URL Prefix 해시 트리를 이용한 URL 목록 검색 속도 향상

(Fast URL Lookup Using URL Prefix Hash Tree)

박 창 욱 † 황 선 영 ††
(Chang-Wook Park) (Sun-Young Hwang)

요 약 본 논문에서는 URL 목록 기반 웹사이트 컨텐츠 필터링 시스템에서 효율적인 URL 목록 검색 방식을 제안한다. 제안된 방식은 URL prefix 형태로 변환된 URL 목록을 해시 트리 형식으로 표현하여 한번의 트리 검색으로 URL 검색을 수행한다. 그 결과 단일 해시 테이블 방식의 중복 탐색을 제거하였다. 실험 결과 제안된 검색 방식은 세그먼트의 개수에 따라 단일 해시 테이블 방식에 비해 62%~210%의 성능 향상을 보인다.

키워드 : 해시 트리, URL 목록 검색

Abstract In this paper, we propose an efficient URL lookup algorithm for URL list- based web contents filtering systems. Converting a URL list into URL prefix form and building a hash tree representation of them, the proposed algorithm performs tree searches for URL lookups. It eliminates redundant searches of hash table method. Experimental results show that proposed algorithm is 62%~210% faster, depending on the number of segment, than conventional hash table method.

Key words : hash tree, URL lookup

1. 서 론

최근 웹과 인터넷이 대중화되면서 그 역기능이 문제가 되어 웹사이트 콘텐츠 필터링 시스템의 필요성이 대두되었고, 콘텐츠 필터링을 효율적으로 수행하기 위한 많은 방식들이 제안되었다.

1.1 판단 기준에 따른 필터링 시스템의 분류

콘텐츠 필터링 시스템은 판단의 기준에 따라 URL 목록 기반 방식과 내용 기반 방식, 이들을 결합한 하이브리드 방식으로 분류된다[1,2]. 그림 1에 콘텐츠 필터링 시스템의 판단 기준에 따른 시스템의 분류를 보인다.

URL 목록 기반 방식은 시스템 내에 미리 수집, 분석된 URL 목록을 가지고 있으며 HTTP 요청을 분석하여 사용자가 방문하려는 사이트의 주소가 이 목록에 있는지를 비교하여 접근을 허용, 또는 차단하는 방식이다[3,4]. 접근 금지 URL 목록을 가지고 있는 경우 블랙리스트 URL 목록 기반 방식이라 하고, 접근 허용 URL 목록을 가지고 있는 경우 화이트리스트 URL 목록 기반 방식이라 한다. 내용 기반 방식은 사용자 측으로 향하는 웹페이지의 내용을 기반으로 하여 차단하는 방식이며 이는 다시 내용 등급 표기 기반 방식과[5], 텍스트와 이미지를 포함하는 내용 해석 방식으로 분류된다[6-8]. 내용 등급 표기 기반 방식은 웹 사이트의 관리자가 표시한 등급에 따라 차단 여부를 결정하는 방식이고 내용 해석 방식은 사용자에게 보여주기 전에 프로그램이 매번 내용을 해석하여 차단 여부를 결정하는 방식이다. 하이브리드 방식은 사용자의 HTTP 요청에 대해 URL 목록 기반 방식을 적용하여 필터링을 수행하고 HTTP 응답에 대해 내용 기반 방식을 적용하는 방식이다.

내용 등급 표기 기반 방식은 웹사이트 관리자가 자율적으로 자신의 사이트에 매긴 등급을 기반으로 필터링을 수행한다. 현재 등급이 표기되지 않는 사이트가 많이 존

· 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT 연구센터 육성지원사업의 연구결과로 수행되었음

† 정 회 원 : 플랜티넷 기술연구소 수석연구원
chang@plantynet.com
†† 종신회원 : 서강대학교 전자공학과 교수
hwang@ccs.sogang.ac.kr

논문접수 : 2005년 6월 14일
심사완료 : 2007년 10월 17일

정보과학회논문지: 정보통신 제35권 제1호(2008.2)

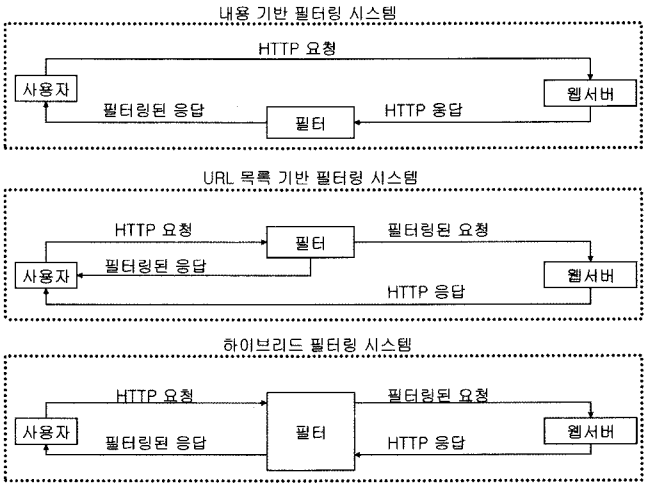

그림 1 판단 기준에 따른 필터링 시스템의 분류

재한다는 것과, 표기된 등급이 필터링 정책과 서로 다르다는 것이 문제점이다. 내용 해석 방식은 해석에 시간이 많이 걸려 실시간 해석이 어려워 사용자의 체감 속도 저하가 심하고, 해석이 올바르지 못한 경우가 많다는 것이 문제점이다. 하이브리드 시스템은 여러 방식을 동일 시스템 내에 내장하여 시스템이 복잡해져 필터링 효율이 떨어지는 단점이 있다. URL 목록 기반 방식은 URL 목록의 구축에 많은 비용이 드는 단점이 있으나 HTTP 응답을 조사하지 않아도 되는 장점으로 인하여 구조가 단순한 장점이 있다. 일반적인 컨텐츠 필터링 시스템의 경우 접근을 금지해야 할 URL 목록의 수보다 접근을 허용해야 할 URL 목록의 수가 많으므로 블랙리스트 URL 목록 기반 방식이 목록 관리 측면에서 효율적이다. 따라서 현재까지 대부분의 컨텐츠 필터링 시스템은 블랙리스트 URL 목록 기반 방식을 채택하고 있다.

1.2 설치 위치에 따른 필터링 시스템의 분류

필터링 시스템은 필터링 시스템의 설치 위치에 따라 네트워 기반 필터링 시스템과 클라이언트 기반 필터링 소프트웨어로 분류된다. 그림 2에 설치 위치에 따른 필터링 시스템의 분류를 보인다. 클라이언트 기반 필터링 소프트웨어는 사용자의 컴퓨터에 설치된다. 네트워 기반 필터링 시스템은 사용자와 접근하려는 웹 사이트의 중간 네트워에 설치된다. 클라이언트 기반 필터링 소프트웨어는 한 명의 사용자가 발생시키는 트래픽만을 처리하므로 성능상에 제약을 받지 않는 장점이 있으나 사용자가 임의로 필터링 소프트웨어를 제거하여 무력화할 수 있는 위험이 있다. 네트워 기반 필터링 시스템은 사용자가 임의로 제거할 수 없는 장점이 있고 관리의 용이성이 있으나 여러 사용자의 트래픽을 한꺼번에 처리하므로 고성능 시스템이 필요하다.

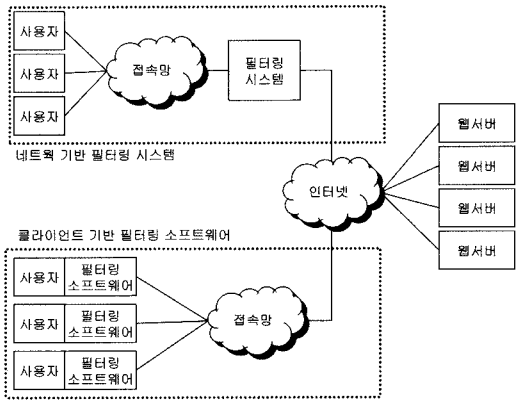

그림 2 설치 위치에 따른 필터링 시스템의 분류

1.3 네트워 기반 블랙리스트 URL 목록 방식 시스템

내용 기반 방식과 하이브리드 방식은 성능의 제약을 덜 받는 클라이언트 기반 필터링 소프트웨어에 많이 쓰이고 URL 목록 기반 방식은 고성능 시스템이 필요한 네트워 기반 필터링 시스템에 많이 쓰인다. 본 논문에서는 네트워 기반, URL 목록 기반 방식 필터링 시스템의 성능을 높이기 위한 구조와 알고리즘을 제안한다. 그림 3에 네트워 기반 블랙리스트 URL 목록 방식 시스템의 동작을 보인다. 네트워 기반 URL 목록 방식 시스템의 필터링 성능은 URL 목록 참조 속도에 좌우된다. 네트워 기반 블랙리스트 URL 목록 방식 시스템에 사용되는 URL 목록의 개수는 수억 개 이상으로 방대하다. 갱신된 URL 목록을 외부 시스템에서 제공할 수 있으므로 시스템의 동작 중에는 URL 목록에 URL을 추가하는 동작이 없으며, URL 참조만을 수행하는 특징이 있다.

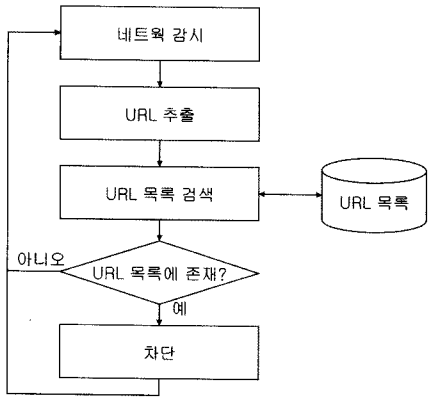

그림 3 네트웍 기반 블랙리스트 URL 목록 방식 시스템의 동작

URL 목록에는 domain, Host, IP, directory, file 등이 다양하게 혼재되어 있다. 사용자의 컴퓨터에 설치되는 필터링 시스템과는 달리 트래픽의 고속 실시간 처리가 필요하고, 이를 위해 제한된 시간 내에 처리를 마쳐야 하므로 URL 목록을 디스크에 둘 수 없고 모두 메모리 상에 두어야 한다. 이 조건을 모두 만족시키는 효율적인 자료 구조와 알고리즘이 필요하다.

URL 목록에서 URL을 탐색하는 문제에 대한 연구는 여러 분야에서 진행되어 왔다. 콘텐츠 기반 스위칭을 위해 URL 탐색을 수행하는 전용 하드웨어에 대한 연구[9]도 진행되었으나 CAM을 사용해야만 하는 특징으로 인해 소프트웨어로 구현하기 어렵고 소프트웨어로 구현 가능한 방식[10]도 있으나 대용량 URL 목록 구축이 필요한 콘텐츠 필터링에 쓰이기에는 적합하지 않다. 해싱을 이용한 URL의 탐색은 웹 크라울러[11], 웹 캐시[12]에서도 진행되었으나, URL 목록 내에 특정 URL의 존재 유무를 알아내는 목적으로 쓰이고 URL 목록에 저장된 URL과의 유사성을 비교하는 콘텐츠 필터링용에는 적합하지 않은 단점이 있다. 패킷 분류 알고리즘[13]을 사용하면 IP 목록 탐색과 처리에는 효과적이나 URL 목록의 처리에는 적합하지 않다. 다차원 해시 트리와 프로그레시브 해싱를 이용한 URL 비교 방법[14]은 해시 트리에 URL을 삽입함에 따른 해시 테이블의 재할당과 충돌 회피를 효과적으로 처리한 방식이다. 이 방식은 해시 트리 구축과 검색을 동시에 수행하는 데에는 효과적이나 해시 트리가 방대해 질 경우 제안하는 검색 전용 해시 트리에 비해 속도와 메모리 비용 측면에서 불리한 단점이 있다.

본 논문에서 제안하는 방식은 URL 목록 검색을 함에 있어 해시 트리 구축 과정과 해시 트리 검색 과정을 분리하였다. 제안하는 방식은 해시 트리 구축에는 많은 비용이 드는 단점이 있으나 검색에는 사전에 최적화된 해시 트리를 이용할 수 있어 속도와 메모리 비용 측면에서 모두 유리한 방식이다.

2. URL 목록 검색

2.1 URL Prefix Form

인터넷 상에 존재하는 개별 URL의 개수는 수십억 개에 이른다[15]. 개별 URL을 모두 URL 목록에 추가하는 방식을 사용하면 URL 목록의 수집, 검사, 관리가 매우 어렵고 목록의 개수가 방대하여 필터링 시스템을 만드는 것이 현실적으로 불가능하다. IP 주소만으로 목록을 구성할 경우 목록 참조를 매우 빠른 속도로 수행할 수 있는 장점이 있으나 한 IP에 여러 사이트를 동시에 호스팅하는 경우 그들을 분리하여 필터링하는 것이 불가능하다. URL로부터 도메인과 호스트 이름만 입력하여 URL 목록을 구성하면 URL 목록의 개수를 대폭 줄일 수 있는 장점이 있으나, 한 호스트 내에 필터링해야 할 콘텐츠와 그렇지 않은 콘텐츠가 혼재할 경우 필터링이 불가능한 단점이 있다. URL 목록의 수를 줄이고 필터링의 정확도를 유지하는 방법은 도메인, 호스트, 디렉토리, 개별 URL 등을 섞어서 URL 목록을 구성하는 것이다. 이 경우 단점은 해당 URL이 목록에 존재하는지를 알아내기 위해서는 검색하고자 하는 URL의 각 부분들, 즉, 도메인, 호스트, 디렉토리, 파일 순으로 여러 번 참조해야 한다는 점이다. 이 순서대로 검색을 하기 위해 URL을 쪼개어 도메인, 호스트, 디렉토리, 파일의 순으로 정렬하여 1차원 배열로 표현한 것을 URL Prefix Form 이라 정의한다. 예를 들어 www.host.com/dir1/dir2/file.html을 URL Prefix Form으로 변환하면 {".com", ".host", ".www", "/dir1", "/dir2", "/file.html"}이다. URL Prefix 배열의 각 요소들을 URL Segment 라 부른다. 각 Segment 의 첫번째 글자에 "." 과 "/" 을 추가하여 호스트 부분과 디렉토리 부분을 구분한다. URL Prefix Form은 URL 탐색에 용이하도록 URL을 변형한 것이다. URL Prefix는 여러 개별 URL들의 집합을 나타내므로 임의의 두 개의 URL Prefix 사이에는 포함관계가 성립할 수 있다. 두개의 URL Prefix U1, U2 사이에 다음과 같은 관계가 있을 경우 URL Prefix U1은 URL Prefix U2를 포함한다고 정의한다.

U1,U2 : segment 들의 배열
U[k] : U의 k 번째 segment
N1 : U1의 세그먼트들의 개수
N2 : U2의 세그먼트들의 개수
k1 : 0 부터 N1-1까지의 정수,
if (N1 <= N2) and (U1[k1] = U2[k1]) for all k1,
then U1⊐U2

URL 목록에서 주어진 URL의 존재 여부를 검색하는 문제는 URL 목록을 URL Prefix Form으로 변환을 할 경우 주어진 URL 포함하는 URL Prefix가 존재하는지의 여부를 알아내는 문제와 같다.

2.2 기존의 단일 해시 테이블 방식 검색

URL 목록을 검색하는 방법으로 한 개의 해시 테이블에 모든 URL 목록을 보관하고 검색하는 방식이 있다. 그림 4에 URL 목록과 이를 해시 테이블로 구성한 예를 보인다. URL 목록에 존재하는 URL의 개수에 따라 적당한 크기의 해시 테이블을 생성한다. 해시 함수를 통해 각 URL들의 해시 테이블에서의 위치를 결정하여 이 위치에 URL들을 저장한다. "*"가 포함된 URL들은 "*" 위치에 임의의 문자열이 위치하는 URL들과 모두 대응됨을 나타낸다. "*"가 포함되지 않은 URL들은 단 1개의 URL을 나타낸다. 해시 테이블에 저장할 때에는 "*"를 제거하고 segment 들을 하나의 문자열로 표현한 URL prefix 문자열을 저장한다.

구성된 해시 테이블에서 주어진 URL을 포함하는 URL Prefix의 존재 여부를 검색하는 알고리즘을 그림 5에 보인다.

U는 검색하고자 하는 URL의 URL Prefix Form이다. Table은 URL 목록이 들어 있는 해시 테이블이다.

URL 목록

12.34.56.*
.domain.com/
www.host.com/dir1/file1.html
www.host.com/dir2/*
www.host.com/dir3/file3.html
data1.site.co.kr/*
data2.site.co.kr/*

해시 테이블

name
12.34.56
NULL
domain.com
NULL
NULL
www.host.com/dir1/file1.html
NULL
data1.site.co.kr
NULL
NULL
NULL
www.host.com/dir3/file3.html
NULL
NULL
data2.site.co.kr
NULL
www.host.com/dir2
NULL

그림 4 URL 목록과 해시 테이블 구성의 예

U 배열의 첫번째 요소인 도메인부터 시작하여 순차적으로 테이블 참조를 수행한다. URL_Concatenation() 함수는 URL Prefix Form의 각 세그먼트들을 합쳐 원래의 URL을 만들어내는 함수이다. HashFunction() 함수는 문자열로부터 해시 인덱스 값을 되돌리는 함수이다. 해시 테이블 구성시 해시 인덱스의 충돌이 감지되면 다음 인덱스 위치에 저장하는 open linear addressing을

```
Algorithm Search_Table (U, Table)
{
        U : An array of given URL in URL Prefix form {U[0], U[1], U[2], ... , U[N]}
        Table : URL HashTable

        i = 0
        while (i <= N)
        {
                // i 까지의 세그먼트들로부터 URL 문자열을 만든다.
                prefix = URL_Concatenation (U[0], U[1], ... , U[i])
                // 해시 인덱스를 구한다.
                index = HashFunction (prefix)
                while (Table[index] != NULL)
                {
                        // 일치하는 엔트리가 발견되면 중단한다.
                        if (Table[index] == prefix) return (FOUND)
                        index = (index + 1) % Table.Size
                }
                // 세그먼트를 하나 더 추가하여 다시 검색한다.
                i = i + 1
        }
        return (NOT_FOUND)
}
```

그림 5 단일 해시 테이블 방식 URL 검색 알고리즘

사용한다고 가정한다. 검색하고자 하는 URL의 길이를 N이라 하고 HashFunction() 함수의 복잡도를 O(N), URL_Concatenation() 함수의 복잡도를 O(N)이라 가정한다. 본 알고리즘은 검색하고자 하는 URL의 세그먼트 개수만큼의 HashFunction() 호출과 URL_Concatenation() 호출이 발생한다. URL 세그먼트 개수의 최대값은 N 개이므로 본 알고리즘은 O(N*N)의 복잡도를 갖는다. 이 알고리즘은 검색하고자 하는 URL 문자열의 길이가 늘어나고 세그먼트의 개수가 늘어나면 성능이 저하되는 단점이 있다.

3. 제안한 해시 트리 구조의 URL 목록 검색

단일 해시 테이블 방식 알고리즘의 복잡도를 낮추는 방법으로 해시 트리 방식 검색 알고리즘을 제안한다. URL Prefix Form으로 구성된 URL 목록에 대해 각 세그먼트들을 노드로 하는 트리로 표현하는 것이 가능하다. 가상의 빈 노드를 루트 노드로 하고 각 URL Prefix의 세그먼트들을 자식 노드로 연결한다. 한 노드에는 여러 자식 노드가 생길 수 있다. 한 노드에 동일한 이름의 자식 노드가 생기는 것을 허용하지 않는다. 구성된 트리에서의 각 노드의 레벨은 URL Prefix 배열의 인덱스 값과 동일한 의미를 가진다. 그림 4의 URL 목록에 대해 이를 URL Prefix 트리 형태로 표현한 예를 그림 6에 보인다.

URL 검색 문제는 주어진 URL을 포함하는 URL Prefix가 이 트리에 존재하는지를 검색하는 것과 같다. 각 트리 노드에서 자식 노드의 개수는 여러 개가 될 수

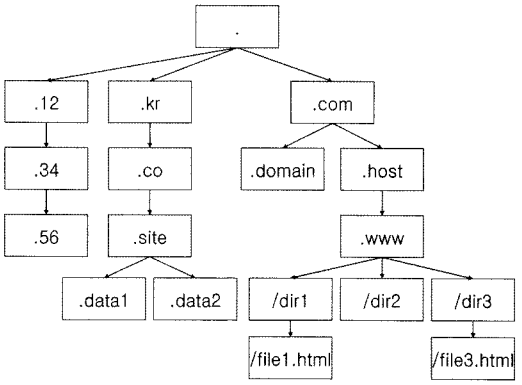

그림 6 트리로 표현된 URL 목록의 예

있고 그 개수가 일정하지 않다. 각 노드마다 해시 테이블을 자식 노드들의 해시 테이블을 만들어 해시 트리를 구성한다. 해시 트리로 표현된 URL 목록의 예를 그림 7에 보인다.

해시 테이블의 각 엔트리는 name 필드와 hashtable 필드를 갖는다. name 필드에는 세그먼트의 이름이 저장되고, hashtable 필드에는 서브트리가 존재할 경우 서브트리의 해시 테이블을 가리킨다. 원하는 세그먼트가 해시 테이블에 존재하지 않는 것은 검색하고자 하는 URL이 목록에 없음을 나타낸다. 세그먼트에 해당하는 서브트리가 존재하지 않는 것은 해당 prefix를 갖는 모든 URL이 목록에 들어 있음을 나타낸다. 세그먼트에 해당하는 서브트리가 존재하는 것은 해당 prefix까지의 비교로는 URL의 존재 여부를 알 수 없고, 다음 세그먼트를

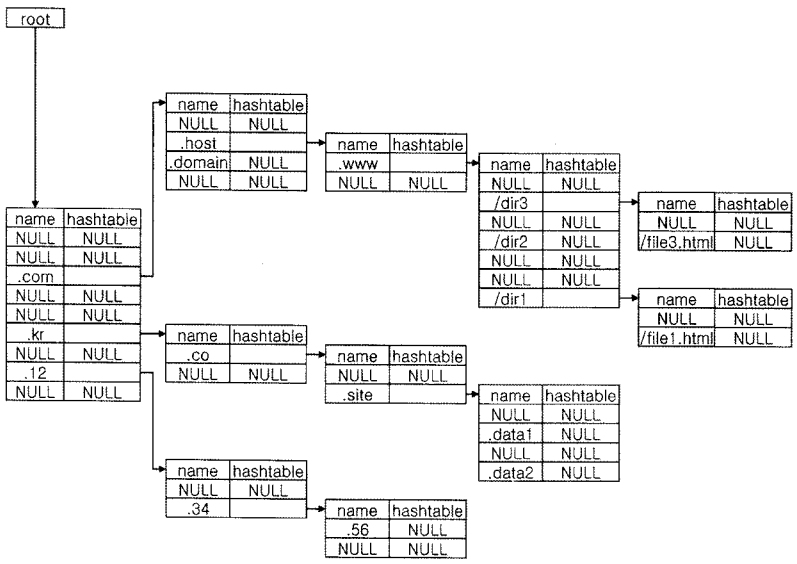

그림 7 해시 트리로 표현된 URL 목록의 예

서브트리에서 검색해야 한다는 것을 나타낸다. 해시 트리를 메모리에 구성하기 위해 해당 노드에 몇 개의 자식 노드가 있는지 미리 알아야 하므로, 전체 URL을 검색하여 각 노드마다의 자식 노드들의 개수를 파악하는 과정이 필요하다. URL 목록이 방대한 경우 이 과정에 시간이 오래 걸리고 메모리의 사용량도 크므로 별도의 프로그램으로 이 과정을 수행하여 해시 트리 파일을 생성한다. 생성된 해시 트리 파일에는 각 해시 트리 노드의 세그먼트 이름과 해시 테이블의 크기, 자식 노드의 유무를 기록한다. URL 목록 검색 프로그램에서는 이 해시 트리 파일을 읽어 들여 메모리에 해시 트리를 구성한다. 그림 8에 BNF 로 표시된 해시 트리 파일의 형식을 보인다.

파일은 루트 해시 테이블을 지정한다. 각 해시 테이블에는 number로 엔트리의 개수를 지정하고 number 개의 엔트리들이 들어간다. 각 엔트리에는 identifier로 엔트리의 이름이 지정되고 해시 테이블이 한 개 존재하거나 존재하지 않을 수 있다. 해시 트리 파일을 통하여 트리 생성과 트리 검색 과정을 분리하고 별도의 프로그램으로 동작하게 하는 것은 다음과 같은 특징이 있다. 트리 검색 프로그램은 해시 트리의 각 노드들의 크기를 해시 트리 파일을 통해 이미 알고 있으므로 최적의 크기를 가지는 해시 테이블을 생성할 수 있어 메모리 비용 효율적이다. 트리 검색 프로그램은 검색 이외에 삽입, 삭제 등 해시 트리에 변경이 일어나는 기능을 수행하기 위해서는 해당 해시 테이블을 재할당해야 할 필요성이 있어 비효율적이다. 해시 트리 생성 프로그램은 해시 트리에 지속적인 삽입을 수행하는 프로그램으로 삽입에 효율적인 구조로 별도로 제작한다. 목표로 하는 네트워 기반 블랙리스트 URL 목록 방식 시스템에서는 URL 목록 검색 속도가 가장 중요한 요소이고 URL 목록 변경의 중요도는 낮으므로 변경을 고려하여 해시 트리를 구성하는 것보다 검색에 최적화된 해시 트리를 구성하는 것이 중요하다. 구성된 해시 트리를 검색하는 알고리즘을 그림 9에 보인다. 검색할 입력 URL은 URL prefix form으로 주어진다. 해시 테이블에서 충돌해결

```
<table> ::= "H" number {<entry>} ".H"
<entry> ::= "E" identifier [<table>] ".E"
```

그림 8 해시 트리 파일의 형식

```
Algorithm SearchTree (U, Tree)
{
        U : An Array of given URL in URL Prefix form {U[0], U[1], U[2], U[3], ... }
        Tree : URL Hashtree

        // 더 이상 탐색할 세그먼트가 없는 경우 탐색 종료
        if (U[0] == NULL) return (NOT FOUND)
        HashIndex = HashFunction (U[0])
        item = Tree.Item [HashIndex]

        // 세그먼트가 테이블에 존재하지 않으면 탐색 종료
        if (item.name == NULL) return (NOT FOUND)
        while (item.name != U[0])
        {
                HashIndex = (HashIndex + 1) % Tree.HashTableSize
                item = Tree.Item [HashIndex]
                if (item.name == NULL) return (NOT FOUND)
        }

        // 해당 세그먼트의 서브트리가 존재하지 않으면 탐색 종료
        if (item.Tree == NULL) return (FOUND)
        else
        {       // 세그먼트의 서브트리 탐색
                U' = {U[1], U[2], U[3], U[4], ...}
                return (SearchTree (U', item.Tree))
        }
}
```

그림 9 해시 트리 검색 알고리즘

방법은 단일 해시 테이블 방식과 같은 open linear addressing을 사용한다. 해당 세그먼트에 서브 트리가 존재한다는 것은 해당 세그먼트까지의 비교로는 URL 목록에 존재하는지의 여부를 알 수 없고 다음 세그먼트를 비교해야 한다는 것을 의미한다. 해당 세그먼트에 서브 트리가 존재하지 않는 것은 검색할 URL을 포함하는 URL의 존재를 의미한다. 해시 테이블의 인덱스들에 충돌이 일어나면 while() 반복을 수행한다. 해시 테이블에 들어갈 엔트리들의 개수와 내용을 미리 알고 있으므로 충돌을 회피하도록 해시 테이블의 크기와 해시 함수를 선택하는 것이 가능하여 while() 최대 반복 회수를 지정한 상수개 이하로 유지하는 것이 가능하다. 따라서 본 알고리즘의 시간 복잡도는 해시 테이블의 엔트리 개수와 관계없다. 주어진 URL의 길이가 유한한 자연수 N이라 할 때, 최대 세그먼트의 개수는 N을 넘지 않고, 재귀적으로 호출되는 SearchTree()의 최대 호출 횟수는 N 보다 작음을 보장한다. 본 알고리즘은 HashFunction()의 복잡도에 의해 결정되며 HashFunction()의 복잡도가 O(N)이라고 하면, 각 세그먼트들의 합이 N을 넘지 않으므로 이 알고리즘의 전체적인 시간 복잡도는 O(N)이다. 해시 테이블에 들어 있는 세그먼트의 개수를 C, 세그먼트의 길이를 L이라 하자. 각 해시 테이블은 최소 C개를 엔트리를 저장할 공간을 필요로 한다. 각 엔트리에는 최대 L개의 문자가 저장될 수 있다. 따라서 제안된 알고리즘의 공간 복잡도는 O(C*L)이다.

4. 실험 결과

표 1에 두 가지 검색 방식을 비교한 결과를 표시하였다. 실험에 사용된 URL 목록은 도메인, 호스트, 디렉토리, URL의 조합으로 유해 사이트로 분류된 약 1000만개의 목록을 실험에 사용하였다. 탐색용 URL의 수집은 플랜티넷[16]의 인터넷 전용선을 일주일동안 모니터링하여 수집하였다. 실험에 사용된 컴퓨터는 AMD Opteron 1.8GHz CPU와 4GB RAM이 장착되어 있고 운영체제로 리눅스를 사용하였다. 제안된 알고리즘은 C로 구현되었다. URL 목록과 탐색용 URL들을 모두 메모리에 미리 로드하여 실험 도중 디스크를 읽거나 쓰는 일이 없도록 하였다.

메모리 소비량은 탐색용 URL을 메모리에 적재하는데 사용된 메모리를 제외하고 순수하게 URL 목록을 구성하는데 사용된 메모리 공간이다. 단일 해시 테이블 방식에서는 URL 목록 내의 모든 URL prefix 들에 대해 따로 메모리 할당을 받았으나, 제안된 방식은 공유된 세그먼트들에 대해서는 단 한번만 메모리 할당을 받아 공유가 가능하여 메모리 소비가 9% 줄어들었다. 제안된 해시 트리 방식은 단일 해시 테이블 방식에 비해 약 3.6배의 속도 향상을 보인다. 제안된 방식은 별도의 프로그램을 사용하여 해시 트리 파일을 생성한다. 표 2에 URL 목록으로부터 해시 트리 파일을 생성하는 과정에 사용된 시간과 메모리 사용량을 보인다.

표 1 검색 방식의 비교 결과

	단일 해시 테이블 방식	제안 방식	성능 비교
메모리 소비량 (MB)	264	242	9% 공간 절약
탐색 속도 (10^6 lookups/sec)	1.33	4.76	3.6배 속도 향상

표 2 URL 개수에 따른 해시 트리 파일 생성 프로그램의 수행 결과

URL 개수 (10^6개)	수행 시간 (초)	최대 메모리 소비량 (MB)
4	34	1864
6	42	2794
8	56	3725
10	114	4656

URL 목록의 개수를 증가시키면서 실험한 결과 테스트 환경의 메모리 크기인 4GB 보다 더 많은 메모리를 소비하는 1000만개 URL 목록의 경우 디스크 스와핑이 발생하여 수행 시간이 급격히 증가하였다. 해시 트리 파일을 생성하는 프로그램은 해시 트리에 지속적인 삽입을 수행하는 프로그램이다. 제안된 해시 트리는 지속적인 삽입을 효율적으로 수행하기에는 비효율적인 구조이다. 각 트리 노드의 해시 테이블의 크기를 정한 이후 삽입을 통해 이 테이블의 엔트리의 수가 증가하면 해시 키 값 충돌 확률이 증가하고 충돌 처리에 시간 비용이 든다. 엔트리의 개수가 해시 테이블의 크기와 같아지면 더 이상 삽입이 불가능하여 해시 테이블 재할당 절차가 필요하고 이것은 역시 시간과 공간 비용이 든다. 실험 결과에 제시된 생성 프로그램은 해시 테이블의 재할당 시 크기를 10배씩 증가하도록 설정한 상태에서 실험한 결과이다. 이 방식의 시간과 공간 비용 문제점은 다차원 해시 트리 방식[13]으로 해결할 수 있다. 탐색하려는 URL 에 세그먼트 수가 많은 경우 해시 트리 방식은 트리 탐색 중 세그먼트 노드가 트리에 없는 것을 발견한 즉시 탐색 종료가 가능하다. 이에 반해 단일 해시 테이블 방식은 참조하려는 URL의 세그먼트의 수만큼 테이블 참조를 수행해야 한다. 세그먼트의 수가 많은 경우 URL 문자열의 길이가 길어져 해시 함수의 수행 시간도 길어지는 단점이 있다. 이런 단점은 URL이 목록에 존재하지 않는 경우 더욱 두드러진다. 이의 비교를 위해 참조하고자 하는 URL 세그먼트의 수에 따라 속도의 차이를 검사하여 표 3에 제시하였고 그 참조 속도의 추이를 그림 10에 나타냈다. 앞의 실험과 동일한 조건에서

표 3 세그먼트 개수에 따른 참조 속도 비교

세그먼트 개수	단일 해시 테이블 방식 (10^6 lookups/sec)	제안 방식 (10^6 lookups/sec)	속도 향상 (%)
3	3.15	5.10	61.9
4	2.08	3.57	71.6
5	1.52	3.12	105.3
6	1.11	2.50	125.2
7	0.89	2.08	133.7
8	0.71	1.92	170.4
9	0.60	1.74	190.0
10	0.49	1.52	210.2

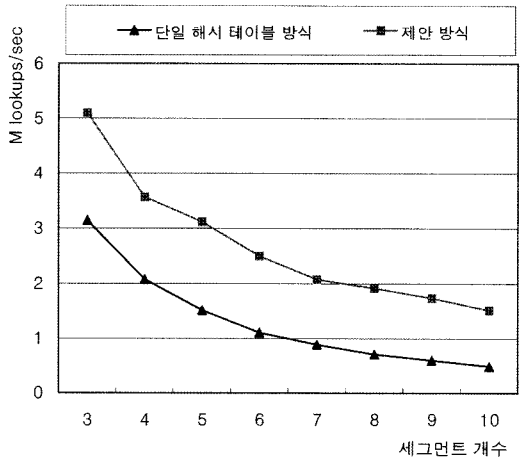

그림 10 세그먼트 개수에 따른 참조 속도 비교

실험하였다. URL 목록에 존재하지 않고, 주어진 세그먼트 개수와 같은 크기의 URL이고, 제안 방식으로 탐색할 경우 주어진 세그먼트 개수에서 트리 탐색이 종료되는 URL들을 선택하여 실험하였다.

실험 결과 사용자가 접근하려는 사이트의 구조가 복잡하여 세그먼트의 개수가 많아질 때 기존 방식과 제안 방식의 성능 차이는 더욱 크게 나타난다. 기존 방식은 주어진 URL의 세그먼트 개수만큼의 비교를 해야 하나, 제안된 방식은 세그먼트의 불일치가 발견되는 시점에서 탐색이 종료되어 전체적인 탐색 속도가 평균 3.6배 증가하였다.

5. 결론 및 추후 과제

본 논문에서는 네트워 기반 블랙리스트 URL 목록 방식 컨텐츠 필터링 시스템에 적합한 효율적인 자료 구조와 URL 목록 탐색 알고리즘을 제안하였다. 제안한 방식은 효율적인 콘텐츠 필터링을 위해 URL Prefix Form을 정의하였다. URL Prefix Form은 URL 집합을 표현하므로 여러 개별 URL들을 단 하나의 URL Prefix Form으로 나타낼 수 있어 URL 목록 내의 엔트리 개수를 줄일 수 있다. URL Prefix Form은 도메인, 호스트, 디렉토리, 개별 URL의 다양한 필터링 목록을 단 하나의 구조로 표현할 수 있다. 단일 해시 테이블 방식에서는 탐색하고자 하는 URL의 세그먼트들의 개수만큼의 해시 테이블 참조가 일어나 중복 탐색으로 인한 비효율성이 있었다. 제안한 방식에서는 URL Prefix Form의 특징을 이용하여 각 세그먼트를 노드로 하는 해시 트리를 구성하고, 이 트리에서 탐색을 수행한다. 단일 해시 테이블 방식과 달리 해시 트리 방식은 URL Prefix 세그먼트의 일부 비교만으로 URL 목록 내에 존재하는지의 여부를 알아낼 수 있어, 중복 탐색의 비효율성을 없앴다. 실험 결과 제안된 방식은 단일 해시 테이블 방식에 비해 평균 3.6배 빠른 탐색 속도를 보였고 메모리 소비량은 비슷하였다. 제안한 방식은 해시 트리의 구성을 위해 별도의 프로그램을 사용하였다. 이 프로그램은 URL 목록 파일을 읽어 해시 트리 파일을 생성한다. 이 과정은 URL 목록 1,000만개의 경우 114초의 수행시간과 4.7GB의 메모리를 요구했다. 해시 트리 생성 과정은 해시 트리 검색과 별도의 진행되므로, 시간과 공간의 제약을 받지 않는다. 해시 트리를 이용한 제안한 방식은 URL 목록 참조 속도가 중요한 네트웍 기반의 블랙리스트 방식 대용량 필터링 시스템에 적합함을 보였다. 필터링에 필요한 블랙리스트 URL 목록은 지속적으로 커지므로 이를 모두 메모리에 탑재하기 위해서는 시스템의 메모리가 부족할 수 있어 이의 개선을 위해 추후 메모리의 사용량을 줄일 수 있는 방안이 필요하다. 네트웍 회선의 용량도 지속적으로 증가하므로 URL 목록 참조 속도를 더욱 빠르게 향상시킬 필요가 있다. 해시 트리 생성 과정의 시간과 공간 비용을 줄일 수 있는 보다 효과적인 방안의 적용도 필요하다.

참 고 문 헌

[1] 정보통신윤리위원회, http://www.icec.or.kr.
[2] 한국전산원, "네트워크용 유해정보 차단도구 NCApatrol Proxy 1.0 개발 보고서", 한국전산원 연구 보고서 IV-PER-98035, 1998년 12월.
[3] Web Sense Inc., http://www.websense.com.
[4] Secure Computing Corporation, http://www.secure-computing.com.
[5] World Wide Web Consortium, "Platform for Internet Content Selection: PICS," http://www.w3.org/PICS/.
[6] M. Hammami, Y. Chahir, and L. Chen, "Web-Guard: Web Based Adult Content Detection and Filtering System," in Proc. IEEE/WIC Int. Conf. on Web Intelligence, pp. 574-578, Oct. 2003.
[7] C. Ding, C. Chi, J. Deng, and C. Dong, "Centralized Content-based Web Filtering and Blocking: How

Far Can It Go?," in Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics, vol.2, pp. 115-119, Oct. 1999.
[8] R. Du, R. Safavi-Naini, and W. Susilo, "Web Filtering Using Text Classification," in Proc. IEEE Int. Conf. on Networks, pp. 325-330, Oct. 2003.
[9] N. Huang, R. Liu, C. Chen, Y. Chen, and L. Huang, "Fast URL Lookup Engine for Content-Aware Multi-Gigabit Switches," in Proc. Int. Conf. on Advanced Information Networking and Applications, vol.1, pp. 641-646, Mar. 2005.
[10] Basso et al., "Method and System for Performing a Pattern Match Search for Text Strings," US Patent No. US 7054855 B2, 2006.
[11] H. Yan, J. Wang, X. Li, and L. Guo, "Architectural Design and Evaluation of an Efficient Web-crawling System," in Proc. Int. Symp. on Parallel and Distributed Processing, pp. 1824-1831, Apr. 2001.
[12] B. Michel, K. Nikoloudakis, P. Reiher, and L. Zhang, "URL Forwarding and Compression in Adaptive Web Caching," in Proc. IEEE. INFOCOM, vol.2, pp. 670-678, Mar. 2000.
[13] P. Gupta and N. McKeown, "Algorithms for packet classification," IEEE Network, vol. 15, no. 2, pp. 24-32, March 2001.
[14] Erik Burckart and Aravind Srinivasan, "Multidimensional hashed tree based URL matching engine using progressive hashing," US Patent Publication Number 2005-0055437 A1, 2005.
[15] Google Inc., http://www.google.com.
[16] 플랜티넷, http://www.plantynet.com.

박 창 욱

1993년 2월 서강대학교 전자공학과 학사 졸업. 1995년 2월 서강대학교 전자공학과 석사 졸업. 1997년 2월 서강대학교 전자공학과 박사 수료. 1996년~2000년 (주) 서두로직 선임 연구원. 2001년~현재 (주) 플랜티넷 연구소장. 주관심분야는 콘텐츠 필터링, 콘텐츠 딜리버리 등임

황 선 영

1976년 2월 서울대학교 전자공학과 졸업. 1978년 2월 한국과학원 전기 및 전자공학과 공학석사 취득. 1986년 10월 미국 Stanford 대학 전자공학 박사 학위 취득. 1976년~1981년 삼성 반도체 주식회사 연구원, 팀장. 1986년~1989년 Stanford 대학 Center for Integrated Systems 연구소 책임연구원. Fairchild Semiconductor Palo Alto Research Center 기술자문. 1989년~1992년 삼성전자(주) 반도체 기술자문. 2002년 4월~2004년 2월 서강대학교 정보통신대학원장. 1989년 3월~현재 서강대학교 전자공학과 교수. 주관심분야는 컴퓨터 시스템, Embedded 시스템 설계, SoC와 framework 설계 등임