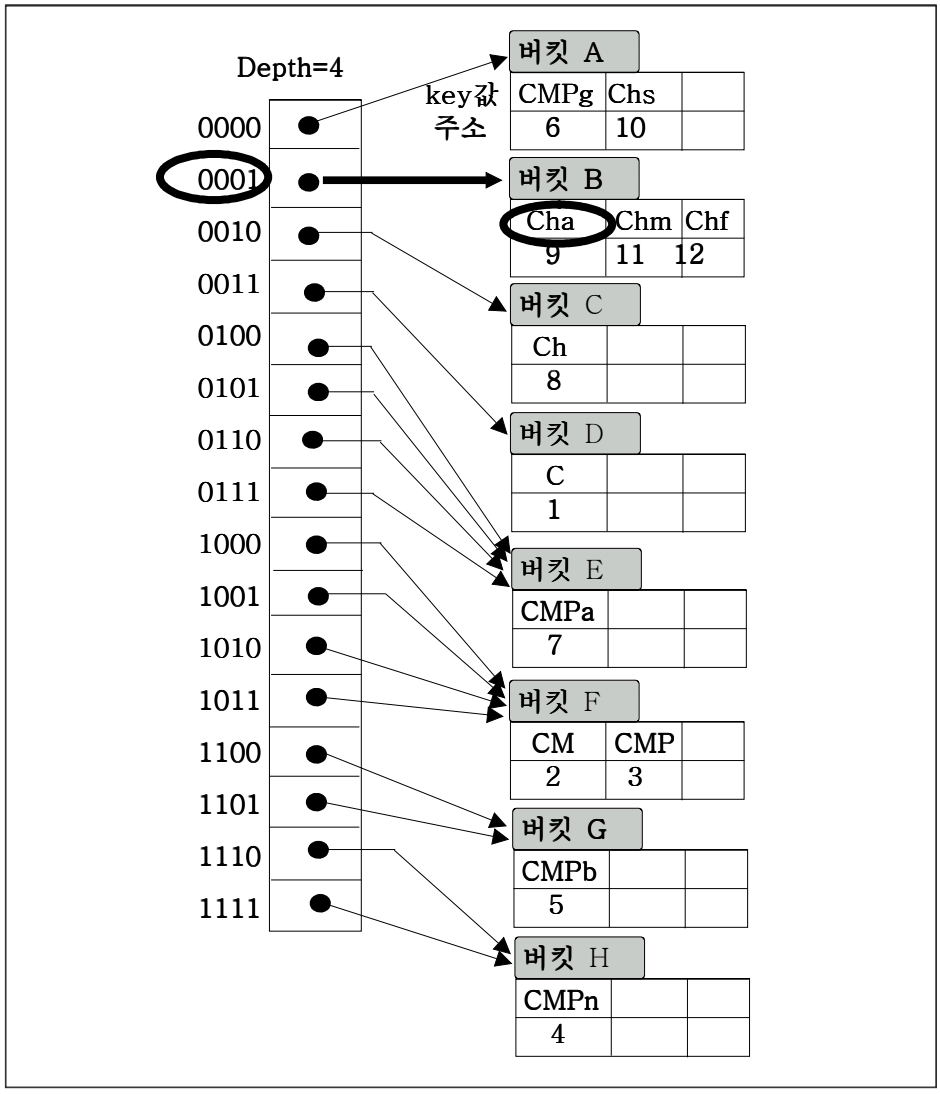


그림 12. 완전 이진 트리에 대한 확장성 해싱 인덱싱 구조

Fig. 12. Extendile Hashing Indexing structure for complete binary tree

예를 들어 /CustomerList/hobbies/artist란 경로-지향 질의어 입력에 대한 검색 수행은 그림 12에서 보이는 것과 같이 굵은 선을 따라 간다. 먼저 입력된 경로-지향 질의어에 포함된 경로의 단축경로 Cha가 만들어지고 이것에 대한 해시코드(0100010100101000)를 생성한다. 디렉토리의 깊이는 4이다. 따라서 해시코드의 오른쪽 4비트를 추출하여 역순으로 만들면 디렉토리 주소는 0001이 된다. 해당 디렉토리 주소에 기억된 포인터가 지시하는 버킷을 찾아 버킷에 저장된 데이터들을 순차적으로 모두 비교 검색하여 그 결과로 반환되는 것은 Cha의 주소가 된다.

만약 서로 다른 엘리먼트들의 경로에 대해 동일한 단축경로가 발생하는 경우에는 버킷에는 하나만 저장된다.

질의처리기는 검색된 결과를 이용하여 다음과 같은 관계형 질의를 생성한다.

```
SELECT * FROM ElementTable e
    WHERE e.Ename='artist'
    AND e.AbbPath matches 질의어의 단축경로;
```

여기서 “matches”는 입력된 경로-지향 질의어의 단축경로와 매치되는 단축경로를 찾기 위한 작업을 수행하는 함수이다.

위의 질의 수행결과 엘리먼트 테이블에서 단축경로가 Cha인 레코드는 모두 검색된다. 따라서 실제 검색 조건을 만족하지 않는 레코드라 할지라도 동일한 단축경로를 가지는 경우에도 질의처리 결과로 나타나게 된다. 이것은 본 논문에서 제안하고 있는 인덱싱 기법의 개선사항 중의 하나이다.

만약 질의가 /CumstomerList/hobbies[artist$containes$'Bach']의 형태로 입력된다면 질의처리기는 다음과 같은 질의를 생성한다.

```
SELECT * FROM ElementTable e TextTable t
    WHERE e.Ename='artist'
            AND e.AbbPath matches 질의어의 단축경로
            AND  e.DocID=t.DocID
            AND e.EID=t.PID AND t.TextVal⊇'Bach';
```

3.1.3 EHP-indexing 기법을 위한 알고리즘

본 논문에서 제안하고 있는 EHP-indexing 기법은 크게 세 가지의 알고리즘 구성요소들에 의해 수행된다. 이들은 각각 파서에 의해 파싱된 XML문서를 처리하기 위한 문서파싱처리 알고리즘, 질의어 처리를 위한 질의처리기 알고리즘 그리고 인덱싱 구조 생성과 매치 처리 그리고 키의 삭제연산을 수행하기 위한 인덱싱 구조 알고리즘이다. 먼저 문서파싱처리 알고리즘은 파서에 의해 파싱된 XML문서의 각 구성요소들을 관계형 데이터베이스 내부에 해당 테이블에 분리하여 저장한다. 즉, 파싱과정에서 엘리먼트 노드는 엘리먼트 테이블에, 애트리뷰트 노드는 애트리뷰트 테이블에 텍스트 노드는 텍스트 테이블에 각각 분리하여 저장한다. 또한, 각 엘리먼트 노드의 단축경로를 생성하고 이것을 엘리먼트 테이블과 단축경로 파일에 저장하는 기능도 함께 수행한다. 본 논문에서는 DOM 파서를 사용하여 XML문서의 파싱을 수행하였으며, XML문서 파싱 처리를 위한 알고리즘은 그림 13과 같다.

```
/* level : level of Current Element node or level of Root Element node */
/* array IDTrace use to trace a path of each element */
/* array PathTrace use to get a abbreviation path of each element node */
 XML_Document_Parsing(CurrentNode, level) {
 Input :  XML document, level
 Output : Abbreviation Path File, Element table, Attribute table, Text table
 Current_NodeType=GetNodeType();
 switch(Current_NodeType) {
   case DOCUMENT_NODE :
        Document d=(Document) node;        /*  create XML document object */
        XML_Document_Parsing(RootElementNode, level+ 1);
        break;
   case ELEMENT_NODE:
         ElementID+ + ;
         IDTrace[level-1]=ElementID;
         PathTrace[level-1]=ElementName;
   for(int loop=1 to level) {              /* to get abbreviation path of each  element */
        Create Abbreviation Path of each element;
      }
   if(level==1) {
        ParentID=0;
   } else {
        ParentID=IDTrace[level-2];
        Insert into ElementTable
            values(DocID, ElementID, ElementName, ParentID, AbbreviationPath);

        Store Abbreviation Path into Abbreviation Path File;
        Get Attributes of CurrentElement;
        for(int loop=1 to Number of AttributeNode) {
             XML_Document_Parsing(AttributeNode, level+ 1);
        }
        Get ChildNoes of CurrentNode;
        if(ChildNode !=null){
          for(int loop=1 to Number of ChildNode) {
               XML_Document_Parsing(ChildNode, level+ 1);
          }
        }
     }
        break;
   case ATTRIBUTE_NODE:
         ParentID=ElementID;
         Insert into AttributeTable
              values(DocID, ParentID, AttributeNodeName(), AttributeNodeValue);
         break;
   case CDATA_SECTION_NODE :
   case TEXT_NODE:
         ParentID=ElementID;
         Get value of Current node;
         Insert into TextTable values(DocID, ParentID, TextNodeValue);
 }
  }
```

그림 13. XML문서 파싱 및 단축경로 파일 생성 알고리즘

Fig. 13. An algorithm for parsing of XML document and generation of abbreviation path file

다음으로 질의처리기 알고리즘은 다음과 같은 기능들을 수행한다.

i) 사용자가 입력한 경로-지향 질의어의 단축경로를 생성한 후 이것을 확장성 해싱 인덱싱 구조로 전달한다.
ii) 확장성 해싱 인덱싱 구조를 사용하여 얻어진 매치된 결과값을 이용하여 경로-지향 질의어를 관계형 질의어로 변환한 다음 관계형 데이터베이스에 대해 질의를 수행한다.
iii) 관계형 질의어 수행 결과로 얻어진 결과값을 사용자가 알 수 있는 적당한 포맷으로 변환하여 사용자에게 최종 결과값으로 반환한다.

마지막으로 확장성 해싱 인덱싱 구조 알고리즘은 3가지의 부속 알고리즘들로 구분된다. 그것은 인덱싱 구조 생성을 위한 알고리즘, 키의 삭제처리를 위한 알고리즘, 검색연산을 위한 매치 처리 알고리즘으로 구성된다. 이 알고리즘은 다음과 같은 기능들을 수행한다.

i) 파싱 과정에서 생성된 단축경로 파일을 이용하여 데이터베이스에 저장된 모든 엘리먼트들에 대한 확장성 해싱 인덱싱 구조를 생성한다. 이 과정 수행 이후에 디렉토리 파일과 버킷 파일이 생성된다.
ii) 질의처리기로부터 전달 받은 이슈화 된 경로-지향 질의어의 단축경로에 대한 해시코드값을 계산한다.
iii) 생성된 해시코드값을 이용하여 정보검색 연산을 수행한 다음 그 결과값 즉, 매치 연산을 수행한 후 그 결과값을 질의처리기로 반환한다.
iv) 만약 키의 삭제 연산이 요청된다면 해당키를 버킷으로부터 삭제하는 연산을 수행한다. 이것은 경우에 따라서 버디버킷의 통합 작업 수행이나 디렉토리 축소작업을 수행하기도 한다.

위의 알고리즘들 중에서 세 번째 알고리즘인 확장성 해싱 인덱싱 구조 알고리즘이 가장 중요한 역할을 수행한다.

그림 14는 확장성 해싱 인덱싱 구조 알고리즘에 사용된 클래스들의 UML 참조 다이어그램을 표현한 것이다.

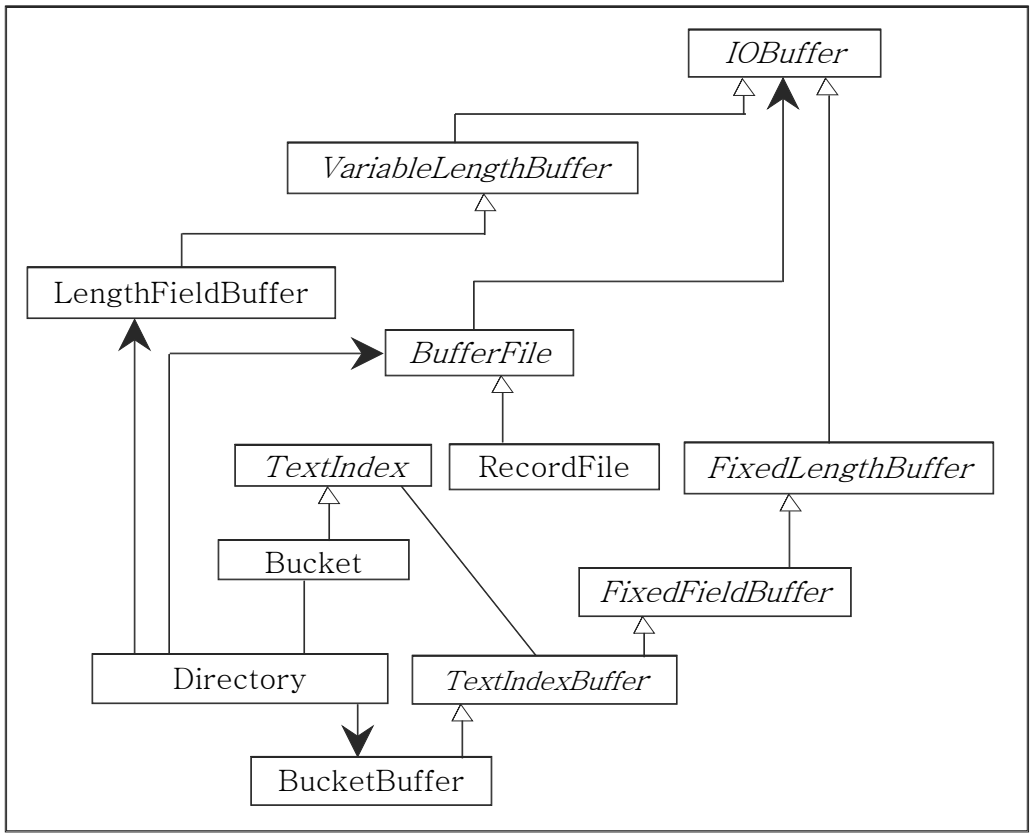


그림 14. EHP-indexing 기법을 위한 알고리즘들의 클래스 참조 다이어그램
Fig. 14. UML class reference diagram of algorithms for EHP-indexing method

확장성 해싱 인덱싱 구조 알고리즘은 실행 후 2개의 파일이 생성된다. 그들은 각각 디렉토리 파일과 버킷 파일이다. 디렉토리 파일은 버킷들의 주소를 참조하기 위한 정보를 저장하며, 버킷 파일은 특정 키들의 집합 및 키들과 관

련된 정보를 포함한다.

위의 참조 클래스들의 역할에 대해 간단히 설명하면 다음과 같다.

Directory 클래스는 다수의 셀들로 구성되어 있으며 이들 각 셀들은 각 버킷의 주소와 관련한 정보를 저장하고 있다.

Bucket 클래스는 버킷과 관련된 정보를 유지하기 위한 클래스이다. 이 클래스는 TextIndex 클래스로부터 유도된 클래스로서 Directory 클래스와 연결되어 있으며, 이것은 Directory 클래스를 통해서만 모든 멤버에 대한 접근이 가능하다. BufferFile 클래스는 파일의 입출력 연산을 제공하기 위해 사용된다. LengthFieldBuffer 클래스는 최대 필드들 중 최대값을 가지는 필드 길이에 해당하는 버퍼 생성을 지원한다.

BucketBuffer 클래스는 버킷들을 위한 버퍼를 제공하는 역할을 하며, TextIndexBuffer 클래스는 FixedFieldBuffer 클래스로부터 유도된 클래스로서 인덱스 객체들의 읽기, 쓰기를 지원한다. VaribaleLengthBuffer 클래스는 가변 길이 레코드들의 읽기와 쓰기를 지원하는 클래스이며, FixedFieldBuffer 클래스는 고정된 크기 필드들을 위한 pack과 unpack연산을 지원하기 위한 클래스이다. TextIndex 클래스는 버킷에 인덱스 키를 추가하기 위한 연산을 지원하며, RecordFile 클래스는 템플레이트 클래스로서 읽기, 쓰기, 파일의 생성, 열기, 닫기 연산을 지원한다. FixedLengthBuffer 클래스는 IOBuffer 클래스의 서브클래스이며, 고정길이 레코드들의 읽기와 쓰기를 지원한다. IOBuffer 클래스는 VariableLengthBuffer 클래스와 FixedLengthBuffer 클래스들의 추상기본 클래스로서 팩된 버퍼들의 입출력을 지원한다.

그림 15는 확장성 해싱 인덱싱 구조를 이용하여 검색연산을 수행하는 과정을 UML 순차 다이어그램으로 표현한 것이다.

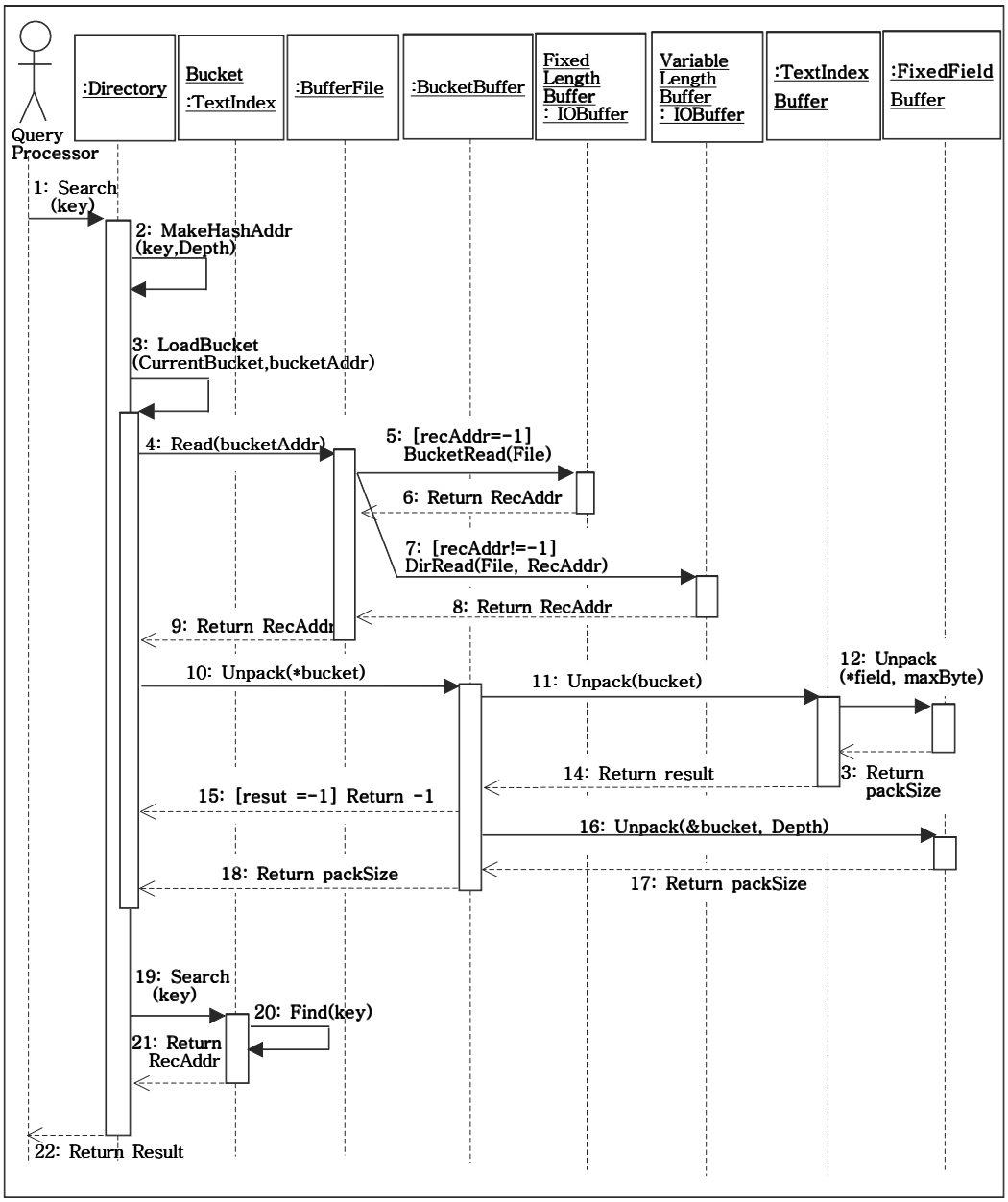


그림 15. EHP-indexing 기법을 이용한 검색작업을 위한 UML 순차 다이어그램

Fig. 15. UML sequence diagram for searching process by using EHP-indexing method

그림 16은 인덱스 키의 삭제 요청에 따라 수행되는 버킷통합 과정에 대한 UML 순차다이어그램을 표현한 것이다.

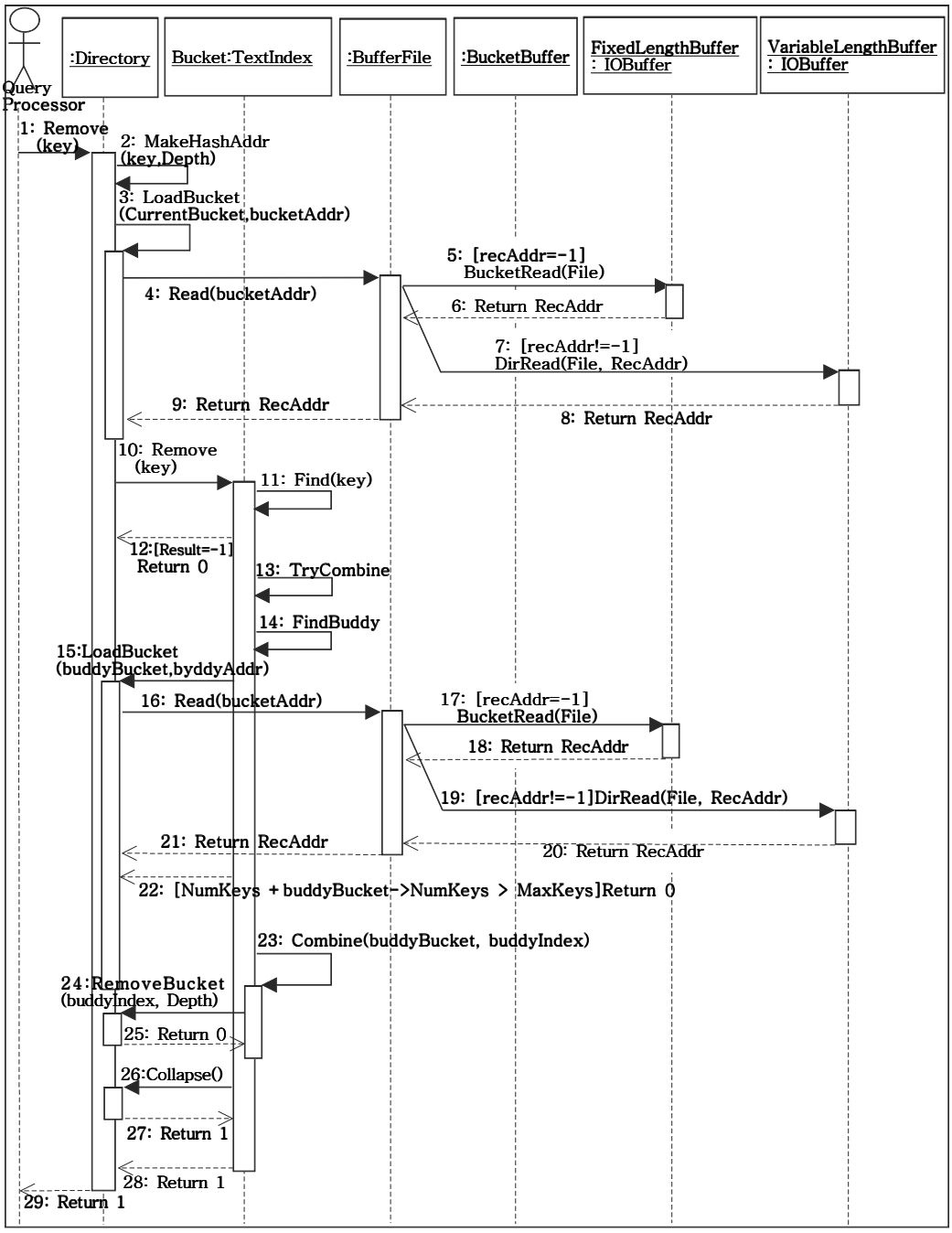


그림 16. 인덱스 키의 삭제 및 버킷통합 처리를 위한 UML 순차 다이어그램
Fig. 16. UML sequence diagram for processing combination of buckets and deletion of index key

3.1.4 EHP-indexing 기법에 관한 실험 및 고찰

본 논문에서 제안한 EHP-indexing 기법에 대한 성능실험은 다음과 같은 컴퓨터 시스템 환경에서 실시하였다. 실험은 Windows 2000® 운영체제를 탑재한 Intel® Pentium® 4 시스템 상에서 이루어졌다. 시스템의 사양은 메모리 1GB, CPU속도 1.7GHz, HDD 80GB로 구성되어 있으며, 데이터베이스 시스템으로 Oracle® 9i를 사용하여 실험을 수행하였다.

실험을 위해 입력되는 경로-지향 질의어의 경로는 모두 절대 위치 경로(absolute location path)형태로 입력하였다. 실험을 위한 경로-지향 질의어의 유형은 단순경로들만 입력되는 형태와 술어를 함께 포함하는 형태 2가지로 실험하였다.

실험에 사용되는 모든 XML문서의 엘리먼트들에 대한 최대 레벨은 4로 제한하여 실시하였다. 실험에 사용한 데이터들은 표 5와 같다.

표 5. 실험에 사용한 데이터들
Table 5. Simulation data

단축경로 개수	XML문서의 최대 깊이	해시코드 길이(bit)	평균 엘리먼트 수/문서	XML문서 파일의 수	슬롯의 수/버킷	경로-지향 질의어의 유형
1,000	4	21	53	19	173	/CustomerList/hobbies /sports
10,000	4	21	210	48	173	
100,000	4	21	300	334	173	/CustomerList/hobbies[artist$contains$'Bach']
500,000	4	21	400	1,250	173	
1,000,000	4	21	500	2,000	173	

실험 결과 버킷 파일과 디렉토리 파일 저장에 소요되는 파일의 크기는 표 6과 같다.

표 6에서 보여준 것과 같이 디렉토리 파일의 크기가 그다지 크지 않기 때문에 전체 디렉토리가 메인 메모리상에 모두 저장될 수 있다.

결과적으로 키 검색을 위한 디렉토리 액세스는 단지 한번만이 요구되기 때문에 시간 복잡도는 *O(1)*이 된다. 왜냐하면 확장성 해싱에서는 오버플로 처리를 위한 별도의 공간을 요구하지 않기 때문이다.

표 6. 실험을 위해 요구되는 디렉토리와 버킷 파일의 크기
Table 6. Required size of bucket file and directory file for simulation

Key의 수	Directory File의 크기	Bucket File의 크기	최저 슬롯의 수(k) (키의 수/버킷)
1,000	6.02KB	19.7 KB	4
10,000	6.02KB	124 KB	11
100,000	6.02KB	1.19 MB	24
500,000	6.02KB	1.91 MB	90
1,000,000	6.02KB	2.7 MB	173

본 논문에서 제안한 기법은 키의 검색 연산을 수행하기 위해 최악의 경우에도 시간복잡도 *O(k)*를 요구하고 있다.

여기서 *k*은 하나의 버킷에 저장할 수 있는 최대 레코드의 수를 말한다. 즉, 버킷 당 정해진 슬롯의 수를 말하는 것이다.

그림 17은 표 6의 실험 데이터를 이용하여 확장성 해싱 인덱싱 구조를 구성하는데 소요된 시간을 측정한 것이다.

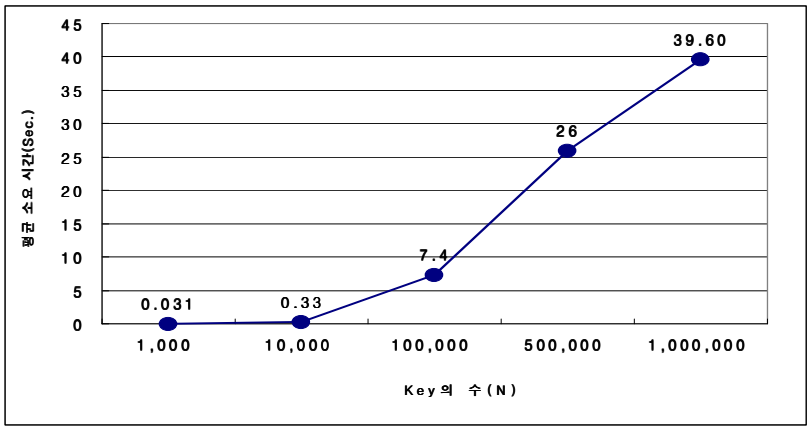


그림 17. 확장성 해싱 인덱싱 구조 생성을 위해 소요된 평균 시간

Fig. 17. Requirement of average time for building Extendible Hashing indexing structure

그림 18은 기존의 인덱싱 기법과 본 논문에서 제안한 인덱싱 기법과 관련하여 정보 검색에 요구되는 키의 최대 비교횟수를 분석한 그래프이다.

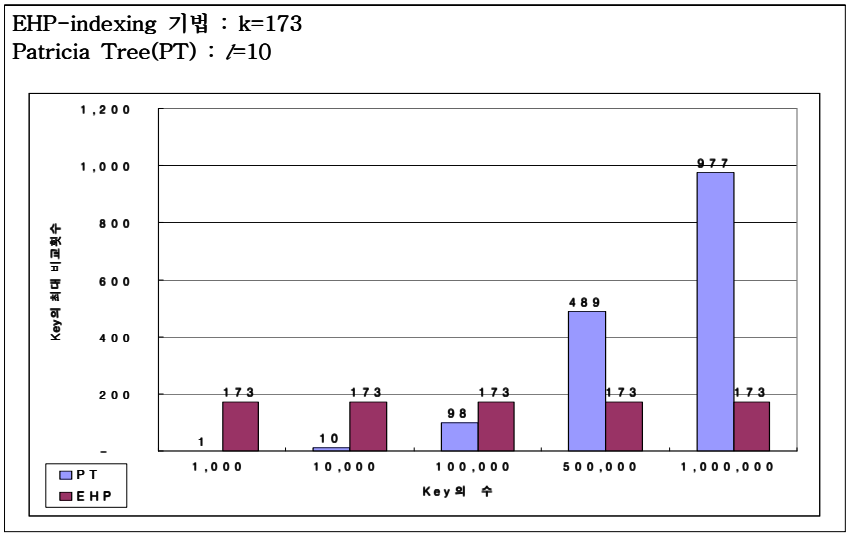


그림 18. 기존의 기법과 EHP-indexing 기법간의 키의 비교횟수 분석 결과

Fig. 18. Analysis result of the number of key comparisons between existing method and EHP-indexing method

실험결과를 비교하기 위해 기존 [18]의 인덱싱 기법을 위한 실험에도 동일한 조건의 데이터들을 사용하는 것으로 가정한다.

제안된 기법은 위의 식 (1.1)과 식 (1.2)에 의해 경로서명의 크기 F를 21로 할 때 D는 XML문서의 최대 깊이인 4가 되며 m은 최대 3이 된다고 볼 수 있으나 기존의 전통적인 경로서명 생성 방법과는 다른 단축경로를 경로서명 대신에 사용하고 있기 때문에 크게 의미는 없다.

기존의 기법도 경로서명의 크기 F를 21로 가정하였기 때문에 허위드롭(False drop)의 가능성을 최소화하고 최대의 성능을 유지하기 위해 l의 값은 10으로 결정하여 비교하였다. 그림 18에서 보여주는 것과 같이 기존의 기법은 각 외부노드에 한 개의 인덱스 키에 대한 정보만을 저장하는 방식을 사용하고 있기 때문에 인덱스 키의 수가 증가함에 따라 검색을 위해 요구되는 키의 비교 횟수가 비례적으로 증가하고 있다. 그러나 본 논문에서 제안한 인덱싱 기법은 외부노드로 다수의 인덱스 키에 대한 정보를 저장할 수 있는 버킷을 사용하고 있다. 따라서 입력되는 인덱스 키의 수에 관계없이 최악의 경우에도 항상 버킷의 슬롯 수인 k만큼만 키의 비교 연산을 수행하기 때문에 인덱스 키의 양에 관계없이 항상 일정한 검색 성능을 유지하고 있음을 알 수 있다. 결과적으로 제안된 기법의 시간 복잡도는 $O(k)$이다.

3.2 PMP-indexing 기법

본 절에서는 PMP-indexing 기법에 관하여 기술한다.

3.2.1 PMP-indexing 기법을 위한 시스템의 설계

본 논문에서 제안하고 있는 두 번째 인덱싱 기법은 경로-지향 질의어 처리 속도를 개선하기 위해 경로서명과 병렬 매치 인덱싱 구조를 결합하는 기법을 사용한다.

그림 19는 PMP-indexing 기법을 수행하기 위해 요구되는 논리적 시스템의 구성도를 보여준 것이다. 시스템의 구조는 3단 구조(3-tier)로 구성되며, 사용자 단, 인덱싱 시스템 단, 데이터베이스 단으로 구성된다. 여기서 데이터베이스의 위치는 인덱싱 시스템과는 동일한 시스템상 또는 물리적으로 떨어진 원격지 시스템상에 구현할 수도 있다.

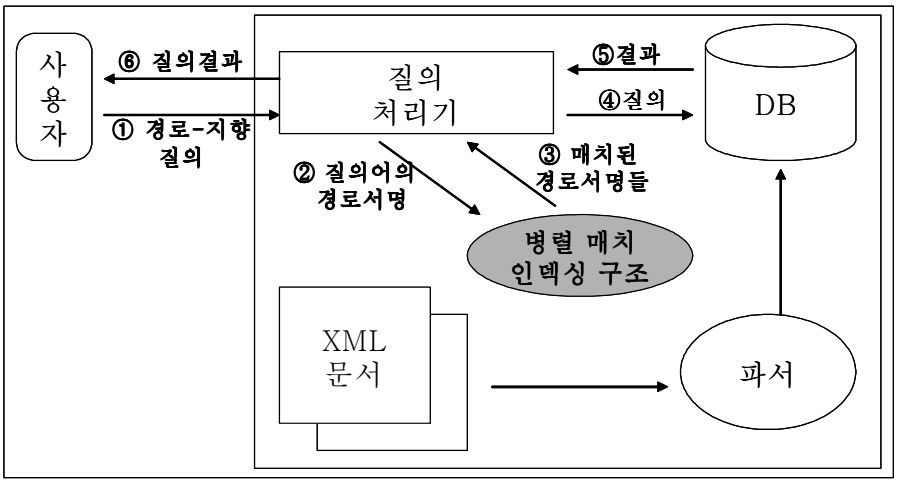


그림 19. PMP-indexing 기법을 위한 논리적 시스템 구성도
Fig. 19. Logical architecture of the system for PMP-indexing method

그림 19의 논리적 시스템 구성도에서 질의처리기(Query processor)는 사용자가 입력한 경로-지향 질의어를 실행 가능한 관계형 질의어 형태로 변환하는 기능, 경로-지향 질의어의 경로서명을 생성하는 기능 그리고 질의어 처리결과를 사용자에게 전달하기 위해 적절한 출력양식을 만들어 주는 기능들을 수행한다.

병렬 매치 인덱싱 구조는 사용자가 입력 질의어의 경로서명과 매치하는 경로서명을 경로서명 파일로부터 검색하는 작업을 수행한다.

파서는 병렬 매치 작업을 수행하기에 앞서 모든 XML문서에 대한 파싱을 수행하여 각 노드들을 관계형 데이터베이스 내에 엘리먼트 테이블, 텍스트 테이블, 애트리뷰트 테이블로 각각 분리 저장하는 작업을 수행한다.

다음 그림 20은 유효한 XML문서의 한 가지 예를 보여준 것이다.

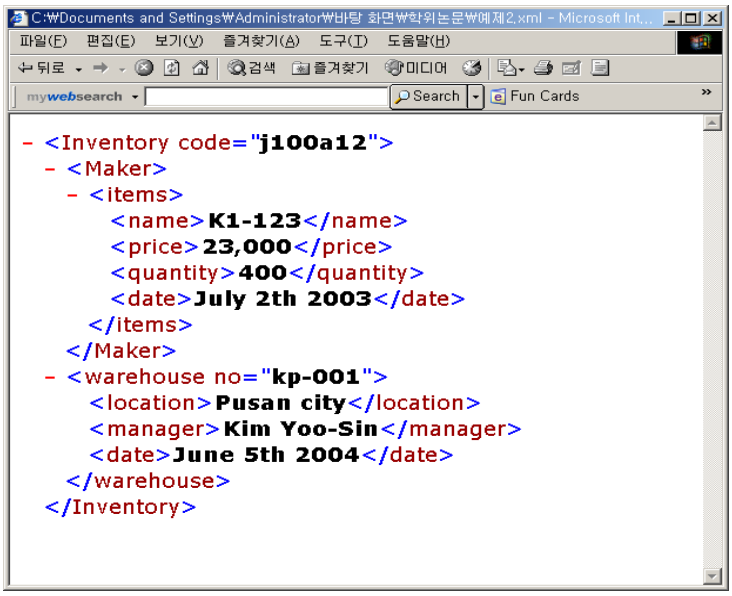


그림 20. 유효한 XML문서 예제

Fig. 20. An example of valid XML document

다음의 표 7, 표 8, 표 9는 그림 20의 예제 XML문서에 대한 파싱 결과를 각 테이블에 저장한 내용이다.

표 7. 엘리먼트 테이블

Table 7. Element table

문서ID	엘리먼트ID	엘리먼트명	부모ID	경로서명
1	1	Inventory	0	0011001
1	2	Maker	1	1010110
1	3	items	2	0001101
1	4	name	3	1100101
1	5	price	3	0000101
1	6	quantity	3	1100001
1	7	date	3	0011101
1	8	warehouse	1	0100101
1	9	location	8	0000110
1	10	manager	8	0010011
1	11	date	8	0010110

표 8. 텍스트 테이블

Table 8. Text table

문서	부모ID	텍스트내용
1	4	K1-123
1	5	2,3000
1	6	400
1	7	July 25th 2003
1	9	Pusan city
1	10	Kim Yoo-Sin
1	11	June 5th 2004

표 9. 애트리뷰트 테이블

Table 9. Attribute table

문서ID	부모ID	애트리뷰트명	속성
1	1	code	j100a12
1	8	no	kp-001

엘리먼트 테이블의 구조는 {문서ID:<Integer>, 엘리먼트ID:<Integer>, 엘리먼트명: <string>, 부모ID:<integer>, 경로서명:<string>}으로 구성된다. 여기서 문서ID 필드는 문서의 식별자를, 엘리먼트ID 필드는 엘리먼트의 식별자를 엘리먼트명 필드는 각 엘리먼트의 이름을, 부모ID 필드는 각 엘리먼트의 부모 엘리먼트에 대한 식별자를, 경로서명 필드는 각 엘리먼트의 경로서명값을 나타낸다.

텍스트 테이블의 구조는 {문서ID : <integer>, 부모ID : <integer>, 텍스트내용 : <string>}으로 구성된다. 여기서 부모ID 필드는 엘리먼트 테이블에 저장된 엘리먼트들의 식별자를 위한 것이며, 이것은 원래의 XML문서 내에서 대응하는 텍스트값을 가진다. 또한, 텍스트는 항상 부모 노드로서 엘리먼트를 가진다는 것에 주목해야만 한다. 텍스트내용 필드는 텍스트 데이터값을 의미한다.

애트리뷰트 테이블의 구조는 {문서ID:<integer>, 부모ID:<integer>, 애트리뷰트명 : <string>, 속성:<string>}으로 구성된다. 여기서의 부모ID 필드도 텍스트 테이블에서와 같이 엘리먼트 테이블에 저장된 엘리먼트들의 식별자들로서 대응하는 애트리뷰트값을 가진다. 속성 필드는 애트리뷰트값을 의미한다.

3.2.2 PMP-indexing 기법을 위한 구조 생성

본 논문에서 두 번째로 제안한 인덱싱 기법은 경로-지향 질의어의 평가 성능을 개선하기 위해 병렬 매치 인덱싱 구조를 사용한다.

병렬 매치 인덱싱 구조는 기존의 이진 트라이 구조와 유사한 구조를 가진다.

이진 트라이는 이진 비트 패턴들을 이용하여 트라이를 구성하며 중간노드인 분기노드와 외부노드인 정보노드로 구성되어 있다. 모든 정보는 외부노드에만 저장되기 때문에 분기노드는 정보를 가지지 않고 링크만을 가진다. 이진 트라이에서 최대 레벨은 경로서명의 길이(h)와 같다.

병렬 매치 인덱싱 구조는 XML문서의 모든 엘리먼트들에 대한 경로서명들을 이진 트라이로 구성하는 설계로부터 시작한다.

예를 들어, 그림 20의 XML문서내의 모든 엘리먼트들에 대한 경로서명 파일의 내용이 다음의 표 10과 같은 것으로 가정하자.

표 10. 예제 XML문서에 대한 경로서명 파일의 내용
Table 10. Contents of path signature file for an example XML document

위치	Element들의 경로	경로서명(h=7)
1	Inventory	0011001
2	Inventory/Maker	1010110
3	Inventory/Maker/items	0001101
4	Inventory/Maker/items/name	1100101
5	Inventory/Maker/items/price	0000101
6	Inventory/Maker/items/quantity	1100001
7	Inventory/Maker/items/date	0011101
8	Inventory/warehouse	0100101
9	Inventory/warehouse/location	0000110
10	Inventory/warehouse/manager	0010011
11	Inventory/warehouse/date	0010110

표10에 대한 경로서명 파일을 이진 트라이 구조로 구성하면 그림 21과 같

다.

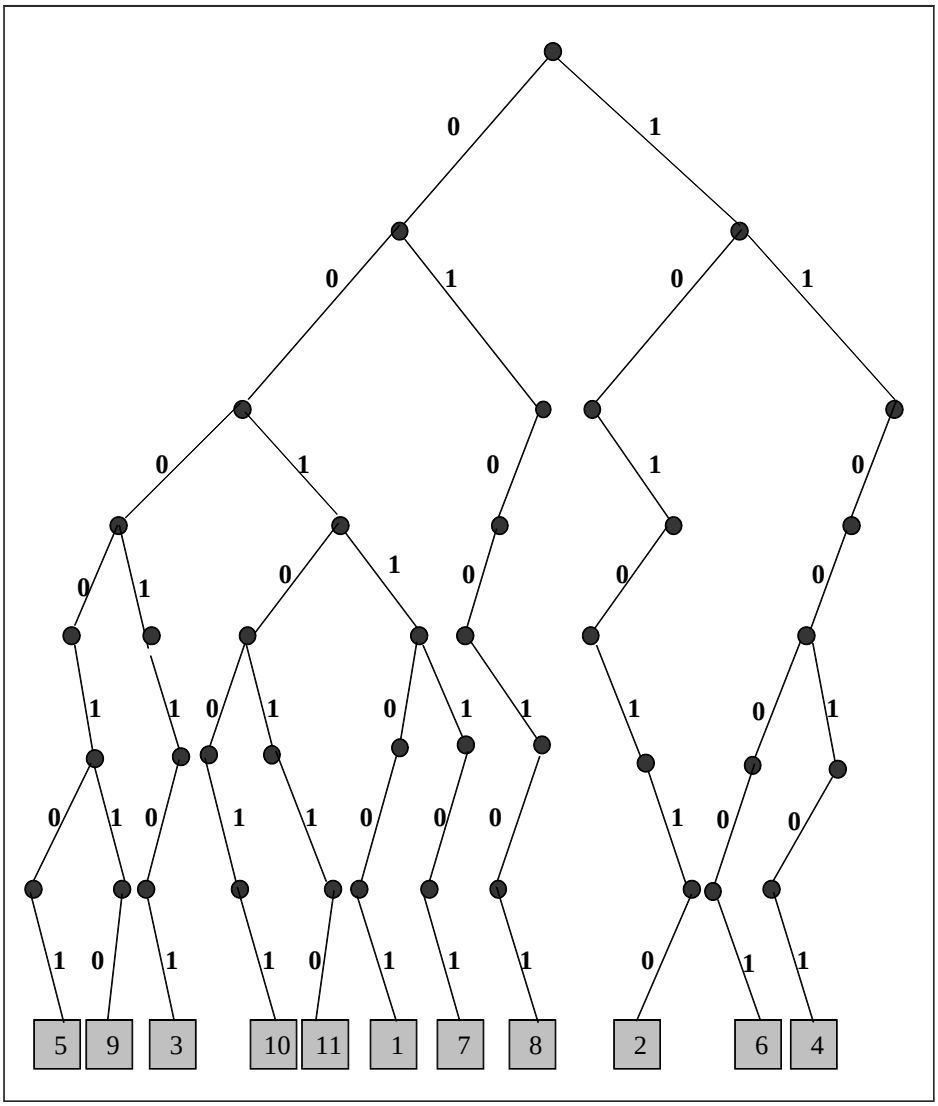


그림 21. 경로서명 파일에 대한 이진 트라이 구조
Fig. 21. Binary trie structure for path signature file

PMP-indexing 기법에서 이진 트라이 구조는 병렬 매치 처리를 위한 병렬 처리 구조로 변환해야만 한다. 따라서 이진 트라이구조에서 중간노드인 분기

노드는 모두 프로세스 엘리먼트로 변환된다.

각 프로세스 엘리먼트의 구조는 약간의 기억장소(Memory)와 단순한 비교기(Comparator)를 가지고 있으며, 하나의 입력선과 2개의 출력선을 가지는 구조가 된다. 여기서 인덱싱 구조를 형성하는 모든 프로세스 엘리먼트들의 기본 동작원리 및 기능은 동일한 것이다. 또한, 모든 프로세스 엘리먼트들은 모두 동시에 병렬로 동작을 한다.

경로서명 파일 내에 저장된 모든 경로서명 정보들은 제안된 병렬 매치 인덱싱 구조에서 외부노드에 저장된다. 각 외부노드들은 경로서명 파일내의 각 경로서명들의 위치정보를 제공하기 위한 약간의 저장 공간(Storage)과 매치 신호 또는 노-매치 신호(Match/No-match signal)를 표시하기 위한 표시기(Indicator)로 구성된다. 이 외부노드를 정보노드라 부른다.

그림 22는 중간노드인 개별 프로세스 엘리먼트노드의 구조와 외부노드인 정보노드의 구조를 나타낸 것이다.

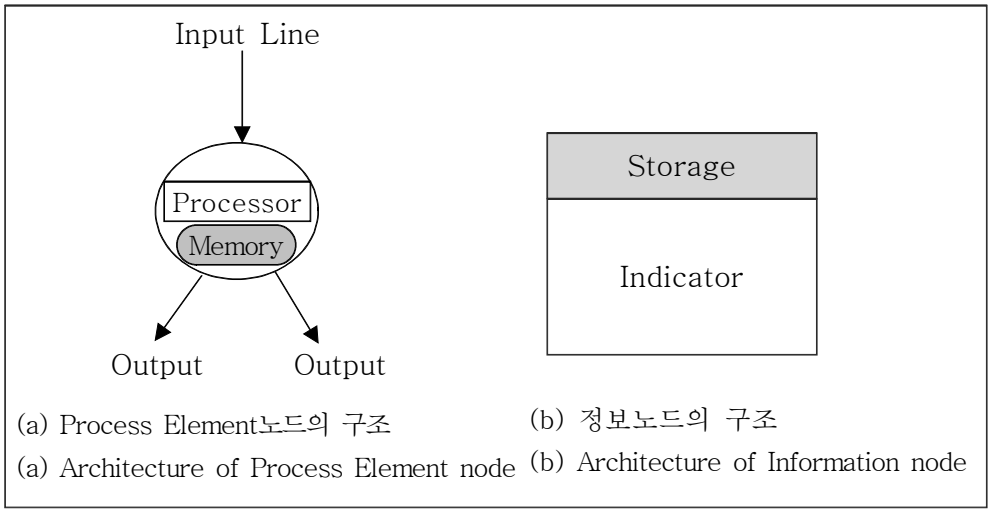


그림 22. Process Element노드와 정보노드의 구조

Fig. 22. Architecture of Process Element node and Information node

그림 23은 그림 21의 트라이 구조를 병렬 매치 인덱싱 구조로 변환한 상태를 보여주고 있다.

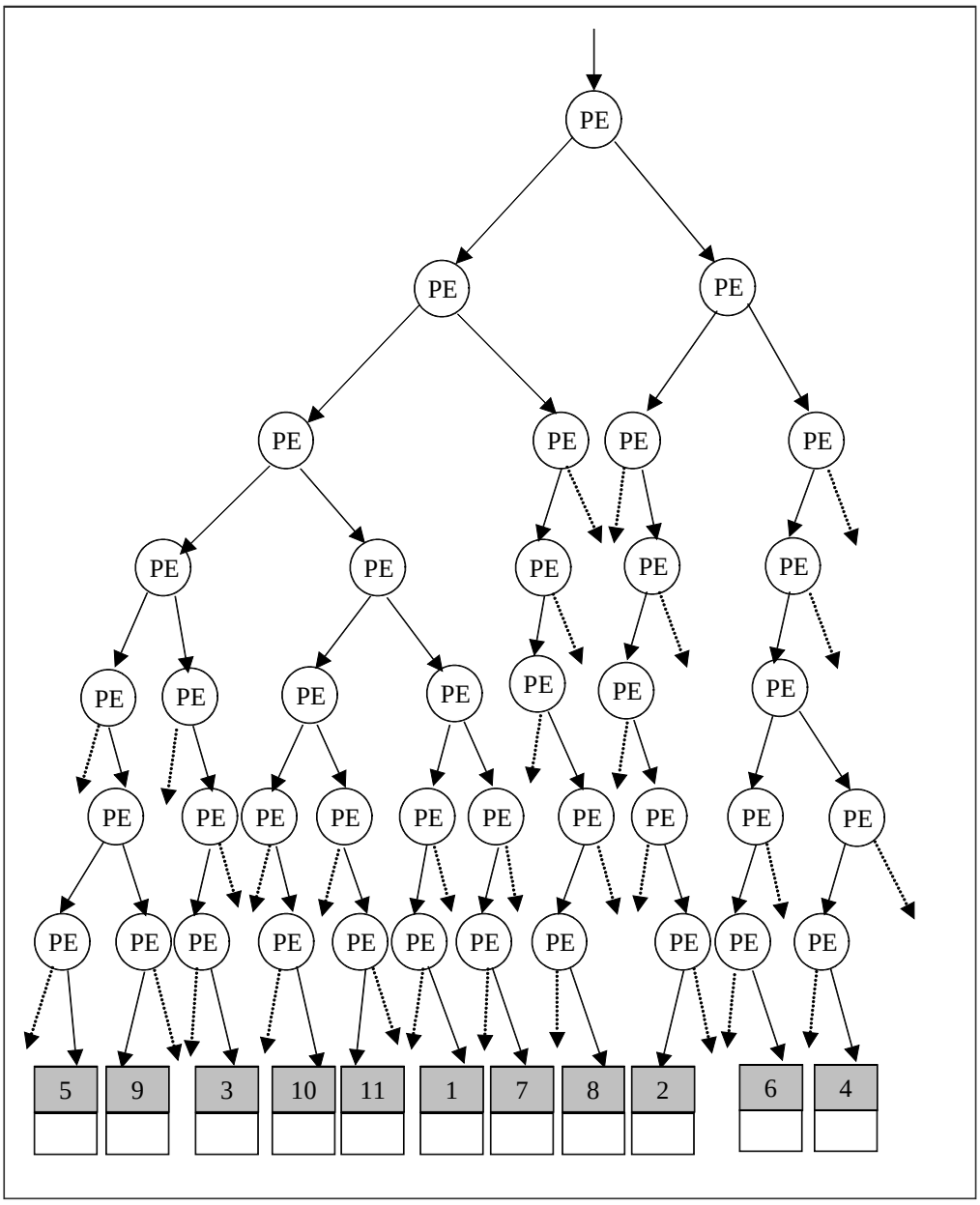


그림 23. 그림 21에 대한 병렬 매치 인덱싱 구조
Fig. 23. Parallel Match Indexing Fabric for Fig. 22

PMP-indexing 기법의 동작원리는 다음과 같은 몇 가지 처리과정을 단계적으로 수행함으로서 동작을 한다.

첫 번째 단계는 검색을 위해 입력된 경로-지향 질의어에 대한 경로서명값을 가장 오른쪽에 위치한 비트부터 가장 왼쪽에 위치한 비트에 이르기까지 역순으로 비트단위의 데이터를 순서대로 병렬 매치 인덱싱 구조의 루트 노드의 입력선을 통하여 전달을 한다. 인덱싱 구조에서 각 노드는 입력선을 통해 부모노드로부터 수신한 비트 데이터를 자신의 기억장소에 저장하고, 이전에 자신이 저장하고 있던 비트 데이터는 두 개의 출력선을 통하여 자신의 왼쪽과 오른쪽 자식 노드 모두로 전달한다. h시간이 경과된 후 매치하려는 경로서명의 가장 오른쪽 비트 데이터는 병렬 매치 인덱싱 구조에서 레벨 h상에 위치한 노드들에 도착을 하게 되고 가장 왼쪽 비트는 루트 노드에 도착을 한다.

두 번째 단계는 전 단계에 이어서 매칭 처리를 수행하기 위해 루트 노드에 매치 신호를 입력하는 단계이다. 여기서 매칭 처리를 시작하기 위한 신호값으로 루트 노드의 입력선에 논리 데이터 'true'를 매치신호로 입력을 한다. 각 노드에서 자식노드로 전달하려는 신호값은 부모노드로부터 입력된 매치 신호와 자신이 현재 저장하고 있는 비트 데이터값 모두를 이용하여 결정한다. 즉, 특정 노드에 저장된 값이 0이라면 입력선으로부터 들어온 신호가 그대로 양쪽 자식 노드 모두로 전달된다.

예를 들어, 만약 매치 신호인 논리 데이터 'true'가 매치 신호로서 입력선으로 들어오고 현재 노드가 저장하고 있는 값이 '0'이라면 따라서 매치 신호 'true'가 두 개의 출력선을 통해 왼쪽과 오른쪽 자식 노드 모두로 전달된다. 그러나 만약 노-매치 신호인 논리 데이터 'false'가 어떤 특정 노드의 입력선으로 들어온다면 해당 노드 자신이 저장하고 있는 값에 상관없이 노-매치 신

호 ‘false’가 두 개의 출력선을 통해 왼쪽과 오른쪽 자식 노드 모두에 전달된다.

만약 각 노드에 저장된 값이 ‘1’인 경우라면 매치 신호 ‘true’가 입력선으로 들어온다면 왼쪽 자식노드와 연결된 출력선으로 노-매치 신호 ‘false’를 전달하는 반면에 오른쪽 자식노드와 연결된 출력선으로 매치 신호 ‘true’를 전달한다. 만약 입력선으로 노-매치 신호 ‘false’가 특정 노드에 입력된 경우라고 하면 해당 노드는 저장된 값에 관계없이 무조건 왼쪽과 오른쪽 자식노드 모두로 노-매치 신호 ‘false’를 그대로 전달한다.

따라서, $2\times h$시간이 경과된 후, 매치를 위해 입력된 질의어에 대한 매치 처리는 완전히 끝나게 된다. 따라서 모든 정보 노드들의 표시기는 입력된 질의어의 경로서명이 경로서명 파일내의 경로서명들과 매치하는지, 노-매치하는지를 나타내기 위한 하나의 상태값을 모두 결정하게 된다.

마지막 단계는 인덱싱 구조의 외부노드들 중에서 매치된 데이터 결과들의 집합을 수집하는 단계이다. 여기서 만약 어떤 정보노드의 표시기가 논리값 ‘true’를 가지고 있다면 그것의 위치정보는 병렬 매치 인덱싱의 결과 집합으로 선택된다.

특히, 수집된 결과들은 잠재적으로 허위드롭(False drop)을 포함할 가능성이 있기 때문에 정확히 매치된 결과 집합들만을 얻기 위해서는 이들 허위드롭(False drop)을 제거하기 위한 추가적인 필터링 처리를 위한 단계가 필요하다.

예를 들어, 사용자에 의해 입력된 경로-지향 질의어에 대한 경로서명값이 ‘1000100’의 패턴을 가진다고 가정하자.

그림 24는 매치 작업을 수행하기 전의 병렬 매치 인덱싱 구조의 초기 상태를 표현한 것이다. 그림 24에서 볼 수 있듯이 모든 PE 노드들의 저장 공간은 '0'으로 외부노드는 모두 'false'로 초기화된 상태이다.

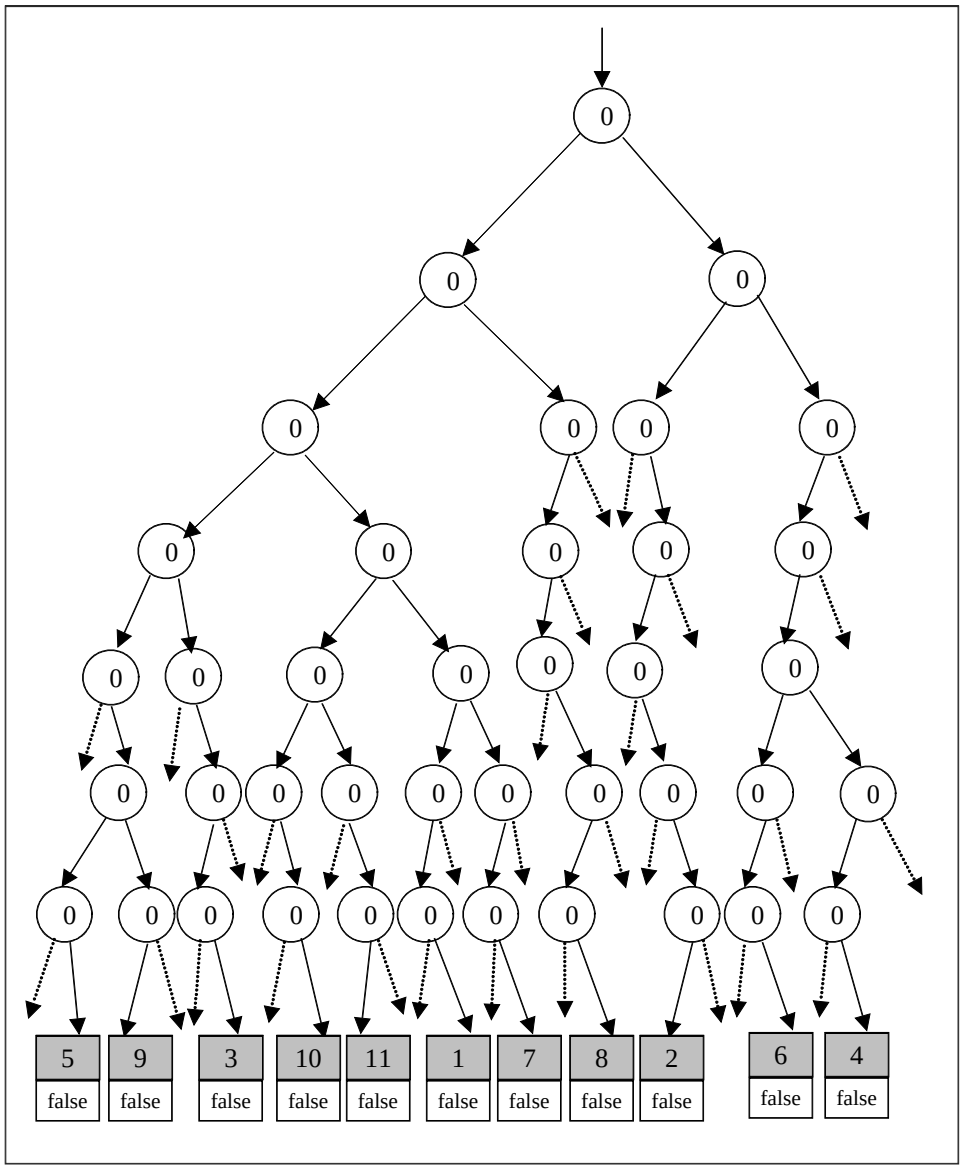


그림 24. 병렬 매치 인덱싱 구조의 초기상태
Fig. 24. Initial state of Parallel Match Indexing Fabric

그림 25는 병렬 매치 인덱싱 구조로 매치하려는 비트 패턴값들이 모두 전달된 *h*시간 이후의 병렬 매치 인덱싱 구조의 상태를 보여주고 있다.

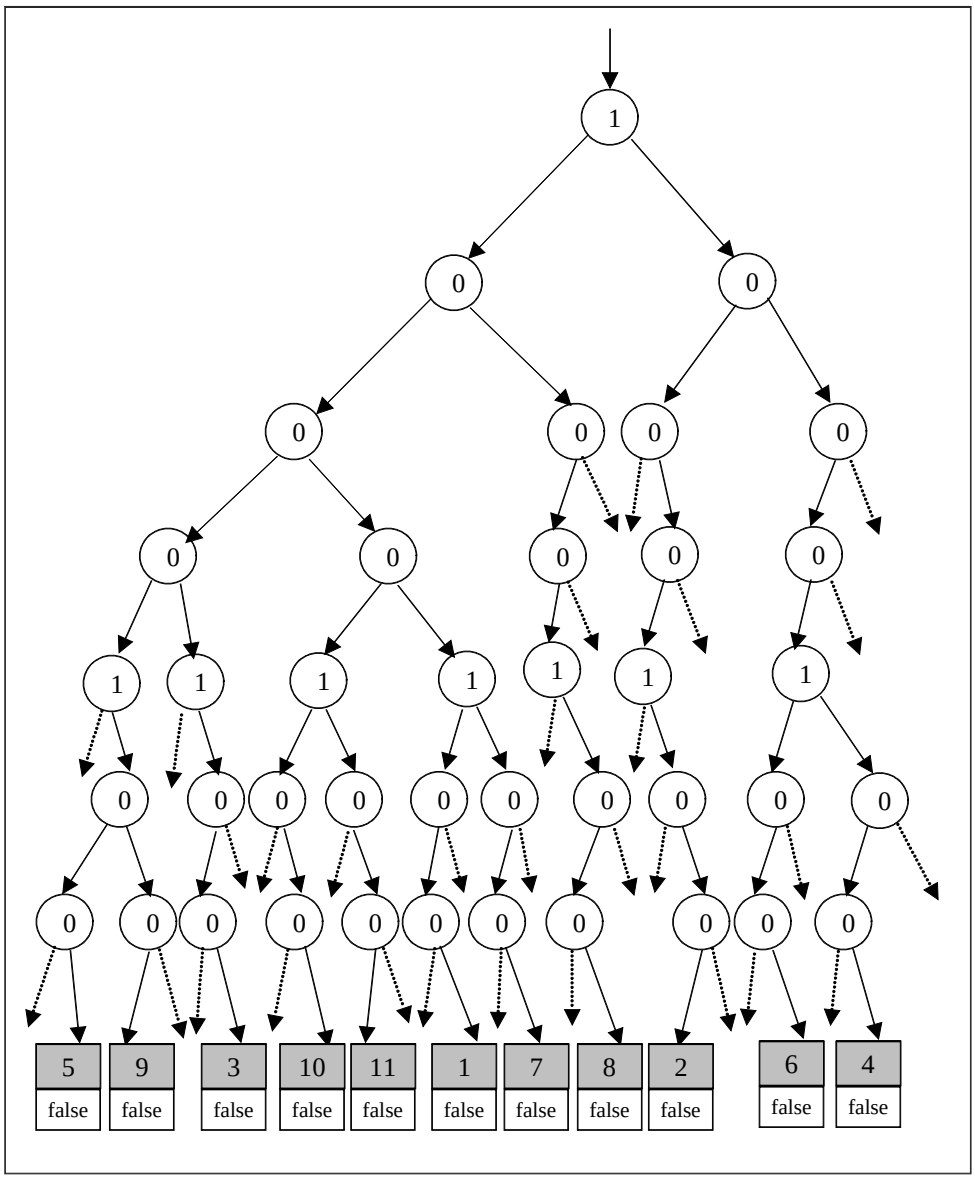


그림 25. *h*시간 이후, 병렬 매치 인덱싱 구조의 상태

Fig. 25. After *h* time, a state of Parallel Match Indexing Fabric

이 때 매치하려는 비트 패턴의 가장 왼쪽 비트는 루트 PE노드에 가장 오른쪽 비트는 레벨 *h*에 위치한 모든 PE 노드들에 도착한 상태이다.

그림 26은 매치 처리 시작 신호(true)를 입력한 후 2×*h*시간 이후의 병렬 매치 인덱싱 구조의 상태를 보여주고 있다.

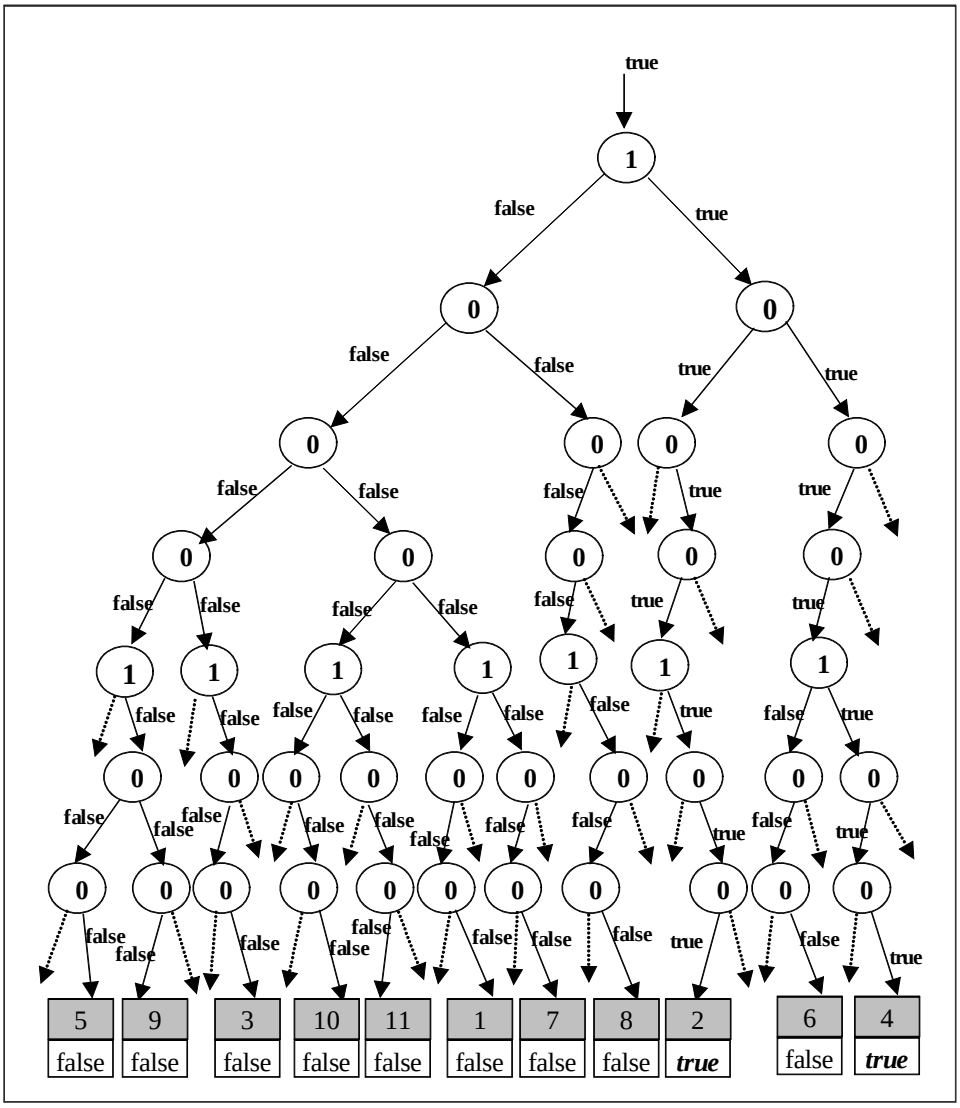


그림 26. *2×h*시간 이후, 병렬 매치 인덱싱 구조의 상태

Fig. 26. After *2×h* time, a state of Parallel Match Indexing Fabric

그림 26에서 보는 것과 같이 병렬 매치 인덱싱 구조와 병렬 매치 알고리즘을 사용하여 인덱싱을 수행한 결과 경로서명 파일에서 경로서명값 ‘1010110’과 ‘1100101’이 사용자가 입력한 경로-지향 질의어의 경로서명값 ‘1000100’과 매치한다. 따라서 이들 두 개의 정보 노드의 표시기는 매치된 데이터임을 나타내기 위해 논리값 ‘true’로 설정 된다. 그렇지만 다른 모든 정보 노드들의 표시기는 질의어의 경로서명과 매치가 되지 않는다는 것을 나타내기 위해 논리값 ‘false’로 설정된다. 따라서 매치 처리의 결과로서 우리는 경로서명값 ‘1010110’의 위치정보 2와 경로서명값 ‘1100101’의 위치정보 4를 얻는다.

예를 들어, 사용자가 경로-지향 질의어로 /Inventory/warehouse[location $contains&'Pusan city']를 입력한다고 가정하자.

병렬 매치 알고리즘 실행으로부터 얻어진 경로서명값의 정보는 관계형 데이터베이스에 대한 질의를 요청할 때 사용된다. 따라서 위의 경로-지향 질의어는 질의처리기에 의해 다음과 같은 관계형 질의어 형태로 변환되어야만 한다.

```
SELECT * FROM ElementTable a, TextTable b
        WHERE a.엘리먼트명='location'
        AND a.경로서명 matches 질의어의 경로서명
        AND a.문서ID=b.문서ID AND a.엘리먼트ID=b.부모ID
        AND b.텍스트내용ㄷ'Pusan city';
```

여기서 “matches”는 함수이며 그것은 매치된 경로서명값의 집합을 의미한다. 말하자면, 허위드롭(False drop)이 포함된 결과값들이다. 더욱 정확한 결과값을 얻기 위해서는 이들 허위드롭(False drop)을 제거하기 위한 추가적인 필터링 과정이 필요하다.

예를 들어, /Inventory/Maker/items란 경로-지향 질의어가 입력된다면 질의처리기는 다음과 같은 관계형 질의어 형태로 변환된다.

```
SELECT * FROM ElementTable a
        WHERE a.엘리먼트명='items'
        AND a.경로서명 matches 질의어의 경로서명;
```

3.2.3 PMP-indexing 기법을 위한 알고리즘

PMP-indexing 기법은 병렬 매치 알고리즘에 의해 구현된다. 따라서 본 절에서는 병렬 매치 알고리즘을 제안하고자 한다.

PMP-indexing 기법을 효율적으로 수행하기 위해서는 문서파싱처리 알고리즘, 질의처리기 알고리즘 그리고 병렬 매치 인덱싱 구조를 위한 알고리즘들이 필요하다.

문서파싱처리 알고리즘은 DOM파서에 의해 파싱된 특정 XML문서의 구성요소들을 각각 엘리먼트 테이블, 애트리뷰트 테이블, 텍스트 테이블에 분리 저장하는 기능과 함께 각 엘리먼트들의 경로서명을 생성하고 이를 엘리먼트 테이블과 경로서명 파일에 저장하는 역할을 수행한다. XML문서 파싱처리 알고리즘은 이전의 EHP-indexing 기법에서 사용하였던 알고리즘과 거의 유사하며, 단지 차이점은 단축경로 대신에 경로서명값을 생성하고 이를 경로서명 파일에 저장한다는 것이다.

그림 27은 XML문서파싱처리 알고리즘을 표현한 것이다.

```
/* level : level of Current Element node or level of Root Element node */
/* array IDTrace use to trace a path of each element */
/* array PathTrace use to get a path signature of each element node */
 XML_Document_Parsing(CurrentNode, level) {
 Input :  XML document, level
 Output : Path Signature File, Element table, Attribute table, Text table
 Current_NodeType=GetNodeType();
 switch(Current_NodeType) {
   case DOCUMENT_NODE :
        Document d=(Document) node;        /*  create XML document object */
        XML_Document_Parsing(RootElementNode, level+ 1);
        break;
   case ELEMENT_NODE:
         ElementID+ + ;
         IDTrace[level-1]=ElementID;
         PathTrace[level-1]=ElementName;
   for(int loop=1 to level) {          /* to get path signature of each element node */
        Create Path Signature of each element;
      }
   if(level==1)
        ParentID=0;
   else
        ParentID=IDTrace[level-2];
        Insert into ElementTable
          values(DocID, ElementID, ElementName, ParentID, PathSignature);
        Store Path Signature into Path Signature File;
        Get Attributes of CurrentElement;
        for(int loop=1 to Number of AttributeNode) {
            XML_Document_Parsing(AttributeNode, level+ 1);
              }
        Get ChildNoes of CurrentNode;
        if(ChildNode !=null){
          for(int loop=1 to Number of ChildNode) {
               XML_Document_Parsing(ChildNode, level+ 1);
                    }
          }
        break;
   case ATTRIBUTE_NODE:
         ParentID=ElementID;
         Insert into AttributeTable
              values(DocID, ParentID, AttributeNodeName(), AttributeNodeValue);
         break;
   case CDATA_SECTION_NODE :
   case TEXT_NODE:
         ParentID=ElementID;
         Get value of Current node;
         Insert into TextTable values(DocID, ParentID, TextNodeValue);
 }
  }
```

그림 27. XML문서 파싱과 경로서명 파일 생성 알고리즘

Fig. 27. Algorithm for generation of path signature file and Parsing of XML document

질의처리기 알고리즘은 다음과 같은 기능들을 수행한다.

i) 사용자가 입력한 경로-지향 질의어의 경로서명을 생성한 후 이것을 병렬 매치 인덱싱 구조로 전달한다.
ii) 병렬 매치 인덱싱 구조를 사용하여 얻어진 매치된 결과값을 이용하여 경로-지향 질의어를 관계형 질의어 형태로 변환한 다음 관계형 데이터베이스에 대한 질의를 수행한다.
iii) 관계형 질의어 수행 결과로 얻어진 결과값을 사용자가 알 수 있는 적당한 포맷으로 변환하여 사용자에게 최종 결과값으로 반환한다.

전체 시스템 구성에서 병렬 매치 인덱싱 구조는 가장 중요한 부분으로서 실제로 사용자에 의해 입력된 경로-지향 질의어의 경로서명과 경로서명 파일에 저장된 경로서명의 매치 처리를 수행한다. 이 알고리즘은 다음과 같은 기능들을 수행한다.

i) 파싱 과정에서 생성된 경로서명 파일을 이용하여 데이터베이스에 저장된 모든 엘리먼트들에 대한 병렬 매치 인덱싱 구조를 생성한다.
ii) 질의처리기로부터 전달 받은 입력된 경로-지향 질의어의 경로서명을 병렬 매치 인덱싱 구조로 전달한다.
iii) 병렬 매치 처리 시작신호(true)를 병렬 매치 인덱싱 구조로 전달한다.
iv) 경로서명값을 이용하여 검색 연산을 수행한 후 정보노드의 표시기가 매치 신호(true)로 설정된 노드들의 정보를 모두 수집한 후 그 결과값을 질의처리기로 반환한다.

본 논문에서는 다음과 같은 몇 가지 이유들 때문에 병렬 매치 알고리즘에 대한 내용을 기술하기에 앞서 순차 매치 알고리즘(Sequential Match Algorithm)의 설계에 대한 내용을 먼저 설명하고자 한다.

첫 번째 이유는 순차 매치 알고리즘은 인덱싱 구조를 생성함에 있어서 트라이 구조를 사용하고 있다. 따라서 순차 매치 인덱싱 구조(Sequential Match Indexing Structure) 생성 과정은 병렬 매치 인덱싱 구조 생성의 초기 단계와 동일하다.

두 번째 이유는 비록 순차 매치 인덱싱 구조와 병렬 매치 인덱싱 구조의 초기 단계 구조가 동일한 트라이 구조를 갖는다 할지라도 매치 처리를 수행하는 방법은 병렬 매치 알고리즘과는 다르게 수행된다는 것을 보여주기 위한 것이다.

세 번째 이유는 순차 매치 알고리즘은 기존 Chen [18]의 기법과 동일한 매치 처리 방법을 사용하고 있기 때문이다.

지금부터 순차적 매치 알고리즘에 대하여 기술한다. 순차 매치 처리에서 검색 연산을 위한 매치 테스트는 이진 트라이를 운행하는 동안 다음과 같은 순서로 수행된다.

i) 만약에 이진 트라이구조에서 검사하려는 경로서명값의 i-번째 위치의 비트값이 '0'을 가진다면 왼쪽과 오른쪽 자식노드를 모두 방문한다.
ii) 만약 해당 위치의 비트값이 '1'이라면 오른쪽 자식 노드만을 방문한다.
iii) 정보노드인 단말노드에 도착 하였을 때 'valid' 마크를 단말노드의 매치 표시기에 저장한다.

순차 매치 알고리즘 수행 결과로는 각 단말노드에서 'valid' 마크를 가진 경

로서명에 대한 데이터베이스 위치를 얻게 된다.

그림 28은 순차 매치 인덱싱 구조를 생성하기 위한 알고리즘이다.

```
Sequential_Match_Indexing_Structure_Construction(Key, pointer_of_Root_Node, h)
 Input: Input: Key, pointer_of_Root_Node, h      /* h : length of Key  */
 Output : Binary Match Trie(sequential match indexing structure)
{
Current_node=pointer_of_Root_Node; /* Pointer of root node is assigned to pointer of current
node */
for(i=1 to h-1) /* Building Sequential Match Indexing Fabric */
 {
 /* Making left child intermediate node and connecting to parent node */
  if(bits(Key, i,1)==0 and Current_node->LLink==NULL){
     Current_node->LLink=Intermedi ate_Node_Creation();
     Current_node=Current_node->LLink;
 }
  /* Making right child intermediate node and connecting to parent node */
  if(bits(Key, i,1)==1 and Current_node->RLink==NULL) {
     Current_node->RLink=Intermediate_Node_Creation();
     Current_node=Current_node->RLink;
  }
  /* Move to left child node */
  if(bits(Key, i,1)==0 and Current_node->LLink!=NULL) {
     Current_node=Current_node->LLink;
  }
     /* Move to right child node */
  if(bits(Key, i,1)==1 and Current_node->RLink!=NULL) {
     Current_node=Current_node->RLink;
  }
}
 /* For making external node */
 if(bits(Key, h, 1)==0)
         Current_node->LLink=External_Node_Creation(Key);
 else
        Current_node->RLink=External_Node_Creation(Key);
}
```

그림 28. 순차 매치 인덱싱 구조 구성을 위한 알고리즘

Fig. 28. Algorithm for building of Sequential match indexing structure

그림 29는 생성된 순차 매치 인덱싱 구조를 이용하여 입력된 경로-지향 질의어의 경로서명값에 대한 순차 매치 처리를 수행하는 알고리즘이다.

```
Sequential_Match_Processing(Key)
Input : Key, pointer_of_Root_Node, h
Output : matched result
{
   Binary_Trie_Traversal(Key, 0, pointer_of_Root_Node)
}
  /* match or no-match signal is set to Indicator of leaf node */
Binary_Trie_Traversal(Key, level, base_Node)
{
if(base_node==NULL)
     return;
if(level==Key.length)
 {
   if(base_Node->LLink != NULL)
      base_Node->LLink->CheckField='valid';
   if(base_Node->RLink != NULL)
      base_Node->RLink->CheckField='valid';
   return;
   }
  /* Match or no-match signal is sent to left and right child PE nodes */
    if(bits(Key, level,1)=='0') {
       Binary_Trie_Traversal(Key, level+1, base_PE->LLink);
       Binary_Trie_Traversal(Key, level+1, base_PE->RLink);
    }
    else
    {
       Binary_Trie_Traversal(Key, level+1, base_PE->RLink);
    }
}
```

그림 29. 경로서명의 순차 매치 처리를 위한 알고리즘

Fig. 29. Algorithm for sequential match processing of path signature

위의 순차 매치 알고리즘의 설계에 따라서 다음과 같은 정리 1을 얻을 수

있다.

정리 1. 순차 매치 알고리즘의 시간 복잡도는 $O(N/2^k)$이다. 여기서 N은 경로서명의 개수를 말하며, k는 매치하려는 경로서명값 내에 '1'로 설정된 비트들의 수이다.

증명) 전체 경로서명을 위한 이진 트라이 내의 단말노드의 수는 N이다. $1 < k < h$, 여기서 h는 경로서명의 길이다. N의 상한 범위는 2^h라는 것이 자명하다. 명백하게, 매치를 위해 주어진 각 경로서명내의 1로 설정된 비트는 실제로 방문하려는 서브트리의 절반의 방문을 금지한다. 따라서 이 기법은 최악의 경우에도 $O(N/2^k)$의 비교 횟수만을 요구한다.□

그러므로, 위의 순차 매치 알고리즘의 시간 복잡도는 Chen [18]의 기법에서의 시간 복잡도와 거의 유사하다는 것을 알 수 있다.

지금부터는 병렬 매치 알고리즘에 대하여 설명한다.

본 논문에서 제안하고 있는 병렬 매치 알고리즘은 두 가지 주요 알고리즘과 몇 개의 부속 알고리즘들로 구성되어 있다.

주요 알고리즘들 중에서 그 첫 번째는 병렬 매치 인덱싱 구조를 구성하기 위한 알고리즘이다. 이 알고리즘은 프로세스 엘리먼트 노드와 정보노드를 생성하기 위한 2가지 부속 알고리즘들을 포함한다.

두 번째 알고리즘은 매칭 처리 실행을 위한 알고리즘이다. 이것은 매치된 데이터 수집 작업을 수행하기 위한 알고리즘, 매치 표시기들의 리셋을 처리하

기 위한 알고리즘, 질의어의 경로서명값을 인덱싱 구조로 전달하기 위한 알고리즘 그리고 매치 처리 시작 신호를 인덱싱 구조로 전달하기 위한 알고리즘들을 포함하고 있다.

그림 30은 병렬 매치 알고리즘의 기능 패키지 다이어그램을 보여주고 있다.

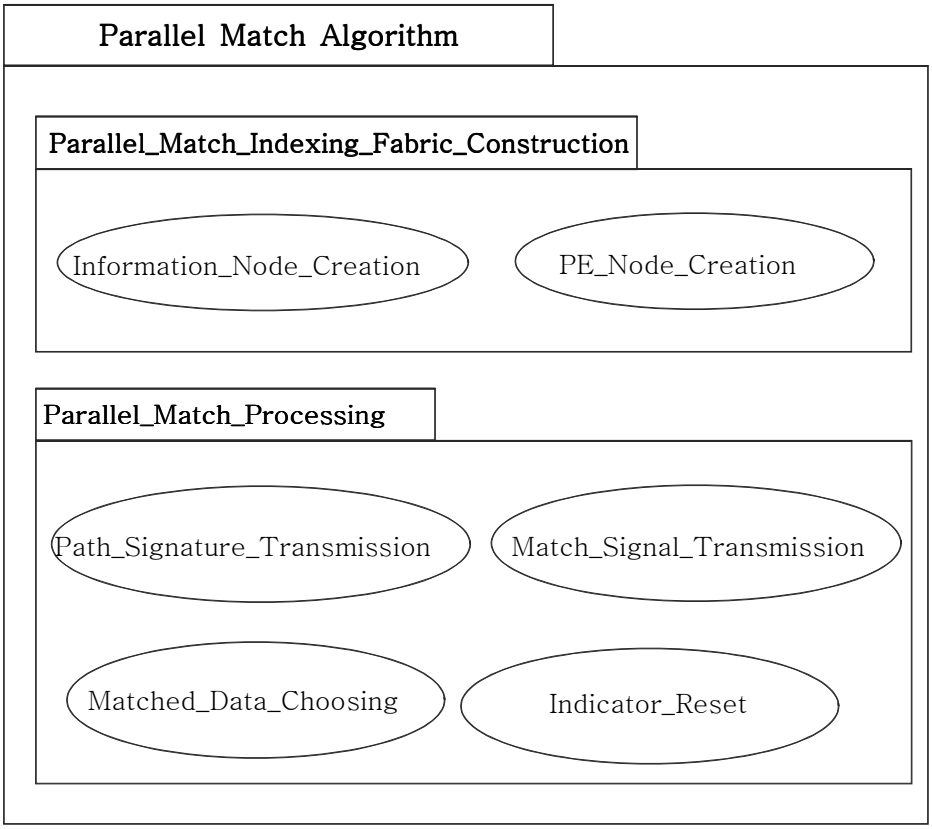


그림 30. 병렬 매치 알고리즘의 기능 패키지 다이어그램
Fig. 30. Function package diagram of parallel match algorithm

본 논문에서 제안하고 있는 병렬 매치 인덱싱 구조를 형성하기 위한 알고리즘은 그림 31과 같다. 여기서 우리는 부속 알고리즘으로 중간노드(PE 노드)와 외부노드(정보노드)를 생성하기 위한 알고리즘을 포함한다. 이 알고리즘의 실

행이 완료된 이후 우리는 경로서명 파일을 가지고 구성한 이진 트라이와 동등한 구조를 가지는 완전한 병렬 매치 인덱싱 구조를 얻게 된다.

```
Parallel_Match_Indexing_Fabric_Construction(Key, Root_PE_Pointer, h)
 Input : Key, Root_PE_Pointer, h
 Output : Parallel Match indexing Fabric
{
  Current_node=Root_PE_Pointer; /* Pointer of root PE node is assigned to pointer of current
node */
  for(i=1 to h-1) {      /* Building Parallel Match Indexing Fabric */
 /* Making left child intermediate(PE) node and connecting to parent node */
    if(bits(Key, i,1)==0 and Current_node->LLink==NULL){
        Current_node->LLink=PE_Node_Creation();
        Current_node=Current_node->LLink;
 }
  /* Making right child intermediate(PE) node and connecting to parent node */
    if(bits(Key, i,1)==1 and Current_node->RLink==NULL) {
        Current_node->RLink=PE_Node_Creation();
        Current_node=Current_node->RLink;
  }
    if(bits(Key, i,1)==0 and Current_node->LLink!=NULL) {   /* Move to left child node */
        Current_node=Current_node->LLink;
  }

    if(bits(Key, i,1)==1 and Current_node->RLink!=NULL) { /* Move to right child node */
        Current_node=Current_node->RLink;
  }
}
 /* Making external(information) node */
  if(bits(Key, h, 1)==0)
        Current_node->LLink=Information_Node_Creation(Key);
  else
        Current_node->RLink=Information_Node_Creation(Key);
}
```

그림 31. 병렬 매치 인덱싱 구조 구성을 위한 알고리즘

Fig. 31. Algorithm for constructing Parallel Match Indexing Fabric

그림 32는 본 논문에서 제안한 병렬 매치 인덱싱 구조를 이용하여 경로서명 파일에 저장된 경로서명값들과 사용자가 입력한 질의어의 경로서명값을 병렬 매치 처리하기 위한 알고리즘을 기술한 것이다. 이 알고리즘의 수행이 종료된 이후 우리는 매치된 결과값들을 얻을 수 있다.

```
Parallel_Match_Processing(Key, h, root_PE)
 Input : Key, h, root_PE
 Output : Position of a matched Key
{
 Information_Node->Indicator=Indicator_Reset(); /* Reset indicator of information node */
 for(beat=1 to h)    /* Each bit value of a signature is transferred to PMIF */
  {
    one_bit_ps=bits(Key, h-beat,1);
    Path_Signature_Transmission (one_bit_ps, beat, root_PE);
  }
    Match_Signal_Transmission(true, h, root_PE); /* Match signal is sent to PMIF  */
    Matched_results[]=Matched_Data_Choosing();  /* Pick up matched data */
 }
```

그림 32. 키의 병렬 매칭 처리를 위한 매치 알고리즘

Fig. 32. Algorithm for processing parallel matching of a key

그림 33은 경로서명값을 비트단위로 역순으로 병렬 매치 인덱싱 구조로 전달하기 위한 알고리즘이다.

```
/* input_bit : a bit value of a path signature of key in reverse order ,
    level : i-th bit position , base_PE : pointer of current node */
Path_Signature_Transmission(input_bit, level, base_PE)  /* First-step process of parallel matching */
  Input : Input_bit, level, base_PE
  Output : 0 or 1
 {
  Temp=base_PE->storedData;  /* A bit data stored in current node is passed to left and right child */
  base_PE->storedData=Input_bit; /* nodes and a new entered bit data is stored into current node */
  if(level==1)
      return;
  else
   {
     if(base_PE->LLink != NULL)
        Path_Signature_Transmission(Input_bit, level-1, base_PE->LLink);
     if(base_PE->RLink != NULL)
        Path_Signature_Transmission(Input_bit, level-1, base_PE->RLink);
   }
  }
```

그림 33. 병렬 매치 인덱싱 구조로 경로서명 비트값을 전달하는 알고리즘

Fig. 33. Algorithm for passing a bit value of a path signature to Parallel Match Indexing Fabric

그림 34는 매칭 처리를 수행하기 위해 매치(true) 또는 노-매치(false) 신호 중 한 가지를 병렬 인덱싱 구조로 전달하는 알고리즘이다.

```
/* signal : match(true) or no_match(false), level : each level, base_PE: pointer of current node */

Match_Signal_Transmission(signal, level, base_PE); /* Second-step process of parallel matching */
Input : signal, level, base_PE
Output : true or false
{
if(level==1)  /* Match or No-match signal is set to Indicator of leaf node */
 {

  if(base_PE->LLink != NULL)
    base_PE->LLink->matchFlag=signal;
  if(base_PE->RLink != NULL)
    base-PE->RLink->matchFlag=signal;
  return;
  }
 /* Match or No-match signal is sent to left and right child PE nodes */
 if(signal==true) {

   if(base_PE->storedData=='0') {
     Match_Signal_Transmission(true, level-1, base_PE->LLink);
     Match_Signal_Transmission(true, level-1, base_PE->RLink);
    }
   else
     {
     Match_Signal_Transmission(false, level-1, base_PE->LLink);
     Match_Signal_Transmission(true, level-1, base_PE->RLink);
     }
 }
  else {
     Match_Signal_Transmission(false, level-1, base_PE->LLink);
     Match_Signal_Transmission(false, level-1, base_PE->RLink);
  }
}
```

그림 34. 병렬 매치 인덱싱 구조로 매치 또는 노-매치 신호 전달 알고리즘

Fig. 34. Algorithm for transmission of a match signal(true)/no-match(false)

위의 병렬 매치 알고리즘의 설계에 따라서 다음과 같은 정리 2를 얻을 수 있다.

정리 2. 병렬 매치 인덱싱 구조를 사용하는 PMP-indexing 기법은 정확하다. 즉, 매치된 결과로서 이진 트라이에서 각 단말노드는 정확한 매치 신호와 함께 유효한 엘리먼트들의 경로서명값에 대한 데이터베이스 내의 위치를 출력한다.

증명) 이진 트라이는 경로서명 파일의 표현이다. 이진 트라이에서 루트 노드에서부터 단말노드까지 이르는 비트 스트링은 경로서명과 동일하다. 본 논문의 병렬 매치 알고리즘은 경로서명 매칭을 위해 두 가지 단계로 구성한다. 첫 번째 단계 수행 이후, 트라이에서 레벨 1에 위치한 루트 노드는 매치하려는 경로서명의 가장 왼쪽 비트값을 가진다. 그리고 레벨 i에 위치한 노드들은 i-번째 비트값을 가진다. 두 번째 단계에서는 매치 처리 시작 신호가 루트 노드로 입력된다. 그런 다음 루트노드가 '0'을 저장하고 있는지 아니면 '1'을 저장하고 있는지에 따라서 'true' 또는 'false'신호 중의 한 가지를 생성한다. 만약 '0'을 저장하고 있는 노드라면 왼쪽과 오른쪽 자식노드 모두로 'true'신호를 전달한다. 다른 한편으로, 만약 '1'을 저장하고 있는 노드라면 왼쪽 자식 노드로는 'false'신호를 오른쪽 자식 노드로는 'true'신호를 각각 전달한다. 각각의 'true'와 'false'신호가 단말노드에 도착한 이후 매치를 하기 위해 주어진 경로서명과 동등한 각 단말노드는 'true'신호를 유지하는 반면, 매치되지 않는 단말노드들은 'false'를 유지한다.□

제안된 병렬 매치 인덱싱 구조와 병렬 매치 알고리즘을 사용함에 따라 다음과 같은 정리 3을 얻을 수 있다.

정리 3. 병렬 매치 인덱싱 구조를 이용하는 PMP-indexing 기법의 시간 복잡도는 $O(h)$이다.

증명) h시간 이후, 매치시키고자 하는 경로서명의 가장 오른쪽 비트값은 레벨 h상에 위치한 모든 노드에 그리고 레벨 1에는 가장 왼쪽 비트값이 도착한다. 또한, $2\times h$ 시간 이후에, 'true' 또는 'false'신호 중의 한 가지가 모든 정보노드에 도착을 한다. 본 논문에서 제안한 PMP-indexing 기법은 병렬 매치 연산을 수행하기 위해 오직 $2\times h$의 시간만을 요구한다. 따라서 시간 복잡도는 $O(h)$이며, 경로서명 파일내의 엔트리의 수(N)에 대한 상한 영역은 2^{h-1}이다. 여기서 h는 경로서명의 길이인 동시에 병렬 매치 인덱싱 구조의 높이와 같다. □

3.2.4 PMP-indexing 기법에 관한 실험 및 고찰

본 논문에서 제안한 PMP-indexing 기법에 대한 성능실험은 다음과 같은 컴퓨터 시스템 환경에서 실시하였다. 실험은 Windows 2000® 운영체제를 탑재한 Intel® Pentium® 4 시스템 상에서 이루어졌다. 시스템의 사양은 메모리 1GB, CPU속도 1.7GHz, HDD 80GB로 구성되어 있으며, 데이터베이스 시스템으로 Oracle® 9i를 사용하여 실험을 수행하였다. 또한, 실험을 위한 시스템의

구현은 C++와 JAVA언어를 사용하여 소프트웨어적으로 구현하였다.

시스템의 성능 실험을 위해 입력되는 경로-지향 질의어의 경로는 절대 위치 경로(absolute location path)형태와 축약형 위치 경로(abbreviated location path)형태 모두를 입력하였으며, 경로-지향 질의어의 유형은 단순히 경로들만 입력하는 형태와 술어를 함께 포함하는 형태 2가지로 실험하였다.

실험에 사용되는 모든 XML문서의 엘리먼트들에 대한 최대 깊이는 4로 제한하여 수행하였다.

실험에 사용한 데이터들은 표 11과 같다.

표 11. 실험에 사용한 데이터들
Table 11. Simulation data

경로서명의 수	XML문서의 최대 깊이	해시코드 길이(bit)	평균 엘리먼트 수/문서	XML문서 파일의 수	경로-지향 질의어의 유형
1,000	4	21	53	19	/Inventory/Maker/items
10,000	4	21	210	48	//manager
100,000	4	21	300	334	/Inventory/warehouse [location$contains& 'Pusan city']
500,000	4	21	400	1,250	
1,000,000	4	21	500	2,000	

실험결과에 따르면, 병렬 매치 알고리즘을 사용하였던 본 논문에서 제안한 PMP-indexing 기법의 시간 복잡도는 $O(h)$임을 알 수 있다. 왜냐하면 h-클럭 이후 입력 스트링의 첫 번째 비트패턴 신호(가장 오른쪽 비트 패턴)가 단말노드에 도착하고 마지막 매치 신호는 h-클럭 이후 루트 노드에 입력되기 때문이다. 또한, $2\times h$ 클럭 이후에 매치 또는 노-매치 신호가 단말노드에 도착하기 때문이다.

결과적으로, PMP-indexing 기법은 매치를 위해 $2\times h$ 클럭을 요구한다. 따

라서 시간 복잡도는 $O(h)$이다. 병렬 매치 알고리즘은 경로서명 파일의 개수(N)에는 영향을 받지 않으며 단지 경로서명의 길이 h에 의해서만 영향을 받는다. 경로서명의 길이가 h일 때 이 알고리즘을 수행 하고자 할 때 최악의 경우에 있어서 총 PE의 수는 2^h-1개이다. 여기서 경로서명 파일의 엔트리(인덱스) 수는 $N \leq 2^{h-1}$이다.

그림 35는 패트리샤 트리를 이용하는 기존 Chen [18]의 알고리즘과 본 논문의 병렬 매치 인덱싱 구조를 사용하는 병렬 매치 알고리즘의 수행결과에 대한 키의 최대 비교 횟수를 비교분석한 것이다.

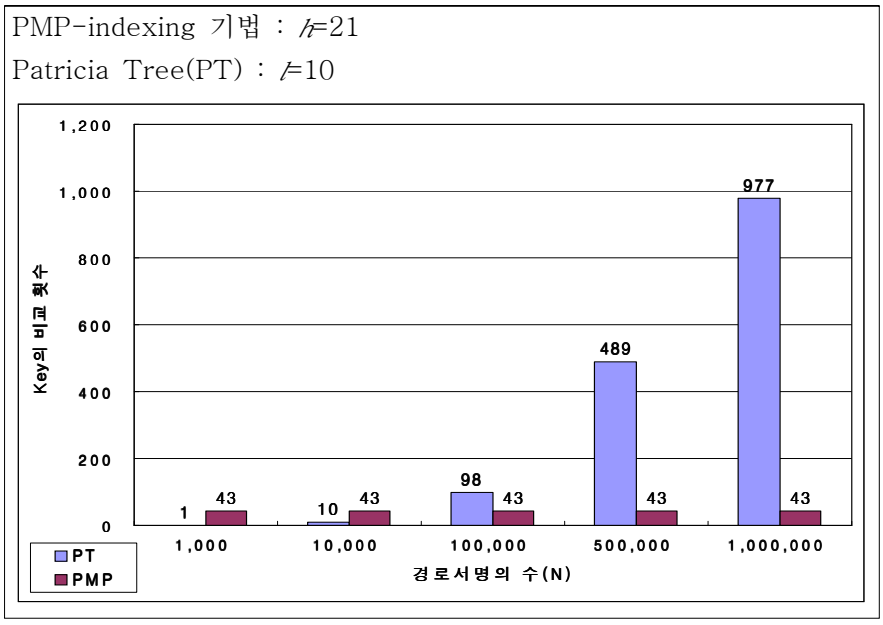


그림 35. 패트리샤 트리(PT)와 PMP-indexing 기법간의 성능의 비교분석 결과

Fig. 35. Result of comparison and analysis of performance between existing method and PMP-indexing method

그림 35에서 보는 바와 같이 경로서명 파일내의 경로서명의 개수(N)가 적

을 때에는 기존의 기법이 성능면에서 우수함을 나타내지만 파일내의 경로서명의 수가 점차 증가함에 따라 제안된 병렬 매치 알고리즘이 더 우수한 성능을 보임을 알 수 있다. 또한, 기존의 기법은 1로 설정된 비트의 수가 많고 적음 또는 경로서명 파일내의 경로서명의 엔트리 수에 따라 수행 성능에 많은 차이를 보인다. 그러나 제안된 기법은 경로서명 파일내의 경로서명의 수가 많고 적음이나 경로서명값에서 1로 설정된 비트의 수가 많고 적음에는 영향을 크게 받지 않으며 오직 경로서명의 길이(h)에만 영향을 받는다. 따라서 중간규모이하의 데이터베이스에 대한 검색에는 기존의 기법을 사용하는 것이 유리하지만 정적인 중간규모 이상의 XML문서 데이터베이스에서는 제안된 기법의 사용이 더 효율적임을 알 수 있다.

본 논문에서 제안한 PMP-indexing 기법을 사용하여 인덱싱 작업을 수행하는 경우에 몇 가지 문제점이 나타날 것으로 예상된다.

병렬 매치 인덱싱 구조에서 PE들을 구현함에 있어서 실제 하드웨어로 구현한다면 성능개선과 비용문제 측면에서 tradeoff가 일어 날 수 있으며, 인덱싱 구조의 유연성 문제가 대두될 것으로 예상된다. 또한, 병렬 시스템을 구현함에 있어서 필연적으로 PE들 사이에 통신오버헤드 문제가 발생할 것이나 본 논문에서는 통신오버헤드 문제는 고려하지 않았다. 하지만 이것을 해결하기 위한 방법에 관한 연구가 필요하다.

또 다른 문제는 PMP-indexing 기법에서도 역시 허위드롭(False drop)을 포함할 잠재적 가능성이 있기 때문에 이것을 해결하기 위한 효율적인 방안에 대한 연구가 필요하다.

4. 실험결과 및 성능비교 분석

본 장에서는 XML문서 데이터베이스로부터 사용자가 입력한 경로-지향 질의어의 평가 성능을 개선하기 위해 본 논문에서 제안한 2가지 기법들과 기존의 기법간의 성능을 비교분석한다.

기존의 기법과 제안된 2가지 기법들 사이의 성능비교 분석 기준은 검색연산 수행시 키의 비교횟수를 기준으로 한다.

기존의 제안된 기법과 본 논문에서 제안한 인덱싱 기법들에 대한 전체적인 실험결과는 그림 36과 같다.

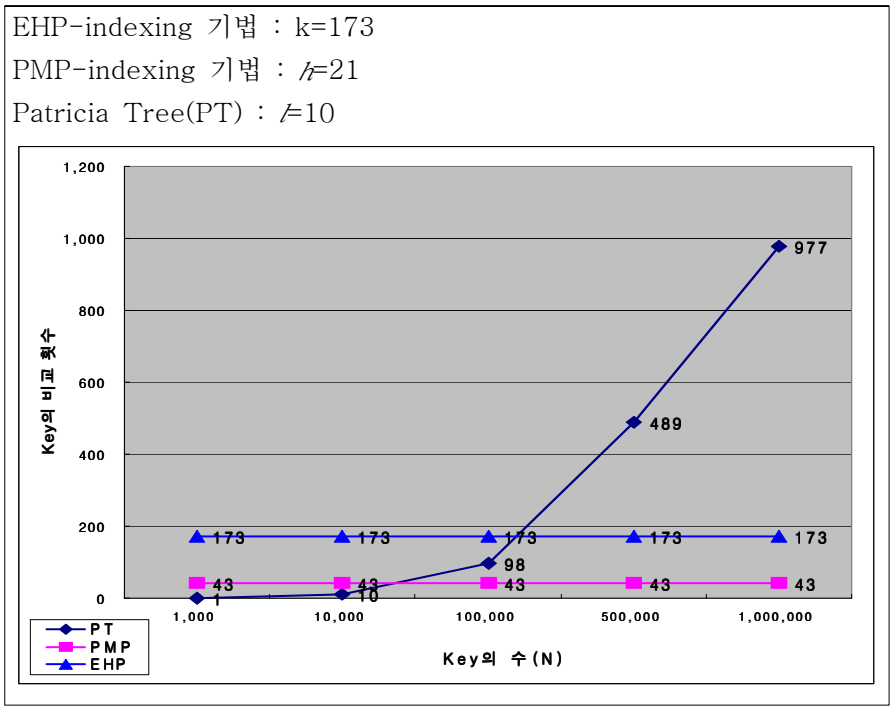


그림 36. 기존 기법과 제안된 기법들 사이의 성능 비교분석 결과
Fig. 36. Result of comparison and analysis of performance between existing method and proposed methods

그림 36에서 보여주는 것과 같이 동일한 데이터를 가지고 실험하였을 경우 기존의 기법과 본 논문에서 제안한 기법들 사이의 키의 비교 횟수를 통한 성능을 비교 분석한 결과는 다음과 같다.

경로-지향 질의어 평가를 위해 검색 연산을 수행할 때 인덱스 키의 수가 점차 증가함에 따라 경로서명 파일과 병렬 인덱싱 구조를 이용한 PMP-indexing 기법이 키의 비교횟수가 가장 적었으며, 다음은 단축경로와 확장성 해싱 인덱싱 구조를 이용하는 EHP-indexing 기법, 경로서명과 패트리샤 트리 구조를 이용하는 기존의 Chen [18]이 제안한 기법 순으로 키의 비교횟수가 증가하는 것으로 나타났다. 따라서 기존의 기법이 키의 비교횟수가 가장 많은 것으로 드러났다. 그러나 키의 개수가 일만 이하의 소규모 데이터베이스를 사용하는 경우에 키의 비교횟수는 기존의 기법이 가장 적었으며, 다음으로 PMP-indexing 기법, EHP-indexing 기법 순으로 키의 비교횟수가 증가하는 것으로 나타났다.

최악의 경우에 있어서 이들 세 가지 기법들에 대한 검색 연산을 수행하기 위해 요구되는 키의 비교를 위한 시간 복잡도는 기존의 기법은 $O(N/2^{\prime})$이며, EHP-indexing 기법은 $O(k)$, PMP-indexing 기법은 $O(h)$이다.

특히, 본 논문에서 제안한 PMP-indexing 기법은 최적의 경우와 최악의 경우 모두 동일한 시간 복잡도를 요구하고 있다.

실험을 수행한 결과 키의 비교 횟수를 기준으로 한 인덱싱 시스템의 성능은 데이터의 양이 증가함에 따라 기존의 기법은 검색 성능이 저하는 되는 반면에 본 논문에서 제안한 기법들은 검색 성능을 향상 시키는 것으로 드러났다. 그러나 본 논문에서 제안한 기법들은 다음과 같은 몇 가지 보완점들을 가지고 있다.

첫 번째로 제안한 EHP-indexing 기법을 이용하는 경우 키의 양이 증가함에 따라 디렉토리와 버킷 파일을 구성하기 위해 더 많은 소요시간을 요구하고 있으며, 확장성 해싱 인덱싱 구조를 생성할 때 더 많은 가상 메모리를 요구하고 있다. 또한, 축약형 위치 경로가 질의어로 입력되는 경우 단축경로 파일의 단순성 문제로 인하여 검색연산이 수행되지 않는 약점을 가지고 있기 때문에 이에 대한 보완이 필요하다.

두 번째로 제안한 PMP-indexing 기법을 실제 하드웨어로 구현하여 사용하는 경우 그 구조의 변경에 있어서 유연성이 떨어진다. 그렇기 때문에 빈번하게 추가 또는 삭제연산이 발생하는 XML문서 데이터들에 대해 사용하는 경우에는 효율성이 다소 떨어질 것으로 예상된다. 따라서 그 내용이 자주 변하지 않는 정적인 XML문서 데이터베이스에 사용하는 것이 효과적이다. 또 다른 문제점은 병렬형 아키텍처 구현의 특성상 내재되어 있는 PE들 사이에 발생하는 통신 오버헤드 문제를 들 수 있다.

5. 결론 및 향후 연구과제

XML문서들에 대한 경로-지향 질의어가 사용자에 의해 입력될 때, 데이터베이스시스템은 데이터베이스 내부에 저장된 정보의 위치를 찾기 위해 질의어가 포함하고 있는 경로들에 대한 질의 평가를 수행 하여야만 한다.

본 논문에서는 관계형 데이터베이스에 저장된 XML문서에 대한 경로-지향 질의어의 평가속도를 개선하기 위한 두 가지의 새로운 인덱싱 기법들을 제안하고 이들의 성능 실험을 수행하였다.

첫 번째로 제안한 EHP-indexing 기법은 단축경로 파일과 확장성 해싱 인덱싱 구조를 결합함으로서 입력되는 키의 수에 관계없이 최악의 경우에도 항상 일정한 검색성능을 나타낸다는 것을 보였다. 또한, 실험에서 인덱스 키의 양에 따라서 하나의 버킷에서 요구되는 최저 슬롯수가 존재함을 알게 되었으며, 최저 슬롯의 수 이하로 각 버킷의 슬롯 수를 결정하는 경우에는 인덱싱 구조를 구성할 때 효율성이 떨어지는 것으로 나타났다.

두 번째로 제안한 PMP-indexing 기법은 관계형 데이터베이스에 저장된 XML문서에 대한 경로-지향 질의어 처리 평가 속도를 개선하기 위해 경로서명 파일과 병렬 매치 인덱싱 구조를 결합하여 사용하고 있다. 이 기법은 그 성능을 실험하기 위해 병렬 매치 알고리즘이 필요하다. 따라서 본 논문에서는 병렬 매치 알고리즘에 대한 설계와 구현을 통하여 PMP-indexing 기법에 대한 성능 실험을 수행하였다.

제안된 두 가지 기법 모두는 이진 트라이 구조에 기반을 둔 것이다. 이진 트라이는 이진 비트 패턴에서 연속적인 비트값들을 이용하여 루트에서부터 아

래쪽으로 향하는 트라이의 경로를 결정한다.

본 논문에 제안한 기법들은 XML문서에 포함된 각 엘리먼트들의 경로 정보를 저장하기 위해 단축경로와 경로서명을 생성하고, 이들을 각각 단축경로 파일과 경로서명 파일에 저장한 후 이진 트라이 구조를 형성하는데 활용하였다. 두 가지 제안된 기법에서는 이진 트라이 구조를 각각 확장성 해싱 인덱싱 구조와 병렬 매치 인덱싱 구조로 변환하였다.

첫 번째로 제안된 기법을 이용한 성능 실험 분석 결과에 따르면 EHP-indexing 기법은 XML문서 파일에 대한 인덱스 파일의 엔트리 수와 관련하여 log_2N과 유사한 크기 순서를 가진다. 따라서 이 기법은 인덱스 파일(단축경로 파일)의 엔트리의 수와 단축경로의 비트 패턴의 형태 모두에 영향을 받지 않으며, 버킷의 슬롯수(k)에만 영향을 받기 때문에 키의 검색을 위한 비교횟수는 최악의 경우에도 시간 복잡도는 $O(k)$이다.

두 번째로 제안된 PMP-indexing 기법을 이용한 성능 실험 분석 결과에 따른 키 검색을 위한 비교횟수는 인덱스 파일(경로서명 파일)의 엔트리의 수와 경로서명의 비트 패턴의 형태에 관계없이 오직 인덱스 키로 사용하는 경로서명의 길이(h)에만 영향을 받기 때문에 최악의 경우에도 시간 복잡도는 $O(h)$이다.

기존의 기법은 인덱스 키의 검색을 위해 최악의 경우에 있어서 키의 최대 비교횟수로 $O(N/2^l)$의 시간 복잡도를 요구하고 있지만 제안된 기법들은 각각 $O(k)$와 $O(h)$를 요구하고 있기 때문에 데이터의 양이 점차 증가함에 따라서 기존의 기법을 사용하는 것보다는 제안된 기법을 사용하는 것이 보다 효율적임을 알 수 있다. 즉, 중간 규모 이상의 데이터베이스에서는 제안된 기법들을 사용하는 것이 유리하다.

향후 연구과제로는 다양한 형태로 입력되는 경로-지향 질의어를 대상으로 하여 실험이 실시되어야 하며, 효과적으로 허위드롭(False drop)을 해결하기 위한 방안에 대한 연구와 함께 인덱스 키의 양에 따른 한 개의 버킷이 가져야 할 가장 적정한 슬롯의 수를 결정하는 문제에 관한 증명이 요구된다. 또한, 병렬 매치 인덱싱 구조에서 PE들을 실제 하드웨어로 구현하고 이에 대한 실험 및 결과 분석에 따른 정확한 성능 평가가 이루어져야 할 것이며, 통신오버헤드 문제 해결을 위한 개선책에 대한 연구가 필요하다.

참 고 문 헌

[1] S. Abiteboul and S. Cluet and V. Christophides and T. Milo and G. Moerkotte and J. Simeon, “Querying documents in object databases”, *Int. Journal on Digital Libraries, Vol. 1*, pp. 5-19, 1997.

[2] S. Abiteboul and S. Cluet and T. Milo, “Querying and updating the file”, *Proc. of the 9th Int. Conference on Very Large Data Bases (VLDB)*, pp. 386-397, 1993.

[3] A. V. Aho and J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley Publishing Company, 1974.

[4] F. Bancihon and C. Delobel and P. Kanellakis, *Building an Object-Oriented Database System: The Story of O2*, Morgan Kaufmann, 1992.

[5] R. Bayer and K. Unterrauer, “Prefix B-tree,” *ACM Transactions on Database Sys-tems, Vol. 2*, pp. 11-26, 1998.

[6] Brian Benz and John R, Durant, *XML Programming Bible*, Wiley, 2003.

[7] E. Bertino, “Optimization of queries using nested indices”, *Proc. of Int. Conference on Extending Database Technology*, pp. 44-59, 1990.

[8] E. Bertino and C. Guglielmani, “Optimization of object-oriented

queries using path indices", *2nd Int. Workshop on Research Issues on Data Engineering: Transaction and Query Processing*, pp. 140-149, 1992.

[9] E. Bertino and W. Kim, "Indexing techniques for queries on nested objects", *IEEE Transactions on Knowledge and Data Engineering, Vol. 1*, pp. 196-214, 1989.

[10] S. Boag and D. Chanberlin and M. Fernandez and D. Florescu and D. Florescu and J. Robie and J. Simeon, "XQuery 1.0 : An XML Query Language", http://www.w3.org/TR/2002/WD-xquery-20020816, Aug. 2002.

[11] Fidel Cacheda and Angel Vina, "Superimposing Codes Representing Hierarchical Information in Web directories", *3rd Int. Workshop on Web Information and Data Management(WIDM 2001)*, pp. 54-60, 2001.

[12] A. Cardenas, "Analysis and performance of inverted data base structures", *Communications of ACM, Vol. 18*, pp. 253-263, 1987.

[13] Patrick Carey, *New Perspectives on XML- Comprehensive (Paperback)*, Course Technology, 2003.

[14] R. G. G. Cattell, *Object Data Management: Object-Oriented and Extended Relational Database Systems*, Addison-Wesley Publishing Company, Inc., 1991.

[15] C. Y. Chan and C. H. Goh and B. C. Ooi, "Indexing OODB instances based on access proximity", *Proc. of 13th Int.*

Conference on Data Engineering, pp. 14-21, 1997.

[16] Yangjun Chen, "On the Signature Trees and Balanced Signature Trees", *Proc. of the 21st Int. Conference on Data Engineering*, pp. 742-753, 1993.

[17] Yangjun Chen, "Building Signature Trees into OODBs", *Journal Of Information Science And Engineering 20*, pp. 275-304, 2004.

[18] Y. Chen and G. Huck, "Path signature: A Way to Speed up Evaluation of Path-oriented Queries in Document Databases", *WISE2000*, pp. 240-244, 2000.

[19] S. Choenni and E. Bertino and H. M. Blanken, and T. Chang, "On the selection of optimal index configuration in OO databases", *Proc. of 10th Int. Conference on Data Engineering*, pp. 526-537, 1994.

[20] S. Christodoulakis and C. Faloutsos, "Design consideration for a message file server", *IEEE Transactions on Software Engineering, Vol. 10, pp. 201-210*, 1984.

[21] Brian F. Cooper et al., "A Fast Index and Querying XML Data for Regular Path Expression", *Proc. of the 27th VLDB Conference*, 2001.

[22] M. Crochemore and W. Rytter, *Text Algorithms*, Oxford University Press, 1994.

[23] Harvey M. Deitel and Paul J. Deitel and T. R. Nieto, Ted Lin, Praveen Sadhu, *XML HOW TO PROGRAM*, Prentice Hall, 2001.

[24] U. Deppisch, “S-tree: A dynamic balanced signature index for office retrieval”, *ACM SIGIR Conference*, pp. 77-87, 1986.

[25] D. Dervos and P. Linardis and Y. Manolopoulos, “Perfect Encoding : a Signature Method for Text Retrieval”, *a selection of ADBIS'96 papers, the British Computer Society eWiC Series*, pp. 176-181, 1996.

[26] D. Dervos and Y. Manolopoulos, and P. Linardis, “Comparison of signature file models with superimposed coding”, *Journal of Information Processing Letters, Vol. 65*, pp. 101-106, 1998.

[27] A. Deutch and M. Fernandez and D. Forescu and A. Levy and D. Suciu, “XML-QL : A Query Language for XML”, http://www.w3.org/TR/NOTE-xml-ql, Aug. 1998.

[28] D. H. C. Du and Member and IEEE and S. Tong, “Multilevel Extensible Hashing: A File Structure for Very Large Database”, *IEEE Trans. on Knowledge and Data Eng, Vol 3*, *No.3*, 1991.

[29] D. Eastlake and J. Reagle and D. Solo and W3C Recommendation, “XML-Signature Syntax and Processing”, http://www.w3.org/TR/2002/REC-xmldsig-core-20020212, Feb. 2002.

[30] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*, Benjamin Cumming, California, 1989.

[31] Chris Faloutsos, “Signature files: Design and performance comparison of some signature extraction methods”, *Proc. of the 1985 ACM SIGMOD Int. Conference on Management of Data*, pp. 63-82, 1985.

[32] C. Faloutsos and ACM Computing Surveys, “Access Methods for Text”, *Vol. 17, No.1*, pp. 48-74, 1985.

[33] M. J. Folk and B. Zoellick and G. Riccardi, *File Structures An Object-Oriented Approach with C++*, Addison-Wesley, 1998.

[34] F. Fotouhi, T. G. Lee and W. I. Grosky, “The generalized index model for object-oriented database systems”, *10th Annual Int. Phonix Conference on Computers and Communication*, pp. 302-308, 1991.

[35] William B. Frakes and Ricardo Baeza-Yates, *Information Retrieval : Data Structures and Algorithms*, Prentice Hall PTR; Facsimile edition, pp. 44-80, 1992.

[36] S. W. Golomb, “Run-length encoding”, *IEEE Transactions on Info-rmation Theory, Vol. 12*, pp. 399- 401, 1996.

[37] Fabio Grandi and Paolo Tiberio and Pavel Zezula, “Frame-Sliced Partitioned Parallel Signature Files”, *Proc. of the 15th Int. ACM SIGIR Conference on Research and development in information retrieval*, pp. 286-287, 1992.

[38] Fabio Grandi, “On the Signature Weight in multiple m Signature Files”, *ACM SIGIR Forum(Sec. Refereed papers) 29:1*, pp. 20-25, 1995.

[39] R. Haskin, “Special purpose processors for text retrieval”, *Database Engineering, Vol. 4*, pp. 16-29, 1981.

[40] S. Helmer and T. Neumann and G. Moerkatte, “A Robust Schema

for Multilevel Extendible", *Proc. 18th Int. Sym. on ISCIS, Antalya*, pp. 220-227, 2003.

[41] Y. Ishikawa and H. Kitagawa and N. Ohbo, "Evaluation of signature files as set access facilities in OODBs", *Proc. of ACM SIGMOD Int. Conference on Management of Data*, pp. 247-256, 1993.

[42] A. Kemper and G. Moerkotte, "Access support relations: an indexing method for object bases", *Information Systems, Vol. 17*, pp. 117-145, 1992.

[43] W. Kim, *Introduction to Object-Oriented Databases*, The MIT Press, 1990.

[44] W. Kim, "A model of queries for object-oriented databases", *Proc. of Int. Conference on Very Large Data Base*, pp. 423-432, 1989.

[45] W. Kim, K. C. Kim, and A. Dale, *Indexing Techniques for Object Oriented Databases*, Addison Wesley, pp. 371-394, 1989.

[46] Hiroyuki KITAGAWA and Yoshiharu ISHIKAWA, "False Drop Analysis of Set Retrieval with Signature Files", *IEICE TRANS. INF. & SYST, Vol. E80-D, No.6*, pp. 1-12, 1997.

[47] D. E. Knuth, *The Art of Computer Programming: Sorting and Searching*, Addison- Wesley Publishing, London, 1973.

[48] SEYIT KOCBERBER, FAZLI CAN, JON M. PATTON, “Optimization of Signature File Parameters for Databases with Varying Record Lengths”, *THE COMPUTER JOURNAL, Vol. 42, No. 1*, pp. 11-23, 1999.

[49] Bin Lan and Beng Chin Ooi and Kian-Lee Tan, “Efficient Indexing Structures for Mining Frequent Patterns”, *Proc. of the 18th Int. Conference on Data Engineering*, pp. 453-462, 2002.

[50] W. Lee and D. L. Lee, “Signature file methods for indexing object-oriented database systems”, *Proc. of 2nd Int. Conference on Data and Knowledge Engineering: Theory and Application*, pp. 616-622, 1992.

[51] Q. Li and B. Moon, “Indexing and Querying XML Data for Regular Path Expression”, *Proc. of the 27th VLDB Conference*, 2001.

[52] C. C. Low and B. C. Ooi and H. Lu, “H-trees: a dynamic associative search index for OODB,” *Proc. of 1992 ACM SIGMOD Conference on the Management of Data*, pp. 134-143, 1992.

[53] D. Maier and J. Stein, “Indexing in an object-oriented DBMS”, *Proc. of Int. Workshop on OODB Systems*, pp. 171-182, 1986.

[54] H. Mochizuki and M. Koyama and M. Shishibori and J. Aoe, “A substring search algorithm in extendible hashing”, *Informatics and Computer Science: An Int. Journal Vol. 108, Issue 1-4*, pp. 13-30, 1998.

[55] A. Moffat and J. Zobel, “Self-indexing inverted files for fast text

retrieval", *ACM Transaction on Information Systems, Vol. 14*, pp. 349-379, 1996.

[56] D. R. Morrison, "PATRICIA-practical algorithm to retrieve information coded in alphanumeric", *Journal of ACM, Vol. 15*, pp. 514-534, 1968.

[57] T. A. Mueck and M. L. Polaschek, "The multikey type index for persistent object sets", *Proc. of 13th Int. Conference on Data Engineering*, pp. 22-31, 1997.

[58] Kjetil Norvag, "Efficient Use of Signature in Object-Oriented Database Systems", *ADBIS '99, LNCS 1691*, pp. 367-381, 1999.

[59] Ajay M. Rambhia, *XML Distributed System Design*, SAMS, 2002.

[60] 박희숙, 조우현, "단축경로와 확장성 해싱 기법을 이용한 경로-지향 질의의 평가속도 개선 방법", *정보처리학회논문지, 제11-D권, 제7호*, pp. 1409-1416, 2004.

[61] 박희숙, 조우현, "XML 데이터베이스에서 경로-지향 처리를 위한 병렬 매치 방법", *정보과학회논문지, 데이터베이스 제32권, 5호*, pp. 1-9, 2005.

[62] H. Shin and J. Chang, "A New Signature Scheme for Query Processing in Object-Oriented Database", *Proc. of the 20th Int. Conference on Computer Software and Applications*, pp. 400-405, 1996.

[63] B. Sreenath and S. Seshadri, "The hcC-tree: an efficient index structure for object oriented database", *Proc. of Int. Conference*

on Very Large Database, pp. 203-213, 1994.

[64] S. Stanfill and B. Kahle, "Parallel free-text search on the connection machine system", *Communication of ACM, Vol.29, No.12*, pp. 1229-1239, 1986.

[65] R. Tarjan, "Depth-first search and linear graph algorithms", *SIAM Journal on Computing, Vol. 1*, pp. 146-150, 1972.

[66] Nassrin Tavakoli and Alan Ray, "A New Signature Approach for Retrieval of Documents from Free-Text Databases", *Information Processing and Management: int. Journal, Vol. 28 No.2*, pp. 153-163, 1992.

[67] J. D. Ullman, *Computational Aspects of VLSI*, Computer Science Press, Maryland, 1984.

[68] I. H. Witten and A. Moffat and T. C. Bell, *Managing Gigabytes*, Van Nostrand Rein-hold, 1994.

[69] W3C, "XML-QL: A Query Language for XML", http://www.w3.org/TR/1998/NOTE-xml-ql-19980819/#intro, 1998.

[70] W3C, "XML Path Language (XPath) 2.0", http://www.w3.org/2003/08/DIFF-xpath20, 2003.

[71] W3C, "Extensible Markup Language (XML)1.0", http://www.w3.org/XML/, 1998.

[72] W3C, "XML Query (XQuery) Requirements", http://www.w3.org/TR/2003/WD-xquery-requirements-20031112, 2003.

[73] W3C, "Document Object Model (DOM)", http://www.w3.org/DOM/,

2002.

[74] H. Yokota, Y. Kanemasa, and J. Miyazaki, “Fat-Btree: an update-conscious parallel directory structure”, *Proc. of 15th Int. Conference on Data Engineering*, pp. 448-457, 1999.

[75] H. S. Yong, S. Lee, and H. J. Kim, “Applying signatures for forward traversal query processing in object-oriented databases”, *Proc. of 10th Int. Conference on Data Engineering*, pp. 518-525, 1994.

[76] C. Zaniolo et al, *Advanced Database Systems*, Morgan Kaufmann Publishers, 1997.

[77] J. Zobel, A. Moffat, and K. Ramamohanarao, “Inverted files versus signature files for text indexing”, *ACM Transactions on Database Systems, Vol. 23*, pp. 453-490, 1998.

감사의 글

본 논문이 완성되기까지 인자하심과 아낌없는 배려로 많은 학문적인 가르침과 바른 학문의 길을 갈수 있도록 등불이 되어 주신 존경하옵는 조우현 교수님께 가슴 깊이 감사드립니다. 교수님의 가르침을 가슴 깊이 간직하여 언제나 바른길을 가는 사람이 되도록 노력하겠습니다.

바쁘신 가운데에도 본 논문을 심사해 주시고 많은 학문적인 조언과 좀 더 나은 논문이 될 수 있도록 보완점을 제시해 주신 정목동 교수님, 윤성대 교수님, 류시국 교수님, 권오흠 교수님께도 머리 숙여 감사드립니다. 또한, 박사과정 중에 물심양면으로 많은 학문적인 가르침을 주신 우종호 교수님, 조경연 교수님, 서경룡 교수님, 신봉기 교수님, 송하주 교수님, 김종남 교수님께도 깊은 감사를 드립니다.

긴 세월 동안 연구실에서 함께 동고동락을 하였던 후배 박주현, 정성주, 이상호, 김락기, 김상훈, 이학국, 김미영, 졸업한 김성록 후배 그리고 마이크로프로세서 연구실의 박사과정 동기 박창수 선생님에게도 감사드립니다. 그리고 저에게 많은 정신적인 도움을 준 친구 김원화, 박은숙, 유경아, 김정희 선생님, 김숙연 선생님, 탁충수 선생님, 하태경 선생님, 윤선정 선생님에게도 고마움을 전합니다.

항상 이 자식이 잘되기만을 기도하시고 노심초사 하셨던 부모님을 비롯하여 형제, 자매 그리고 형부와 제부에게도 감사드립니다.

마지막으로 힘겨워 할 때마다 저에게 항상 힘이 되어주고 많은 격려와 용기를 북돋아 주신 박진상님에게도 깊은 감사를 드립니다.

2006년 2월 박 희 숙 드림