정규 표현식을 이용한 패턴 매칭 엔진 개발

Development of the Pattern Matching Engine using Regular Expression

고광만, 박홍진
상지대학교 컴퓨터정보공학부

Kwang-Man Ko(kkman@sangji.ac.kr), Hong-Jin Park(hjpark1@sangji.ac.kr)

요약

스트링 패턴 매칭 알고리즘은 특정 검색어, 키워드를 검색하는 속도에서는 우수성이 다양한 방법으로 입증되었지만 다양한 패턴에 대해서는 기존의 알고리즘으로는 한계를 가지고 있다. 본 논문에서는 정규 표현식을 이용하여 특정 키워드를 포함하여 다양한 패턴의 검색어에 대해서도 효율적인 패턴 매칭을 수행하여 패턴 검색의 효율을 높이고자 한다. 이러한 연구는 기존의 단순한 키워드 매칭에 비해 각종 유해한 스트링 패턴을 효과적으로 검색할 수 있으며 스트링 패턴 매칭 속도에서도 기존의 알고리즘에 비해 우수성을 갖는다. 본 연구에서 제안한 LEX로부터 생성된 스트링 검색 엔진은 패턴 검색 속도에 대한 실험에서 패턴의 수가 1000개 이상인 경우에서는 BM&AC 알고리즘보다 효율적이지만 키워드 검색에서는 유사한 결과를 얻었다.

■ 중심어 : | 스트링 패턴 매칭 | 정규 표현 | LEX | BM & AC 알고리즘 |

Abstract

In various manners, string pattern matching algorithm has been proven for prominence in speed of searching particular queries and keywords. Whereas, the existing algorithms are limited in terms of various pattern. In this paper, regular expression has been utilized to improve efficiency of pattern matching through efficient execution towards various pattern of queries including particular keywords. Such as this research would enable to search various harmful string pattern more efficiently, rather than matching simple keywords, which also implies excellent speed of string pattern matching compared to that of those existing algorism. In this research, the proposed string search engine generated from the LEX are more efficient than BM & AC algorithm for a string patterns search speed in cases of 1000 with more than patterns, but we have got similar results for the keywords pattern matching.

■ keyword : | String Pattern Matching | Regular Expression | LEX | BM & AC Algorithm |

I. 서 론

최근 인터넷의 보급이 급속도로 확산되면서 인터넷 환경에 유해하는 각종 스팸 메일에 대한 피해가 급증하고 있다. 또한 정상적인 서버에 대해 비정상적인 접근 및 불순한 목적을 가지고 접근하여 발생되는 피해도 그

* 본 논문은 2005년도 상지대학교 교내 연구비 지원에 의한 것입니다.

접수번호 : #080129-002
접수일자 : 2008년 01월 29일
심사완료일 : 2008년 02월 15일
교신저자 : 고광만, e-mail : kkman@sangji.ac.kr

규모를 파악하기 힘들 정도로 심각하다. 따라서 정상적인 서버 등에 불순한 목적으로 접근하는 사용자와 각종 스팸성 명령어 및 용어를 차단하기 위한 각종 연구 및 상용화 서비스가 오래전부터 진행되고 있다.

본 논문에서는 스팸성 명령어 및 용어, 음란성 용어 등이 서버에 접근할 수 없도록 입력되는 명령어에 대한 검색을 강화하기 위해 정규 표현식을 이용한 패턴 매칭 알고리즘을 제시하고 기존 방법과 비교하여 성능 우수성을 검증하고자 한다. 기존에 국내외적으로 수많은 스트링 패턴 매칭 알고리즘이 설계되고 구현되어 현재 사용되고 있지만 알고리즘의 적용 분야에 따라 패턴 매칭의 속도가 차이가 있으며 사용자에 대해 양질의 서비스를 제공하기 위해서는 스트링 패턴 매칭의 속도가 중요한 요소가 된다. 기존의 스트링 패턴 매칭 알고리즘은 특정 검색어, 키워드를 검색하는 속도에서는 우수성이 다양한 방법으로 입증되었지만 다양한 패턴에 대해서는 기존의 알고리즘으로는 한계를 가지고 있다. 본 논문에서는 정규 표현식(regular expression)을 이용하여 특정 키워드를 포함하여 다양한 패턴의 검색어에 대해서도 효율적인 패턴 매칭을 수행하여 패턴 검색의 효율을 높이고자 한다. 본 연구를 진행하기 위해 컴파일러 자동화 도구인 LEX를 기반으로 하며 LEX가 갖는 속도의 문제점을 개선하기 위한 기법을 제안하고 이를 구현하였다. 구현된 결과는 기존 스트링 패턴 매칭 알고리즘과 속도 및 검색율에 대한 비교 분석을 진행하였으며 입증된 성능 결과를 활용하여 실제 상용화 검색 엔진에 적용하였다.

본 논문의 구성은 제 2장에서 본 연구의 기반이 되는 스트링 패턴 매칭 알고리즘과 컴파일러 자동화 도구인 LEX에 대해 고찰한다. 제 3장에서는 본 연구에서 개발하고자는 하는 정규 표현식을 이용한 검색 엔진 모델을 설계한 후 그개발 결과와 성능 분석 결과를 제시한다. 제 4장에서는 본 연구의 최종적인 결론과 향후 연구 방향에 대해 기술한다.

II. 관련 연구

2.1 스트링 패턴 매칭 알고리즘

일반적인 싱글 스트링 매칭 알고리즘중에서 가장 효율적이라고 인식되고 있는 알고리즘으로 텍스트 에디터에서 '찾기' 혹은 '바꾸기'의 기능 을 위해 사용되고 있다. 전처리 과정에서 O(m + δ)의 공간과 시간 복잡도를 가지며 검색과정에서 O(m) 시간 복잡도를 가진다. Boyer-Moore 알고리즘의 패턴 매칭 방법은 "Right-To-Left Scan", "Bad Character Rule", "Good Suffix Rule"의 세 가지 아이디어로 이루어져 있으며 Brute Force 알고리즘, Knuth-Morris-Pratt 알고리즘은 패턴과 텍스트를 비교할 때 패턴의 이동 방향이나 캐릭터 비교 방향이 모두 왼쪽에서 오른쪽으로 진행되는 반면, Boyer-Moore 알고리즘은 패턴의 이동 방향은 왼쪽에서 오른쪽으로 동일하지만 캐릭터 비교 방향이 오른쪽에서 왼쪽으로 진행된다. 이러한 방법은 알고리즘의 수행속도에 영향을 미치지 못하지만, 나머지 두 아이디어와 합쳐져 빠른 수행 속도를 낼 수 있게 된다[1].

Aho-Corasick 알고리즘은 키워드의 finite set(K)으로 구성된 유한 상태 패턴 매칭 엔진을 이용한 알고리즘으로 Knuth-Morris-Pratt 알고리즘을 개선했다. 유한 상태 패턴 매칭 엔진을 구성한 후, 입력으로 text string(x)을 받아 x에서 키워드들의 집합인 K에 속한 임의의 키워드가 나타나는 위치를 출력한다. 패턴 매칭 엔진은 goto function g, failure function f, output function으로 이루어진다[2].

AC_BM 알고리즘은 스트링의 앞부분에 중복되는 경우가 많은 경우 여러 개의 패턴을 검색할 때 그 속도를 높여보자는 것이다. 이것은 Boyer-Moore의 알고리즘을 거꾸로 해서 여래개의 패턴에 한꺼번에 적용한다. 즉, 패턴 내에서의 캐릭터 비교는 왼쪽에서 오른쪽으로 하고 패턴을 왼쪽으로 이동하는 것이다. Boyer-Moore의 Bad Character Shift Rule과 Good Suffix Shift Rule의 개념도 그대로 도입했으며 단지 이동 방향이 반대인 것이 특징이다[3].

2.2 LEX와 정규 표현식

정규 표현식은 정규 언어를 표현하기 위한 한 방법으로 정규 언어에 속해있는 스트링의 모양을 직접 기술하는 형태를 갖는다. 정규 표현은 기본 소자와 연산자로

구성되어 있으며 기본 소자는 정규 표현을 이루고 있는 Θ, ε과 터미널 심볼로 구성되어 있다. 일반적으로 언어에 대한 인식기는 입력으로 스트링을 받아 스트링이 그 언어의 문장이면 "yes"를 답하고 그렇지 않으면 "no"를 답하는 기능을 수행한다. 이와 같은 인식기 중에 가장 간단한 형태가 유한 오토마타이며 어휘 분석기를 고안하고 구현하는 방법에 사용된다. 문법이 4개의 요소(4-tuple)로 정의되는데 비해 FA는 다음과 같은 5가지 요소로 정의된다.

FA $M = (Q, \Sigma, \delta, q_0, F)$

Q : 상태(state) 들의 유한 집합

Σ : 입력 심볼의 유한 집합

δ : 사상 함수(mapping function)

$- Q \times \Sigma \rightarrow 2^Q$

q_0 : 시작 상태 ($q_0 \in Q$)

F : 종결 상태의 집합 ($F \subseteq Q$)

어휘 분석기 생성기의 대표적인 예로 LEX를 들 수 있다. LEX는 1975년에 Lesk와 Schmit에 의해 발표된 어휘 분석기 생성기로 정규 표현들과 정규 표현이 매칭되었을 때 처리를 나타내는 수행 코드로 구성된 입력을 받아 테이블과 그 테이블을 이용하여 처리하는 프로그램(driver routine)을 출력한다[5][6].

LEX의 입력 화일은 메타 언어인 정규 표현으로 기술되며 확장 이름으로 *.l을 갖는다. LEX는 입력 화일 *.l을 C 언어로 작성된 lex.yy.c 화일로 출력한다. 이 화일을 컴파일하여 실행시키면 필요한 스캐너가 생성되는데 컴파일시에는 LEX의 라이브러리(-ll)를 포함해야 한다. LEX의 입력은 다음과 같이 세 부분으로 구성되어 있다.

<정의 부분>
%%
<규칙 부분>
%%
<사용자 부프로그램 부분>

<정의 부분>은 %{와 %} 사이에는 실행 코드를 C 언어로 기술할 때 필요한 자료 구조, 변수, 상수를 정의할 수 있는 부분이며 특정한 정규 표현을 하나의 이름으로 정의하여 그 형태의 정규 표현이 필요할 때마다 쓸 수 있도록 해주는 부분이다.

<규칙 부분>은 LEX 입력의 토큰의 형태를 표현하는 정규 표현과 그 토큰이 인식되었을 때 처리할 행위를 기술하기 위한 부분인 실행 코드로 구성된다.

<사용자 부프로그램 부분>은 LEX의 입력 작성시 사용되는 부프로그램들을 정의하기 위한 곳으로 LEX의 출력인 lex.yy.c에 복사된다.

LEX의 입력에서 스트링을 표현하기 위해 사용되는 메타 심볼과 그 의미는 다음과 같다. LEX에서 정규 표현을 나타내기 위하 텍스트 문자와 연산자로 구성된 표현을 이용한다. 텍스트 문자는 입력 스트림에서 실제로 매칭되는 부분에 해당되며 연산자 문자는 반복 또는 선택 등을 나타내는 특수 문자이다[5][6].

III. 정규 표현식을 이용한 스트링 패턴 매칭

3.1 시스템 모델

본 논문에서 설계하고 개발하고자 하는 정규 표현식을 이용한 패턴 매칭 엔진은 첫째, 다수의 정규 표현식에 대해 LEX의 입력 및 처리를 위한 모듈 작성하여 정규 표현식에 매칭되는 패턴을 최적의 시간에 탐색하여 출력 정보를 생성하고자 한다. 둘째, 특정 키워드 검색을 위한 패턴 매칭이 최적의 시간내에 가능하도록 한다. 마지막으로 위 두 가지 패턴 매칭을 기법을 하나의 엔진에 통합하여 패턴 매칭 시간을 최적화하고 탐색 효율을 높이고자 한다.

본 연구에서 설계하고 개발하고자 하는 정규 표현식을 이용한 패턴 매칭 엔진의 전체 시스템 모델은 [그림 1]과 같다. 하이브리드 오토마타 생성기는 서버에 접근하고자 하는 불특정 텍스트 문서에 포함될 수 있는 스팸성 명령어, 음란성 용어 등을 검출할 수 있는 하이브리드 오토마타를 생성한다. 특정 키워드에 대해서는 기존의 스트링 패턴 매칭 기법을 사용하고 있지만 본 논

문에서 구현한 하이브리드 오토마타 생성기는 특정 키워드와 동시에 다양한 스트링 패턴을 검출할 수 있도록 스트링 패턴을 정규표현으로 정의하였다. 하이브리드 오토마타는 서버에 접근하고자 하는 명령어 스트림인 텍스트 문서에 대해 실질적인 음해성 및 스펨성 스트링을 검출하는 기능을 수행한다. 스트링 패턴 매칭 수행시에 검출된 음해성 및 스펨성 스트링은 별도의 자료구조에 저장되어 반복적인 매칭 시간을 감소시킨다.

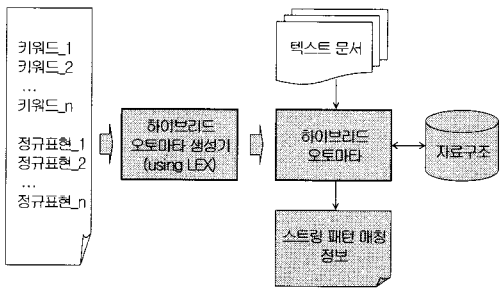


그림 1. 스트링 패턴 매칭 엔진 모델

실질적으로 하이브리드 오토마타는 입력되는 명령어 스트림인 텍스트 문서에 대해 키워드는 길이가 최소 64바이트부터 최대 512바이트 길이이며 누적 개수가 최대 30,000여개에 이르는 방대한 크기에 대해 스트링 패턴 매칭을 수행한다. 또한 텍스트 문서에 대한 스트링 패턴에 대한 검색도 패턴의 개수가 기존에 정의된 스트링 패턴에 대해 점진적으로 추가되어 2,000여개 이상에 대해 검색을 진행한다. 보안 수준 관리를 위해서는 선의의 키워드 및 스트링 패턴에 대한 접근을 허용하기 위해 입력되는 텍스트 문서의 길이, 보안 등급 등을 고려하여 상위 수준의 관리자에 의해 조정된다.

논문에서 이용한 LEX를 이용한 스트링 패턴 매칭 검색 엔진은 패턴 검색의 정확도가 이미 검증되어 사용되고 있으며 본 논문의 실험에서도 만족할만한 결과를 확인하였다. 또한

3.2 LEX 입력 및 출력

검색 엔진에서 검출하고자 하는 키워드와 스트링에 대한 패턴을 LEX 입력 형식을 참고하여 [그림 2]와 같이 기술하였다.

```
%%
\.eml       {print(1); REJECT;          return(1074914916);}
\.emf       {print(2); REJECT;          return(1074914916);}
\.wmf       {print(3); REJECT;          return(1074914916);}
"new XMLHttpRequest("
            {print(4); REJECT;          return(1074914916);}
\/readme\.eml
            {print(5); REJECT;          return(1074914916);}
window\.open\(\"readme\.eml\"
            {print(6); REJECT;          return(1074914916);}
document\.domain\(
            {print(7); REJECT;          return(1074914916);}
javascript\:
            {print(8); REJECT;          return(1074914916);}
Content-Type\:
            {print(9); REJECT;          return(1074914916);}
file"://"
            {print(10); REJECT;         return(1074914916);}
http"://"
            {print(11); REJECT;         return(1074914916);}
...
file"://"[^\n]{0,400}\/smi
            {print(22); REJECT;         return(1074914916);}
http"://".{0,400}\/smi
            {print(23); REJECT;         return(1074914916);}
// 스트링 패턴
CF_SETDATASOURCEUSERNAME"()"
    {print(20); REJECT;                 return(1074914916);}
Content-Type\:.*application\/smi.*?<area[
                \n\r]+href\=\'file:javascript\:\/smi
    {print(21); REJECT;                 return(1074914916);}
file"://"[^\n]{0,400}\/smi
    {print(22); REJECT;                 return(1074914916);}
http"://".{0,400}\/smi
    {print(23); REJECT;                 return(1074914916);}
Location\:.*URL.*\:\/smi
    {print(24); REJECT;                 return(1074914916);}
Content\-Disposition\:.*[0-9]{8}(\-[0-9]{4}){3}\-[0-9]{12}\/smi
    {print(25); REJECT;  return(1074914916);}
        return(-1);
%%
```

그림 2. LEX 입력 소스(source.l)

[그림 2]에서 작성된 LEX 입력은 [그림 3]과 같은 실제 패턴 검색 프로그램을 출력한다.

```
int main() {
        // ...
```

```
        gettimeofday(&s, NULL);
        tn = yylex();
        gettimeofday(&e, NULL);
        print_duration("Lex Search", s, e);
        while((tn = yylex()) != 0) {
                switch (tn) {
                }
                offset += yyleng;
        }
}
```

그림 3. LEX 출력

실질적인 패턴 매칭의 수행은 yylex() 함수에 의해 진행된다. yylex()에 의해 수행되는 매칭 동작은 일반적인 LEX의 수행 방법과 동일하다. 또한 실질적인 매칭 시간을 측정하기 위해 [그림 4]와 같은 함수를 사용하였다.

```
void print_duration(char* title, struct timeval s,
                    struct timeval e) {
    if (s.tv_usec > e.tv_usec) {
        e.tv_usec += 1000000;
        e.tv_sec --;
    }
    printf("%s duration: %d sec %d usec\n", title,
        e.tv_sec - s.tv_sec, e.tv_usec - s.tv_usec);
    return;
}
```

그림 4. 매칭 시간 측정 함수

LEX에 의해 자동 생성된 lex.yy.c 파일은 패턴 매칭을 수행할 수 있는 소스 파일의 정보를 C 언어 구문 형식으로 유지하고 있다. 따라서 실질적인 성능 분석을 위해서 생성된 lex.yy.c 파일의 yylex() 함수에 의해 실질적으로 수행된다. 키워드 및 정규 표현에 대해 스트링 패턴 매칭을 수행하는 yylex() 함수를 호출한 후 실제로 yylex() 함수에 의한 스트링 패턴이 시작되는 시점과 yylex() 함수가 종료되기 직전에 gettimeofday() 함수를 이용하여 패턴 매칭 시작 시간과 패턴 매칭을 완료한 시간을 측정하였으며 패턴 매칭 시간 측정시에 매턴 매칭이 수행된 결과를 출력하는 시간을 제외하여 패턴 매칭 시간으로 간주하였다.

3.3 성능 측정 결과 및 분석

키워드에 대한 스트링 매칭 시간의 성능을 비교 측정하기 위해 BM 알고리즘(three variations of the Boyer-Moore Set matching algorithm), AC 알고리즘(Aho-Corasick State Machine - version 2.0), LEX(ver 2.9)를 이용한 하이브리드 오토마타를 Linux(Redhat 9.0)/IBM 환경에서 구현하여 [그림 5]와 같은 결과를 얻었다. 또한 시간 측정을 위해서 gettimeofday() 함수를 이용하였다. 키워드 개수(10, 50, 100, ...,500, 1000, 2000, 3000)를 변화시키며 BM 알고리즘, AC 알고리즘, LEX를 이용한 방법 적용하여 속도 측정(단위, usec)을 진행하였다.

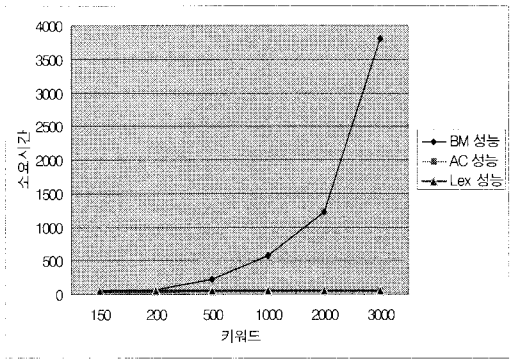


그림 5. 키워드에 대한 매칭 속도 측정

[그림 5]의 키워드에 대한 매칭 속도 측정에서는 기본적으로 64 바이트의 문자열 길이에 대해 실험을 하였다. BM 알고리즘의 경우에는 비교할 키워드의 길이에 민감하게 반응하므로 키워드의 길이가 길어지면 수행시간이 감소되는 효과를 얻을 수 있었다. BM 알고리즘과 AC 알고리즘이 비슷한 속도로 수행되는 시점은 키워드의 개수가 100개 이상일 때이며 이후부터는 큰 편차가 보이기 시작하였다. LEX를 이용하여 키워드에 대한 스트링 패턴 매칭을 적용한 경우는 키워드의 개수가 적은 경우에는 BM 알고리즘, AC 알고리즘에 비해 느린 속도로 패턴 매칭 시간 결과를 얻지만 키워드의 개수 증가하는 경우에도 전체적인 패턴 매칭 속도에는 커

다란 변화가 없는 것을 확인할 수 있다. 즉, LEX를 이용하여 정규 표현식을 적용하여 키워드에 대한 매칭을 진행할 경우에는 비교하고자 하는 키워드에 길이에 영향을 받고 키워드의 개수가 적은 경우에는 타 알고리즘과 비교하여 검색 속도에 대한 부담이 발생하므로 키워드의 개수가 1000개 이상에서 성능의 우위를 보일 수 있음을 확인할 수 있다. 또한 타 알고리즘에 비해 다중 키워드 매칭일 가능하므로 이를 적절하게 적용할 수 있는 분야에 적합한 것으로 판단된다. 실질적으로 산업 현장에서는 일반적으로 키워드의 개수가 최소 1000개 이상을 사용하는 경우가 많고 보안 수준을 높이기 위해 스펨성 및 음란성 용어는 누적되는 수가 점진적으로 증가되므로본 논문에서 제시한 LEX를 이용한 방법이 보다 효과적으로 사용될 수 있음을 보인다. 또한 단일 키워드 검색 알고리즘인 BM의 경우 하나의 문자열에 대해서 검사해야 할 키워드의 개수만큼 검사가 반복되어야 하기 때문에 n이 하나의 키워드를 검출해야 하는데 소요되는 시간이고, m이 키워드의 개수이면 복잡도는 O(nm)이 되므로 키워드의 개수가 증가하면 성능이 크게 저하된다.

스트링 패턴에 대한 스트링 매칭 시간의 성능을 비교 측정하기 위해 BM 알고리즘, AC 알고리즘, Linux(Redhat 9.0)/IBM 환경에서 구현하여 [그림 6]과 같은 결과를 얻었다. [그림 6]의 비교 결과를 통해 2014개의 패턴 입력에 대해 19개를 검출하는데 시간 차이가 20배 이상의 차이를 발견하였다. 또한 검색하고자 하는 패턴 검색의 정확도에서 기존에 검증된 것과 같이 정확성을 확인할 수 있었다. 따라서 본 논문에서 스트링 패턴에 대한 검색 결과를 측정하기 위해 BM 알고리즘을 제외하고 AC 알고리즘과 LEX를 이용한 하이브리드 오토마타의 결과를 중점적으로 분석하였다.

```
start BMSET search
pattern count: 2014, packet count:353, match count:19
BMSET search duration: 0 sec 8184 usec

start AC search
pattern count: 2014, packet count:353, match count:19
AC search duration: 0 sec 404 usec
```

그림 6. BM과 AC 알고리즘 스트링 패턴 매칭 시간 비교

우선적으로 AC 알고리즘을 구현하여 패턴의 개수와 길이를 고려하여 [표 1]과 같은 결과를 얻었다.

표 1. AC 알고리즘 패턴 검색 속도

(단위 :usec)

패턴개수	64 Byte	128 Byte	256 Byte	512 Byte
10	4:5	2:3	2:4	3:4
50	8:13	3:6	4:6	5:6
100	13:21	3:13	5:12	7:10
500	19:31	4:23	8:17	11:15
1000	33:59	11:35	17:31	19:30
1800	57:88	16:41	21:39	24:36

스트링 패턴에 대한 매칭에서는 우선 AC 알고리즘을 적용하여 비교적 패턴의 개수가 적은 경우(10, 50, ...)와 패턴의 개수가 많은 경우(1000이상)에 대해 측정하였다. 스트링 패턴에 대한 매칭 속도 측정시에는 비교해야할 문자열의 길이가 상당히 민감한 요소가 되므로 크기를 64 바이트, 128 바이트, 255 바이트, 512 바이트 단위로 측정하였다. 동일한 패턴에 대해 정규 표현식을 작성하여 LEX를 이용한 측정한 패턴 검색 속도는 [표 2]와 같다.

표 2. 하이브리드 오토마를 이용한 패턴 검색 속도

(단위 :usec)

패턴개수	64 Byte	128 Byte	256 Byte	512 Byte
10	73	88	132	182
50	77	93	138	193
100	79	94	138	195
500	79	96	144	204
1000	81	83	149	255
1800	82	85	152	287

하이브리드 오토마타를 이용하여 패턴의 개수가 적은 경우에는 매칭 속도가 AC 알고리즘에 비해 현저하게 속도가 증가되는 것을 확인할 수 있다. 즉, 단일 패턴에 대한 검색에서는 기존 알고리즘이 성능이 우수함을 확인할 수 있다. 하지만 동일 패턴에 대해 다중 패턴 매칭을 수행하는 경우에는 패턴의 검색 속도가 기존 알고리즘에 근접하여 패턴의 개수가 증가하는 경우에도 검색 속도 완만하게 상승하는 결과를 얻었다.

따라서 본 연구를 통해 키워드 및 스트링 패턴에 대한 검색을 진행할 경우 키워드 및 패턴의 길이를 고려하여 스트링 패턴 매칭 알고리즘 선정이 요구되며 검색하고자 하는 키워드 및 패턴의 개수가 1000개 이상이 되는 경우에는 정규 표현식을 이용하여 본 연구에서 제안한 방법을 적용하는 것이 성능 우수성이 증가됨을 확인할 수 있다. 하지만 패턴의 길이가 작거나 키워드의 개수 100개 이하인 경우에는 기존의 AC 스트링 패턴 매칭 알고리즘을 적용하는 것이 성능 우수성을 보장한다. 마지막으로 키워드 및 패턴에 대해 다중 검색을 지원하고자 하는 경우에는 키워드 및 패턴의 길이와 개수에 상관없이 LEX를 이용한 하이브리드 오토마타를 이용하는 것이 적절한 선택으로 판단된다.

Ⅳ. 결론 및 향후 연구 방향

스팸성 명령어 및 용어, 음란성 용어 등이 서버에 접근할 수 없도록 입력되는 명령어에 대한 검색을 강화하기 위해 정규 표현식을 이용한 패턴 매칭 알고리즘을 제시하고 기존 방법과 비교하여 성능 우수성을 검증하고자 하였다. 기존의 수많은 스트링 패턴 매칭 알고리즘은 적용 분야에 따라 패턴 매칭의 속도가 차이가 있으며 사용자에 대해 양질의 서비스를 제공하기 위해서는 스트링 패턴 매칭의 속도가 중요한 요소가 된다. 본 논문에서는 정규 표현식을 이용하여 특정 키워드를 포함하여 다양한 패턴의 검색어에 대해서 다중 패턴 매칭이 가능하도록 LEX를 이용하여 하이브리드 오토마타를 생성한 후 성능 측정 결과를 제시하였다.

향후에는 보다 하이브리드 오토마타에 적합한 개선된 스트링 매칭 방법을 적용하여 키워드 및 패턴의 개수가 작은 경우에도 패턴 매칭 속도가 개선된 방법을 제시하고자 한다.

참 고 문 헌

[1] S. B. Robert and J. M. Strother, "A Fast String Searching Algorithm," CACM, Vol.20. No.10, 1977.

[2] A. Aho and M. Corasick, "Efficient string matching: an aid to bibliographic search," CACM, Vol.18, 1975.

[3] D. Gusfield, "Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology," University of California Press, CA, 1997.

[4] C. C. Jason, Stuart Stanford, Joseph McAlerney, "Towards Faster String Matching for Intrusion Detection or Exceeding the Speed of Snort," CACM, 1992.

[5] T. S. Axel, "Using C with CURSES, LEX and YACC," Prentice Hall, 1990.

[6] J. P. Tremblay and P. G. Sorenson, "The Theory and Practice of Compiler Writing," McGraw-Hill, 1993.

[7] H. Ellis and S. Sartaj, "Fundmentals of Computer Algorithms," Computer and Science Press, 1978.

[8] K. Ilgun, R. A. Kemmerer, and P. A. Porras, "State Transition Analysis: A Rule-Based Intrusion Detection System," IEEE Transactions on Software Engineering, Vol.21, No.3, 1995.

[9] R. Sekar and P. Uppuluri, "Synthesizing Fast Intrusion Detection/Prevention Systems from High-Level Specifications," In Proceedings of the USENIX Security Symposium, 1999.

[10] L. Kyle, "Mastering Algorithms with C," O'Reilly, 1999.

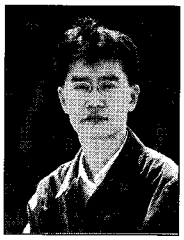
[11] F. Mike and V. George, "An Anylysis of Fast String Matching Applied to Content-Based Forwarding and Intrusion Detection," IEEE INFOCOM, 2002.

[12] G. Stephen, "String Searching Algorithms," World Scientific, 1994.

[13] http://www-igm.uiv-mlv.fr/~lecroq/string

저 자 소 개

고 광 만(Kwang-Man Ko) 정회원



- 1991년 2월 : 원광대학교 컴퓨터공학과(공학사)
- 1993년 2월 : 동국대학교 컴퓨터공학과(공학석사)
- 1998년 2월 : 동국대학교 컴퓨터공학과(공학박사)
- 2001년 8월 : 광주여자대학교 컴퓨터과학과
- 2002년 ~ 2003년 : Queensland University of Technology 연구교수
- 2001년 9월 ~ 현재 : 상지대학교 컴퓨터정보공학부 부교수

<관심분야> : 프로그래밍언어론 및 컴파일러

박 홍 진(Hong-Jin Park) 정회원

- 1993년 2월 : 원광대학교 컴퓨터공학과(공학사)
- 1995년 8월 : 중앙대학교 컴퓨터공학과(공학석사)
- 2001년 8월 : 중앙대학교 컴퓨터공학과(공학박사)
- 2001년 9월 ~ 현재 : 상지대학교 컴퓨터정보공학부 부교수

<관심분야> : 분산시스템 및 운영체제