고성능 웹크롤러의 설계 및 구현*

김희철* · 채수환**
*대구대학교 정보통신공학부 · **항공대학교 전자정보통신컴퓨터공학부

요 약

웹크롤러는 인터넷 검색엔진을 포함한 다양한 웹 응용프로그램에 활용되는 중요한 인터넷 소프트웨어 기술이다. 인터넷의 급격한 성장에 따라 고성능 웹크롤러의 구현이 시급히 요구되고 있다. 이를 위해서는 웹크롤러에 대한 성능확장성에 초점을 둔 연구가 수행되어야 한다. 본 논문에서는 병렬 프로세스 기반 웹크롤러(Crawler)의 성능향상에 필수적인 동적 스케줄링의 구현 기법을 제안한다. 웹크롤러는 웹문서의 수집 성능요구를 만족시키기 위하여 일반적으로 다중 프로세스 기반으로 설계되고 있다. 이러한 다중 프로세스 기반의 설계에서 프로세스 별로 문서수집 대상을 적정하게 선택하여 할당하는 크롤 스케줄링(Crawl Scheduling)은 시스템의 성능향상에 매우 중요한 요소이다. 본 논문에서는 먼저 크롤 스케줄링에 있어 중요한 문제점들에 대한 연구 결과를 제시한 후 공유메모리 기반 동적 스케줄링 지원 기법을 고안, 이를 구현하는 웹 크롤러 시스템 구조(Architecture)를 제안한다. 본 논문에서는 동적 스케줄링 지원 기능을 갖는 웹크롤러의 설계 및 구현에 대하여 기술한다.

Design and Implementation of a High Performance Web Crawler*

Hie-Cheol Kim* · Soo-Hoan Chae**

ABSTRACT

A Web crawler is an important Internet software technology used in a variety of Internet application software which includes search engines. As Internet continues to grow, implementations of high performance web crawlers are urgently demanded. In this paper, we study how to support dynamic scheduling for a multiprocess-based web crawler. For high performance, web crawlers are usually based on multiprocess in their implementations. In these systems, crawl scheduling which manages the allocation of web pages to each process for loading is one of the important issues. In this paper, we identify issues which are important and challenging in the crawl scheduling. To address the issue, we propose a dynamic crawl scheduling framework and subsequently a system architecture for a web crawler with dynamic crawl scheduling support. This paper presents the design of the Web crawler with dynamic scheduling support.

* 본 논문은 2002년 항공대학교 IRC(Internet Information Research Center)로부터 연구비를 지원 받아 수행되었음.

1. 서 론

인터넷의 발전과 더불어 검색엔진의 발전 또한 커다란 변화를 가져왔다. 인터넷이 처음 등장할 당시에는 텍스트 위주의 정보교환이나 의사전달과 같은 커뮤니케이션의 기능 위주이었기 때문에 검색엔진 또한 단순히 문서를 수집해서 분류한 후 실시간으로 제공하는 것이 주된 역할이었다. 그러나 현재는 인터넷기술의 급격한 발전과 이용자들의 증가 및 다양한 요구로 인해서 실시간 정보 제공과 이용자의 특성을 분석한 후 이용자가 원하는 정보를 카테고리별로 서비스 할 수 있다.

검색엔진의 근간이 되는 웹크롤러는 인터넷상에 존재하는 웹 문서들을 추적하여 필요한 정보를 수집하는 기술을 말하며 야후와 같은 인터넷 검색시스템 산업, 전자상거래 상품검색 산업, 인트라넷 검색 소프트웨어 산업 등 현재 대부분의 인터넷 산업의 근간이 되는 핵심 기술이다.

인터넷이 계속적으로 성장함에 따라 그 웹 문서의 양이 매우 거대해지면서 기존의 단일시스템 환경 기반의 웹크롤러 기술은 한계에 직면하고 있다. 기존 국내 기술 수준으로는 400만 웹 문서의 데이터를 수집하는 데 약 20일 정도가 소요되고 있으며 데이터 량이 증가할 경우에는 더욱 많은 시간이 소요될 것으로 예상된다[1,2,3,4]. 향후 5년 이후 국내의 경우 문서 량이 약 1억 페이지가 될 경우 수집에 약 4,600시간이 소요될 것으로 추산되면서 기존의 웹크롤러에 대한 성능향상은 불가피하다.

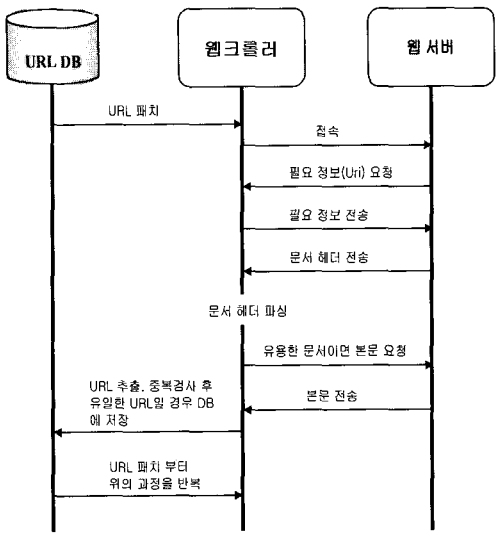
본 논문에서는 수집 속도를 최대화시킬 수 있는 공유메모리를 이용한 동적 스케줄링 기법과 아울러 상대 시스템과 네트웍 과부하를 막기 위한 동적 부하 분산 기법을 제안한다. 또한 그 구현에 관한 가능성(Feasibility), 정확성(Correctness) 등에 관한 연구를 수행한다.

본 논문의 구성은 다음과 같다. Ⅱ장에서는 웹크롤러에 대해 설명하고, 기존 웹크롤러 시스템의 문제점을 고찰하고, Ⅲ장에서는 본 논문에서 제안하는 웹크롤러 시스템의 구조와 특징을 설명하고 Ⅳ장에서는 실제 구현에 대하여 상세히 설명한다. 마지막으로 Ⅴ장에서 결론을 맺는다.

2.연구배경

2.1웹 문서 수집 개념

웹크롤러는 웹 서버를 순회하며 각 홈페이지에 있는 수많은 정보를 수집하는 프로그램으로 사람이 홈페이지의 각 링크를 따라가서 정보를 얻는 반복적인 작업을 대신하여 프로그램이 자동으로 웹 페이지의 내용을 분석하고 그 안의 포함되어 있는 URL들을 추출한 후 그 URL들로 하나씩 접속하면서 정보를 수집한다[5,6,7,8].

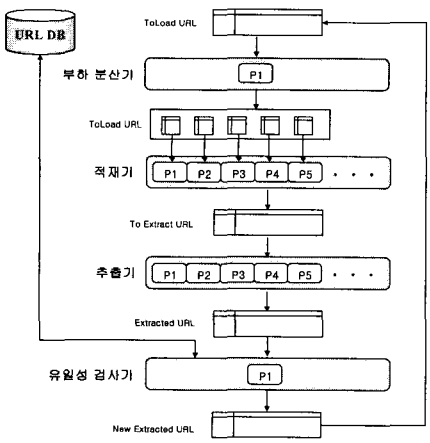


(그림 1) 웹크롤러 동작 알고리즘

웹 문서 수집을 위해서는 수집할 문서의 시작 URL이 있어야한다. 이를 본 논문에서는 ToLoad URL이라한다. ToLoad URL을 적재기가 적재하고 적재한 문서 내에서 참조되는 URL을 추출기가 추출해서 이를 ToLoad URL에 삽입합니다. 위의 과정을 되풀이하면서 자동적으로 문서를 수집하게 된다.

2.2웹크롤러 동작 알고리즘

(그림 1)을 살펴보면 웹크롤러의 동작 알고리즘은 다음과 같다. URL DB에서 하나의 URL을 패치하고, 그 URL의 호스트에 접속한다. 접속이 성공적으로 이루어지면 웹서버는 웹크롤러에 URI를 요청하고 웹크롤러가 URI를 보내면 웹 서버는 문서의 헤더를 보내준다. 웹크롤러는 적재한 헤더 정보를 파싱하여 유용한 문서인지를 체크한다. 유용한 문서이면 웹 서버에 문서의 본문을 요청하고 적재한다[9,10,11,12]. 적재한 문서에서 참조되는 URL을 추출해 내고, 추출된 URL과 URL DB를 비교하여 신규 URL만 URL DB에 삽입합니다. 이 과정을 되풀이 하면서 문서를 수집하게 된다.

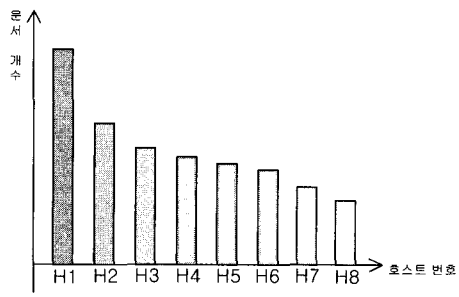


(그림 2) 기존 웹크롤러 모델 구조

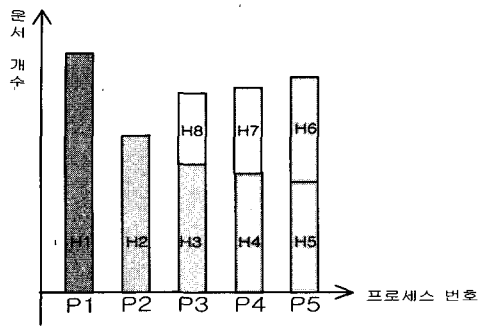
2.3기존 모델 구조 및 동작 사례

(그림 2)의 기존 웹크롤러 모델의 구조를 살펴보면 적재기 모듈은 여러 개의 프로세스를 생성하여 동작하게 된다. 적재기 모듈에서 다음 모듈인 추출기 모듈을 수행하려면 적재기의 모든 프로세스가 종료된 이후에 가능하다. 하지만 적재기 프로세스는 정적 스케줄링에 의해 할당된 URL을 대상으로 문서 수집을 하게 되므로 각 프로세스가 종료되는 시점이 모두 다르다. 그러므로 적재기의 수행 시간은 가장 긴 프로세스의 수행 시간이 된다.

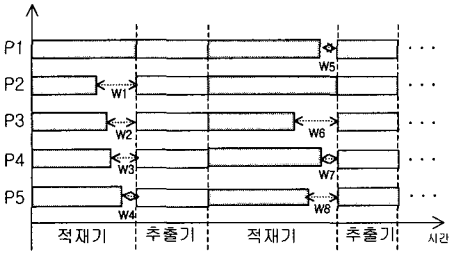
수집해야할 문서들이 호스트 별로 (그림 3)과 같이 존재한다고 가정하고 프로세스 개수를 5개로 하여 동작 사례를 도식화하였다. 일반적으로 사용되고 있는 정적 스케줄링 기법은 적재기가 동작하기 전에 (그림 4)와 같이 프로세스별로 호스트를 할당한다. 그림에서 보는 것과 같이 호스트별로 하나의 프로세스만 접근할 수 있도록 했지만 프로세스별로 할당된 문서 개수는 다르다. 그러므로 적재기 프로세스마다 종료되는 시점이 (그림 5)와 같이 다르게 나타나므로 W1~W8과 같은 시간 지연이 발생한다.



(그림 3) 호스트별 URL 개수



(그림 4) 프로세스별 URL 할당



(그림 5) 기존 모델 동작 사례

3. 제안하는 웹크롤러 시스템

3.1설계 고려사항

인터넷에서 정보검색 서비스를 제공하려면 일단 웹 문서를 수집해야 한다. 원리적으로는 아주 간단하지만 실제로 구현하기 위해서는 몇 가지 어려운 문제를 해결해야 한다. 우선 끊임없이 변하는 웹에 대한 최신 정보를 제공하기 위해 다운로드 성능을 높이는 "속도 문제" 와 한번 다운로드 받은 문서를 다시 다운로드 하지 않게 막도록 하는 "유일성 문제" 그리고 과거에 다운로드 받은 문서가 아직도 유효한지, 혹은 변해서 없어졌는지를 알아내는 "현행화 문제" 가 있다. 그 외에도 로봇 배제 표준과 저작권 문제를 해결해야 한다[5,6,7,8].

웹크롤러 시스템의 설계 및 구현에 있어 기본적으로 다음과 같은 기본 사항을 고려하여야 한다.

웹크롤러의 기본적인 요구사항으로 문서 수집 속도가 빠른 웹크롤러가 가장 성능이 우수한 웹크롤러라고 말 할 수 있다. 동일한 네트워크 환경하에서 웹크롤러가 문서 수집을 할 경우 성능을 향상할 수 있는 방법은 웹크롤러 시스템 내의 처리 속도를 최대화시키는 방법뿐이다. 웹크롤러 시스템의 성능을 향상시키는 방법으로는 일반적으로 멀티 프로세스를 사용한다.

멀티 프로세스로 동작하는 웹크롤러 시스템에서 여러 개의 프로세스가 동시에 한 서버에 문서를 요청한다면 상대 시스템과 네트웍에 과부하를 걸리게 한다. 웹크롤러는 상대 시스템과 네트웍의 과부하를 방지하기 위해 동일 시간에 단일 프로세스만 접근하도록 해야 한다[13].

과부하를 방지하기 위한 또 다른 방법으로 배제 규약이 있다. 로봇 배제 규약은 상대 시스템에 계속적인 접근으로 인해 과부하를 주거나, 일시적인 정보, CGI 등의 적당치 않은 부분을 검색하는 것을 막아보고자 Martijn Koster에 의해 로봇배제 표준이 제안되었다. 웹크롤러 웹서버의 입장으로 가장 간단한 방법으로 "robots.txt" 파일을 생성하는 것이다[14].

3.2제안 모델의 특징

현재 웹크롤러의 문서 수집 속도, 신규URL에 대한 빠른 검색, 등의 요소들이 검색엔진을 평가하는 기준으로 사용되고 있으므로 전술한바와 같이 대용량의 웹 문서를 신속하게 수집하는 웹크롤러 시스템에 대한 연구가 불가피하게 되었다. 본 논문에서는 공유메모리를 이용하여 적재기의 프로세스가 동적으로 URL을 할당하여 문서를 수집할

수 있는 동적 스케줄링 방식을 사용하였고, 문서 적재 시 상대 시스템과 네트웍 과부하 문제를 부하 분산 기술로 해결하였다.

● 멀티프로세싱

문서처리 속도의 향상을 위한 각 모듈들을 다중 프로세스로 처리할 수 있도록 하였다. 또한 공유 메모리 데이터의 동기화를 위해 공유메모리에 접근할 때는 항상 락(lock)을 사용한다.

● 동적 스케줄링

각 적재기 프로세스마다 정적으로 URL을 할당할 경우 각 프로세스간의 종료 시간이 차이가 있으므로 적재기 프로세스가 URL을 동적으로 할당받아 수행할 수 있도록 하였다.

● 동적 부하 분산

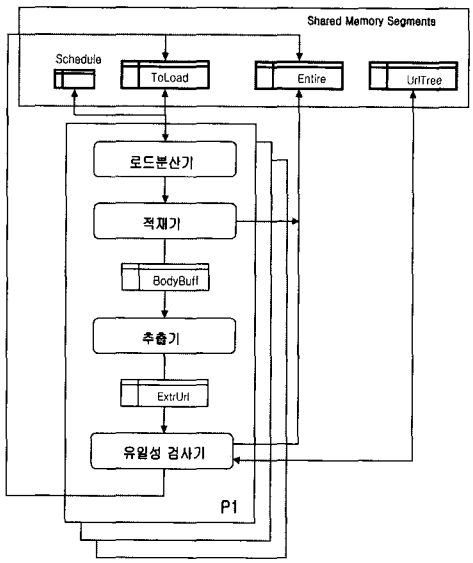
한 개 이상의 프로세스가 동일한 호스트에 접근하면 상대 시스템과 네트웍에 과부하가 걸리므로 각 프로세스가 할당받은 URL의 호스트(host)가 다른 프로세스에서 사용하고 있는지 검사하고, 만약 사용하고 있다면 새로운 URL을 할당받아 문서를 수집할 수 있도록 하였다.

본 절의 나머지 부분에서는 제안하는 웹크롤러 시스템의 구성, 동작 모델 그리고 세부구현에 관하여 기술한다.

3.3제안 모델의 구조

최근 연구에서 웹크롤러의 성능을 판단할 때 가장 먼저 고려해야 할 점은 문서 수집 속도를 최대화하면서 상대 시스템과 네트웍에 과부하를 주지 않는 것이다. 본 논문에서는 이를 위하여 공유메모리를 기반으로한 동적 스케줄링 기법을 제안하여 속도를 향상하였고 동적 부하 분산을 보장한다.

메모리는 크게 공유 메모리와 프로세스 메모리 영역으로 구분된다. (그림 6)에서와 같이 공유메모리는 Entire, UrlTree, ToLoad, Schedule 버퍼가 있고, 프로세스 메모리는 BodyBuff, ExtrUrl 버퍼가 있다. Entire는 전체 URL 정보 저장하기 위해, UrlTree는 유일성 검사를 위해, ToLoad는 적재할 URL 정보 저장을 위해 사용되며, Schedule는 부하 분산을 위해 사용된다. 각 프로세스는 부하 분산기, 적재기, 추출기, 유일성 검사기로 구성되며, 공유 메모리의 데이터를 처리하게 된다.



(그림 6) 제안 모델 구조

3.4 제안 모델의 동작

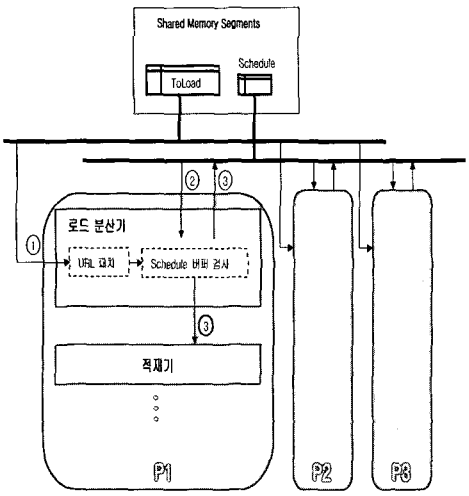
각 프로세스는 문서 수집을 위해 ToLoad URL에서 하나의 URL을 패치한 다음 부하분산을 위해 Schedule 버퍼를 사용한다. (그림 7)과 같이 각 프로세스의 URL 할당은 다음과 같은 순서로 이루어진다.

① URL 하나를 패치한다.

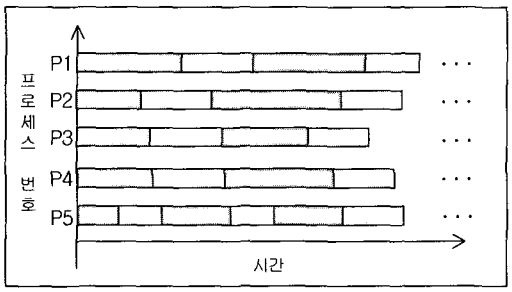
② 패치한 URL의 호스트가 Schedule버퍼에

존재하는지 검사한다.(존재 한다면 현재 해당 호스트에 대해 문서를 수집중이라는 의미이다.)

③ 존재하지 않는다면 Schedule버퍼에 호스트 삽입, 해당 URL 문서를 수집한다.



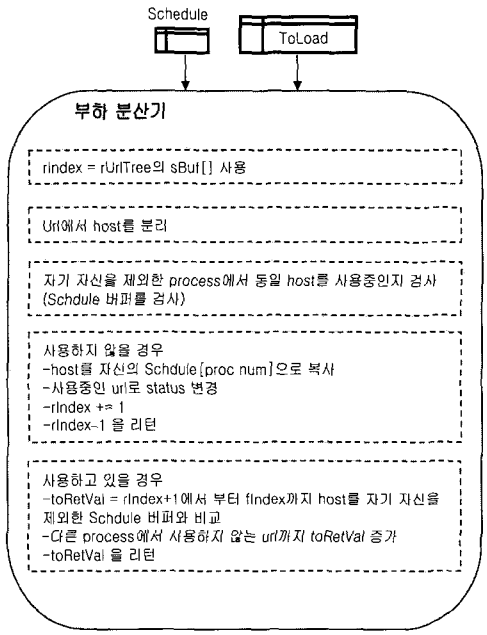
(그림 7) 제안 모델 동작



(그림 8) 제안 동적 스케줄링 기반 수행 모델

- 멀티 프로세스 : 수집 속도를 증가시키기 위해 (그림 2)와 같이 멀티프로세스로 동작하게 된다.
- 동적 스케줄링 : 각 프로세스는 자신이 수집할 URL들을 정적으로 할당받지 않고 공유메모리 영역의 ToLoad URL 버퍼에서 필요시 하나씩 할당받는다.
- 동적 부하분산 : Schedule 버퍼에서는 동일한 호스트가 존재할 수 없다. 즉 동일한 호스트에 대해서 문서 수집을 할 수 없으므로 부하분산이 이루어진다.
- 공유메모리 데이터 동기화 : 공유메모리 영역에 접근 할 때는 항상 락(lock)을 사용하여 데이터 동기화 문제를 해결하였다.

각 프로세스는 문서 수집을 위해 ToLoad URL에서 하나의 URL을 패치한 다음 부하분산을 위해 Schedule 버퍼를 사용한다. 각 프로세스의 URL 할당은 (그림 8)과 같은 순서로 이루어진다.

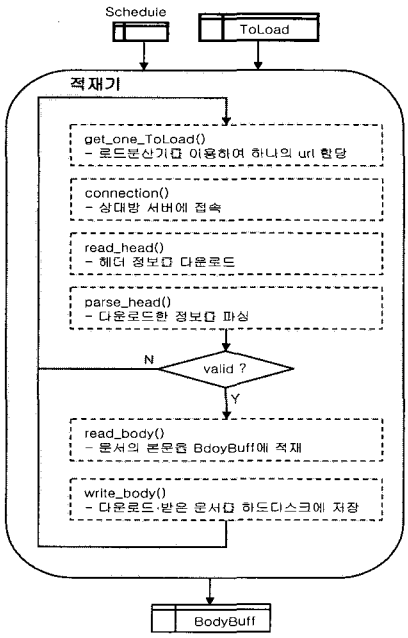


(그림 9) 부하 분산기

4.웹크롤러 시스템의 구현

4.1부하 분산기

한개 이상의 프로세스가 동시에 한 호스트에 접근하게 된다면 상대 시스템과 네트웍에 과부하를 주게 된다. 이를 방지하기 위해 제안하는 시스템에서는 Schedule 버퍼를 사용한다. Schedule 버퍼는 char Schedule[process number][256]로 선언된 2차원 배열이다.

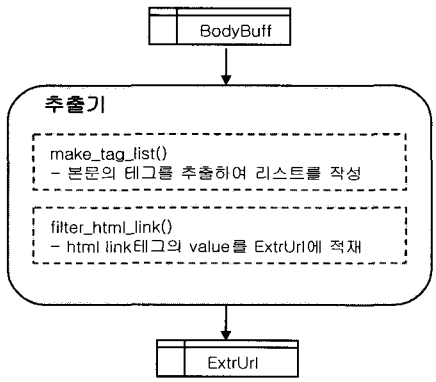


(그림 10) 적재기

(그림 9)와 같이 부하 분산기는 ToLoad버퍼에서 URL을 하나 패치한 다음 Schedule 버퍼에 패치해온 URL의 호스트가 존재하는지 검사한다. 만약 존재하지 않는다면 해당 호스트를 Schedule에 삽입하고 URL의 문서를 적재한다. 존재한다면 다시 ToLoad버퍼에서 URL을 한 개 패치하고 Schedule버퍼와 비교하는 과정을 되풀이한다. 이 과정에서 동일한 호스트는 Schedule버퍼에 존재하지 못하게 된다. 즉 동시에 2개 이상의 프로세스가 한 호스트에 접근하지 못하게 된다.

4.2 적재기

(그림 10)과 같이 적재기는 웹크롤러의 가장 필수적인 모듈로서 URL을 입력으로 하여 해당 URL의 호스트에 소켓 접속한 다음 웹 서버에 URI를 요청하게 된다. 웹 서버는 요청된 URI를 검색한 다음, 검색 결과에 따라 문서의 헤드를 웹크롤러 시스템에 전송해 준다. 웹크롤러는 적재한 헤드를 파싱하여 유용한 문서인지를 검사하고, 유용한 문서이면 본문의 내용을 적재한다.적재한 문서는 웹크롤러의 로컬시스템에 저장된다.



(그림 11) 추출기

4.3 추출기

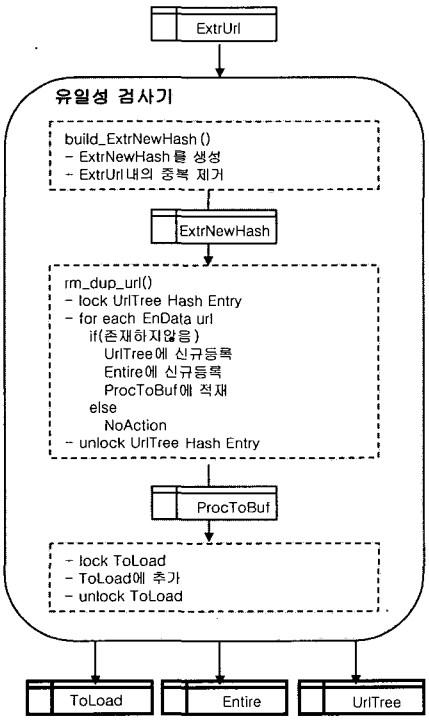
(그림 .11)과 같이 추출기의 주요 기능은 웹 문서 파싱이다. 웹 문서 파싱의 가장 큰 문제점은 사용자들이 HTML 문법에 맞지 않는 코드를 생성해 놓기 때문에 사용자의 잘못된 코드까지 충분히 고려해야 한다는 것이다. 제안 시스템에서는

이러한 문제를 해결하기 위해 문법에 맞지 않는 코드까지 지원할 수 있도록 구성되어 있다.

추출기는 적재기에서 다운로드 받은 BodyBuff에서 태그(Tag)만을 추출하여 만든 태그 리스트(Tag List)를 작성한다. 그리고 작성된 태그 리스트에서 웹 문서 참조(Link)를 추출하여 ExtrUrl에 삽입한다.

4.4유일성 검사기

웹크롤러는 한번 수집한 문서를 다시 수집하는 일이 없어야 한다. 수집한 문서는 URL DB에 저장하고 새로운 URL이 추출되면 URL DB를 검색하여 존재하지 않는 URL만 적재기의 입력으로 삽입해야 한다.



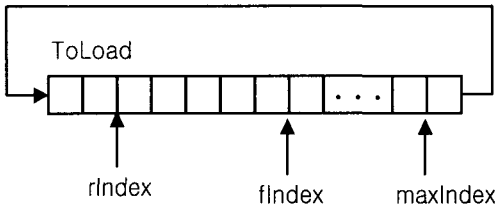
(그림 12) 유일성 검사기

웹크롤러는 다중 프로세스가 동작하므로 URL 유일성 검사(그림 12)를 하는 동안 다른 프로세스는 유일성 검사를 하면 데이터 동기화 문제가 발생한다. 유일성 검사를 하기 위해서는 먼저 URL DB에 lock을 걸고, 유일성 검사를 한 다음 다시 락(Lock)을 해제한다.

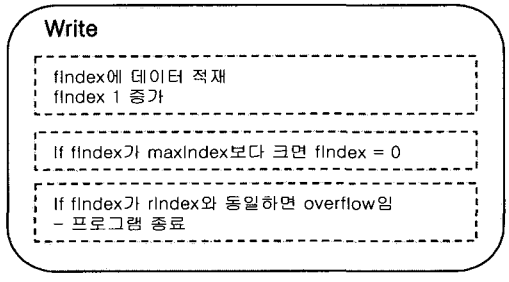
동일시간에 하나의 URL만 유일성 검사가 가능하므로 전체적인 시간 지연이 발생하게 된다.

```
url[strlen(url)*1/11]+(128-url[strlen(url)*1/11) +
url[strlen(url)*3/11]+(128-url[strlen(url)*4/11)+
url[strlen(url)*5/11]+(128-url[strlen(url)*6/11)+
url[strlen(url)*7/11]+(128-url[strlen(url)*8/11)+
url[strlen(url)*9/11]+(128-url[strlen(url)*10/11)
```

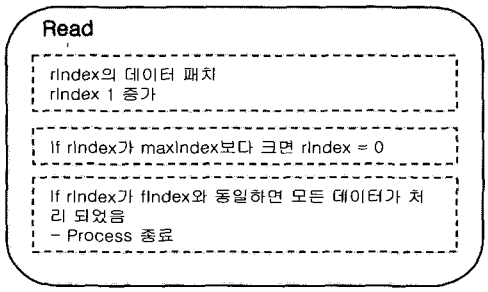
(그림 13) Hash Key 생성 알고리즘



(그림 14) ToLoad 원형큐



(그림 15) ToLoad 버퍼 Write 알고리즘



(그림 16) ToLoad 버퍼 Read 알고리즘

```
#robots.txt for lactt.taegu.ac.kr
User-agent: *
Disallow: /admin/
User-agent: NSS.10
Disallow:
```

(그림 17) robots.txt 파일 사례

제안 시스템에서는 유일성 검사의 시간을 줄이기 위해 다중 루트(Root) 이진 트리 기법을 사용하고 있다. 다중 루트는 해쉬(Hash)로 구성되어 있다. 그러므로 하나의 URL 중복검사를 위해 그 Url의 해쉬 키((Hash Key)를 구한다음 다중 루트에서 해쉬 키에 해당하는 부분만 락이 걸리게 된다. 따라서 해당 URL의 유일성 검사를 위해 하나의 해당 URL의 해쉬 키를 제외한 나머지 루트에서 동시에 중복 검사가 가능하다.

URL DB를 이진트리로 만든 UrTreeHash또한 ExtrNewHash와 동일한 Hash를 구조를 가지고 있으므로 ExtrNewHash의 n번째 Entry에 락을 거는 것은 UrlTreeHash에 n번째 Entry에 락을 거는 것과 동일하다.

Hash Root는 128byte로 구성되어 있고 해쉬 키 생성알고리즘은 다음과 같다. 추출기에서 추출된 ExtrUrl을 입력으로 받아 (그림 13)과 같은 해쉬 키 생성 알고리즘을 사용하여 ExtrNewHash를 만든다. ExtrNewHash의 1부터 128번째 Entry까지 차례로 lock을 걸고 유일성 검사를 한 다음 락을 해제 한다. 유일성 검사에서 통과된 URL들을 ProcToBuf에 저장하고, ToLoad에 락을 건다음 ProcToBuf의 Url들을 ToLoad에 삽입하고 ToLoad의 락을 해제하면 유일성 검사가 끝난다.

4.5ToLoad 버퍼 접근 알고리즘

ToLoad 버퍼는 신규로 추출된 Url을 삽입하고, 적재기에서 ToLoad에 있는 Url을 다운로딩한다. (그림 14)에서와 같이 신규로 추출된 Url을 적재할때는 fIndex를 사용하고, Url을 패치할 때는 rIndex를 사용하는 원형큐로 구성되어 있다. ToLoad 버퍼 쓰기 알고리즘과 일긱 알고리즘은 (그림 15,16)과 같다.

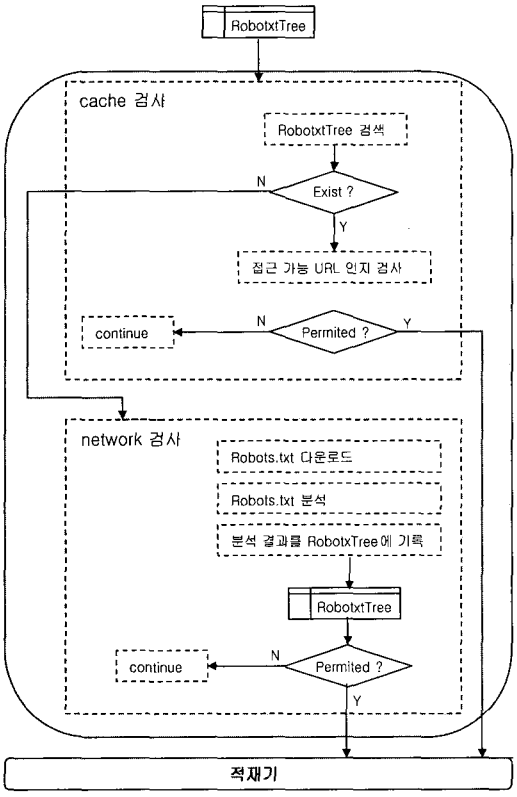
4.6로봇배제 표준

robots.txt에 기술되는 로봇 배재의 포맷은 필드와 값의 쌍으로 기술되며 주석문은 "#"으로 처리한다. 필드는 두 가지로 나눌 수 있는데 에이전트의 이름을 적는 User-agent 필드와, 방문하지 말아야할 URI를 기술하는 Disallow 필드로 구분된다. 또한 빈 robots.txt는 모든 웹크롤러의 접근을 허락한다는 의미이다.

(그림 16)은 로봇 배제 표준의 사례 파일로서 User-Agent의 이름이 NSS1.0인 웹크롤러를 제외한 모든 웹크롤러는 "/admin/"으로 시작하는 페이지의 접근을 제한한다는 것이다.

로봇배제 규약을 준수하기 위해서는 상대시스템의 웹 문서뿐만 아니라 robots.txt 파일도 다운로드하고 이를 분석하여 해야 한다. 즉 문서 하나를 수집하기 위하여 매번 robots.txt파일을 다운로드

한다면 웹 서버에 많은 부하를 줄 뿐만 아니라 문서수집 속도를 감소시킨다. 이 문제를 해결하기 위하여 (그림 18)과 같이 호스트에 대한 로봇배제 정보를 캐싱하여 저장한다.



(그림 18) robots.txt 캐싱 알고리즘

5. 결 론

웹 문서수의 급속한 증가로 인해 보다 수집 성능이 우수한 웹크롤러에 대한 연구가 활발히 진행되고 있다. 하지만 웹크롤러의 성능이 우수해 질수록 상대 시스템과 네트워에 과부하를 부여하게 된다. 기존에는 이 문제를 해결하기 위해 정적 스케줄링 기법을 이용하였다. 하지만 정적 스케줄링 기법은 멀티 프로세스 하에서 프로세스 간 종료 시점이 달라 가장 긴 시간의 프로세스를 제외한 나머지 프로세스에서 시간 지연의 문제가 발생하였다. 본 논문에서는 웹크롤러의 성능향상과 부하분산을 효과적으로 지원할 수 있는 공유 메모리를 사용한 동적 스케줄링 기법을 제안하였다. 또한 이러한 기법을 적용한 웹크롤러의 설계 및 구현에 관한 연구 결과를 발표하였다. 이러한 구현을 바탕으로 하여 현재 구체적인 성능평가 및 분석을 수행하고 있다. 이러한 성능평가 결과는 향후 고성능 웹크롤러의 설계를 위한 방향을 제시할 것으로 기대하고 있다.

References

[1] http://stat.nic.or.kr, http://www.nic.or.kr

[2] http://www.nue.ie, http://www.w3c.org

[3] 송관호, "2001 한국인터넷 통계집", *한국인터넷정보센터*.

[4] 박성득, "2001 한국인터넷 백서", *한국전산원*

[5] Martijn Koster, ConneXions, "Robots in the Web: threat or treat ?", Volume 9, No. 4, April 1995.

[6] M. Wooldridge and N. Jennings, "Intelligent Agents: Theory and Practice," *Knowledge Engineering Review*, 10(2), 1995.

[7] Riechen, Doug, "Intelligent Agents", *Communications of the ACM* Vol. 37 No. 7, July 1994.

[8] Allison Woodruff, Paul M.Akoi, Eric Brewer and Paul Gauthier, "An Investigation of Documents from the World Wide Web".

[9] R. Fielding, J.Gettys, J.C. Mogul, H. Frystyk and T. Berners-Lee, "RFC 2068 Hypertext Transfer Protocol HTTP/1.1", UC Irvine, Digital Equipment Corporation, MIT, 1997.

[10] N.J. Yeager and R.E. McGrath, "Web Server Technology", Morgan Kaufman Publishers, 1996.

[11] T. Berners-Lee, R. Fielding, H. Frystyk, "Hypertext Transfer Protocol -- HTTP//1.0", RFC 1945, May 1996.

[12] Fielding, Roy. "Maintaining Distributed Hypertext Infostructures: Welcome to MOMspider's Web". *Proceedings of the First International World-Wide Web Conference*, Geneva Switzerland, May 1994.

[13] Michele Colajanni, et al, "Dynamic Load Balancing on Web Servers systems", http://computer.org May 1999.

[14] Martijin Koster, "A proposed standard for Robot exclusion", Published on the WWW at http://web.nexor.co.uk/mak/doc/robots/norobots.htmi

김 희 철

1983 : 연세대학교 전자공학과 졸업(공학사)

1991 : Univ. of Southern Californi (Computer Eng. M.S.)

1996 : Univ. of Southern Californi (Computer Eng. Ph.D.)

1983~1988 : (주)삼성전자 주임연구원

1996~1997 : (주)삼성SDS 수석연구원

1997~현재 : 대구대학교 정보통신학부 부교수

관심분야 : 병렬처리, 컴퓨터구조, 컴파일러

채 수 환

1973 : 한국한공대학교 통신공학 졸업(공학사)

1985 : Univ of Alabama (Computer Science. M.S.)

1988 : Univ of Alabama (Computer Science. Ph.D)

1989~현재 : 한국항공대학교, 교수

관심분야 : 컴퓨터구조, 분산컴퓨팅