대화형 인덱싱을 위한 로봇 에이전트의 설계 및 구현

박민우, 박철재
현대정보기술(주) 정보기술연구소

Design and Implementation of a Robot Agent for Interactive Indexing

Min-Woo Park, Chul-Jae Park
R&D Center for Information Technology, Hyundai Information Technology

요 약

에이전트는 분산 환경에서 작업을 수행하는 지적인 특성을 갖는 응용 프로그램으로 정의되며, 연구 분야에 따라 다양한 의미로 해석이 가능하다. 그중에서 로봇 에이전트는 전세계에 산재된 방대한 양의 정보를 스스로 추적하며 새로운 정보를 찾는다. 로봇 에이전트에 대한 기존의 연구는 대부분 통계적인 목적이나 검색엔진을 위한 데이터의 수집을 목적으로 사용되었다. 많은 정보를 수집하기 위해 더 높은 성능의 로봇 에이전트들이 제작되었고, 이러한 프로그램들이 팽창하면서 네트워크를 과부하시키는 현상을 초래하게 되었다. 재귀적인 방법으로 수행되는 로봇 에이전트의 사용을 억제하기 위한 연구들이 많이 발표되었으나, 수동적인 방법에 의존하는 연구가 대부분이며 대표적인 것이 로봇 배제를 위한 표준안 정도이다. 본 연구에서는 이러한 로봇 에이전트의 문제점을 개선하여, 서버와 클라이언트간에 대화형으로 수행되는 인덱스 로봇 에이전트를 제안하며, 사용자의 요구에 따라 수행되는 로봇 에이전트에 의한 정보 획득의 방법을 시도하여 네트워크의 과부하를 억제하면서도 정보의 신뢰성과 정확성을 보장한다.

1. 서론

로봇 에이전트의 시조는 1993년에 전세계의 웹 서버가 몇개 있는지를 조사할 목적으로 MIT대학의 매튜 그래이(Matthew Gray)에 의해 개발된 World Wide Web Wanderer라는 로봇 에이전트가 있다[1]. 이후 많은 검색 엔진들이 만들어 지면서 이러한 로봇 에이전트의 수도 급증하였다. 일반적인 사용자에 의한 경우보다 로봇 에이전트에 의한 네트워크의 과부하가 심각한 문제점으로 대두되었고, 이러한 문제를 해결하고자 제시된 것이 Harvest 프로젝트이다. 한 개의 서버에 대한 지나친 오버헤드를 줄이고 네트워크의 부하의 분산과 다단계적인 인덱스를 통한 효율적인 검색을 제시한다. 본 연구에서는 Harvest를 분석하며 네트워크 프로그래밍 언어인 자바(Java)를 이용하여 새로운 형태의 로봇 에이전트를 구현하고, 대화형 로봇 에이전트를 통해 네트워크의 부하를 주지 않으면서 사용자의 요구에 빠르게 반응할 수 있는 클라이언트-서버 모델을 제안한다[2].

대화형이라는 의미는 고전적인 로봇 에이전트가 스스로 수행되는 것에 비해서, 대화형 로봇 에이전트는 사용자의 요구에 의해서 수행되며, 요구한 결과에 대해서 사용자에게 메시지를 전달해준다. 웹 디렉토리와 로봇을 이용한 검색 엔진의 중간 형태를 취하고 있으며, 자바 에플릿(Applet)과 어플리케이션(Application)을 이용하여 구현된 클라이언트는 서버에게 서비스를 요구하는 메시지를 전달한다. 메시지를 전달 받은 서버는 사용자가 요구한 URL로 로봇 에이전트를 수행시키고, 수집해온 문서에 대해서 인덱스 작업의 수행을 마친다.

본 논문의 구성은 2장에서 Hypertext Transfer Protocol(HTTP), 로봇 에이전트를 고찰하고, Harvest 연구를 분석한다. 3장에서는 제안하는 대화형 로봇 에이전트 시스템에 필요한 자료구조와 알고리즘을 설명한다. 4장에서는 실험 및 분석, 마지막으로 5장에서는 본 논문의 결론과 개선점, 향후 연구과제를 제시한다.

2. 관련연구

본 장에서는 본 연구에서 구현하고자 하는 에이전트의 하부구조를 이루고 있는 Hypertext Transfer Protocol과 로봇 에이전트의 수행목적과 방법을 고찰하고 네트워크 오버헤드의 분산과 효율적인 정보획득을 위한 Harvest 프로젝트를 분석한다.

2.1 Hypertext Transfer Protocol.

Hypertext Transfer Protocol(HTTP, RFC 1945)는 1990년에 시작된 분산된 환경에서, 상호 협조적인 하이퍼 미디어 정보를 위한 가볍고 빠른, 어플리케이션 레벨 프로토콜이다.

HTTP에 관한 기본적인 제안은 Simple Mail Transfer Protocol(SMTP, RFC 821) 에 기초를 두고, Standard for the Format of ARPA Internet Text Message(RFC 822)에 근거하여 기본 골격을 갖추고 있다[3][4]. 또한 HTTP에서는 Multipurpose Internet Mail Extensions(MIME, RFC 1521) 에 따르는 다른 인터넷 프로토콜(NNTP, FTP, Gopher, WAIS)도 지원이 가능하다[5].

[그림 2.1]은 HTTP 메시지를 Augmented BNF로 나타낸 것이다[6].

HTTP 프로토콜하에의 통신은 Request / Response 파라다임을 기본으로 이루어진다. 이 방식은 먼저 클라이언트는 서버와의 연결을 확립하고, Request 메소드, URI, 프로토콜 버전 형태의 Request를 보낸다. 서버는 메시지의 프로토콜 버전과 상태 코드를 포함한 상태 라인을 되돌려 줌으로서 클라이언트, 서버간의 통신이 이루어진다.

2.2 로봇 에이전트

로봇 에이전트는 원하는 문서를 얻기 위해 문서들을 검색 혹은 참조되는 문서들을 재귀적으로 검색하여 웹의 하이퍼 텍스트 구조를 자동적으로 추적하는 프로그램이다. 일반적인 웹 브라우저는 사용자에 의해 운영되고 관련된 문서에 대해서 자동적으로 추적하지 않기 때문에 로봇이라고 하지는 않는다. 이러한 로봇 에이전트의 시초는 Web Crawler로서 현재도 로봇 에이전트에 대한 연구가 지속적으로 이루어지고 있다[7].

2.2.1 수행 목적

통계 분석을 위해 사용할 수 있다. 서버당 문서의 평균수를 포함해서 파일 타입의 분포, 웹 페이지의 평균 사이즈, 상호 연결성의 깊이 등 통계적인 자료를 구축하기 위해 사용될 수 있다.

유지보수(Maintenance)를 위해 사용할 수 있다. 하이퍼 텍스트 구조를 유지하는데 주된 어려움에 하나는 레퍼런스 되어지고 있는 페이지가 옮겨졌거나 없어졌을 경우에 해당하는 죽은 링크(dead link)가 발생하는 것이다. 이런 경우 MOMSpider와 같이 레퍼런스를 입증하는 로봇은 죽은 링크를 찾는데 도움을 줄 수 있다.

미러링(Mirroring)을 위해 사용할 수 있다. 미러링은 FTP 아카이브를 유지하는 일반적인 테크닉으로 호스트의 파손을 대비하고 서버 로드를 분산시키는 것을 목적으로 하고 있다. 웹에서는 미러링은 로봇으로 구현될 수 있다. 그러나 단순히 FTP 미러링같이 복사만 하는 것이 아니라 상대 경로로 쓰여진 레퍼런스는 다시 절대 경로로 바꾸어 쓰여져야 할 필요가 있다.

리소스 발견을 위해 사용할 수 있다. 현재 가장 많은 목적으로 사용되어 지고 있는 분야로서 사람이 처리할 수 없는 막대한 양의 정보에 대해 컴퓨터가 처리할 수 있도록 한 것으로서 로봇이 가져온 결과에 대해서 문서를 요약하고 이들 결과에 대해서 검색 엔진을 통하여 데이터베이스의 접근을 제공하고 있다.

복합적인 목적으로 사용할 수 있다. 위에서 열거된 목적 중에 두 가지 이상을 동시에 처리할 수 있다. RBSE Spider는 리소스 발견 데이터베이스를 제공할 뿐만 아니라 검색된 문서들의 통계분석까지 처리할 수 있다[8].

- *HTTP-message = Simple-Request | Simple-Response | Full-Request | Full-Response*
- *Full-Request = Request-Line * (General-Header | Request-Header | Entity-Header) CRLF [Entity-Body]*
  Request-Line = Method Request-URI HTTP-Version CRLF
  Request-Header = Authorization | From | If-Modified-Since | Referer | User-Agent
  Method = GET | POST | HEAD | extension-method
- *Full-Response = Status-Line * (General-Header | Response-Header | Entity-Header) CRLF [Entity-Body]*
  Status-Line = HTTP-Version Status-Code Reason-Phrase CRLF
  Response-Header = Location | Server | WWW-Authenticate
  Status-Code = 200 | ... | 503 | extension-code
- *Simple-Request = GET Request-URI CRLF*
- *Simple-Response = [Entity-Body]*

[그림 2.1] HTTP 메시지의 Augumented BNF

2.2.2 순회 방법

로봇의 순회 방법은 크게 두 가지로 나눌 수 있다. 우선 넓이-우선 순회(Breadth-first traversal)으로서 상위 레벨에서 제한된 깊이까지만 순회한다. 상위 레벨의 문서와 서비스의 집합을 찾는 데 좋은 특성을 가진다. 즉 한 문서에 여러 개의 레퍼런스가 존재할 때 각각의 레퍼런스에 대해 먼저 검색을 하고 그 다음 한 단계 깊이 들어가서 검색하는 방법으로 주로 새로운 리소스의 발견을 위해서 사용된다. 또 다른 방법으로는 깊이-우선 순회(Depth-first traversal)이 있다. 이 방법은 한 문서에 여러 레퍼런스가 있을 경우 한 레퍼런스에 대해서 계속 따라가며 검색하는 방법이다. 이 방법은 새로운 사이트를 찾는데 유리하다.

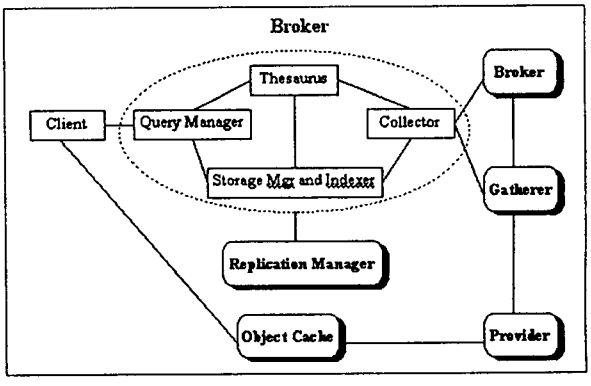
2.3. Harvest

Harvest는 Internet Research Task Force Research Group on Resource Discovery (IRTF-RD)에서 설계되고 만들어진 인터넷을 통해 관련된 정보를 모으고, 검색하기 위한 통합된 형태의 도구이다. 여기서는 수집, 추출, 조직화, 검색, 캐쉬 등이 이루어진다. Harvest는 다양한 인덱스 타입을 재구성할 수 있는 융통성 있는 시스템을 지원하는 것을 목표로 하고 있으며, 인터넷 서버와 네트워크 링크, 인덱스가 차지하는 디스크량을 매우 효과적으로 사용할 수 있게 한다. Harvest는 자신들의 측정 기준에 따르면 Archie, WAIS, World Wide Web Worm등과 비교할 때 서버 부하를 6000개, 네트워크 트래픽의 경우 60개, 인덱스량의 경우 40개의 요소를 절감할 수 있다. 또한 Harvest는 타이틀 필드로부터 정규화 된 표현을 사용한 질의를 통하여 다양한 포맷으로부터 속성-값의 쌍으로 이루어진 구조화된 정보를 사용자가 추출할 수 있도록 허용한다[9].

2.3.1 시스템 소개

Harvest는 몇 가지 서브 시스템으로 구성되어 있다. Gatherer 서브 시스템은 HTTP나 FTP같은 정보 제공자로부터 사용 가능한 자원들(키워드, 저자 이름, 타이틀 등)의 인덱싱된 정보를 모은다. Broker 서브 시스템은 하나 또는 그 이상의 Gatherer로부터 인덱싱된 정보를 검색한다. Replicator 서브 시스템은 인터넷에 있는 Broker들을 효과적으로 복제한다. 사용자들은 Cache 서브 시스템을 이용하여 정보들을 보다 빠르게 검색이 가능하다. Harvest Server Registry(HSR)는 각 Harvest 구성요소들의 정보를 가지고 있는 Broker를 구분해 낸다.

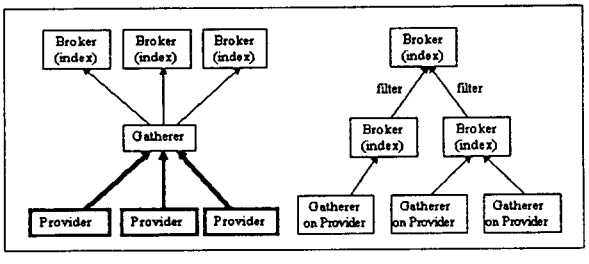
[그림 2.2]은 Harvest 서브 시스템의 개요를 설명한 것이다.



[그림 2.2] Harvest 소프트웨어 요소들

Harvest는 일반적인 데이터의 접근시 네트워크 트래픽을 줄이기 위해 Object Cache를 실행 시킨다. 기본적인 구성을 갖춘 뒤에 추가적인 효과로서 대용량의 정보를 인덱스할 때 분산된 Gatherer 프로세스를 사용하여 CPU 및 네트워크 부하를 줄일 수 있고, 데이터 타입을 비교할 때 주어진 정보에 대해서 어떻게 추출하고, 인덱스하고, 찾을 것인지에 대한 방법을 제공할 수 있다. 또한 Replicator를 사용하여 Broker의 과부하도 절감시킨다.

[그림 2.3]은 Harvest Gatherer와 Broker가 다양한 방법으로 재구성될 수 있음을 보여준다. 왼쪽 그림은 Gatherer 가 정보제공자 사이트로부터 원격적으로 수행되는 것을 허용함을 보여준다. Harvest는 FTP, Gopher, HTTP, NNTP같은 표준적인 객체 Retrieval 프로토콜을 이용하여 Gatherer가 수행되지 않는 사이트들과의 상호처리가 가능하다.



[그림 2.3] Harvest 구성 옵션

그러나 왼쪽 그림의 굵은 선으로 표시된 Provider의 경우 Gatherer 서버와 네트워크에 부하를 초래하게 된다. 여기에 비해서 오른쪽 그림에서와 같이 Gatherer가

정보제공자 사이트의 로컬에서 독립적으로 자신의 인덱스 정보를 모을 때 보다 효과가 크다. 그럼에도 불구하고 많은 Broker와 WAIS나 Glimpse같은 다른 검색 엔진들은 Gatherer가 모아둔 인덱스 정보를 공유할 수 있기 때문에 왼쪽 그림에서와 같이 Gatherer가 원격적으로 실행되는 것이 오른쪽 그림같이 많은 사이트들이 독립적으로 인덱스 정보를 모으는 것보다 더 좋다.

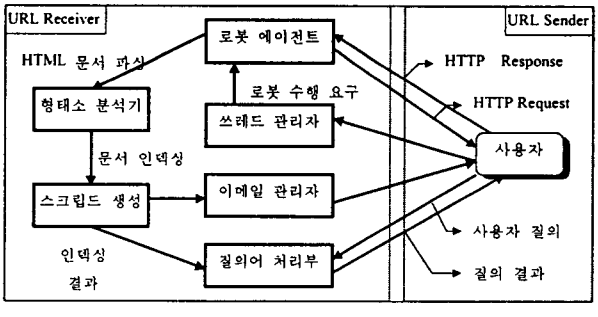
[그림 2.3]에서와 같이 Broker는 넓게 분산된 정보를 인덱스로 만들기 위한 많은 Gatherer들로부터 정보를 모을 수 있다. Broker들은 다른 Broker들로부터 인덱스된 view를 통하여 다 단계적인 정보 검색이 가능하며 하나의 Broker에서 다른 Broker로 이동될 때 필터 등을 허용하여 사용자 질의 인터페이스를 사용하는 정보 검색이 가능하게 한다.

3. 대화형 로봇 에이전트의 구현

본 장에서는 제안한 대화형 인덱스 로봇 에이전트의 시스템 구성을 설명하고, 본 연구에서 구현한 URL Receiver, URL Sender의 자료구조, 알고리즘과 형태소 분석 기법에 대한 내용을 설명하고 구현한다.

3.1. 시스템 구성

대화형 인덱스 로봇 에이전트의 구현 환경은 [그림 3.1]과 같이 인터넷에 연결된 HTTP 서버가 탑재된 워크스테이션을 대상으로 하며, 필요에 따라 여러대의 워크스테이션을 네트워크 파일 시스템으로 연결하여 분산처리로 수행한다.



[그림 3.1] 시스템 구성도

사용자는 URL Receiver와 대화형으로 작업을 수행하기 위해서 URL Sender를 사용해서 서버구조의 쓰레드 관리자에게 서비스를 요청하며, 이것은 Hypertext Transfer Protocol을 사용하여 서버에서 수행될 수도 있고 독립적인 프로그램으로 클라이언트에서 수행될 수도 있다.

URL Sender는 TCP/IP 네트워크 프로토콜 기반위에서 소켓 라이브러리를 사용하여 서버에게 메시지를 전달한다. URL Receiver와 URL Sender는 SUN Micorsystems사의 자바 프로그램밍 언어를 사용하여 개발 되었으며, JDK가 탑제된 모든 워크스테이션과 PC에서 바이너리 호환이 가능하다. 따라서 높은 성능의 머신으로 업그레이드시 시스템의 재구성없이도 이전 작업과 호환이 유지된다. URL Sender가 자바 에플릿으로 수행될 경우에 JDK가 없이도 자바 인터프리터 가상머신이 탑재된 모든 브라우져에서 수행이 가능하다.

3.2 URL Sender

사용자에 의해서 수행되는 클라이언트 구조의 URL Sender는 쓰레드 관리자에게 서비스 받고자하는 URL을 전달하는 것을 목적으로 한다. 이때 쓰레드 관리자에게 전달하고자 하는 패킷 구조는 [그림 3.2]와 같다.

```
private class message_type
{
  String email;  // 수신의 성패 결과를 받기 위한 메일주소
  String url;    // 서비스 받고자 하는 URL
  String property;  // 통계를 위한 시스템 정보들
}
```

[그림 3.2] URL Sender 패킷 구조

URL Sender의 가장 중요한 요소는 프로그램의 크기를 줄이는 것이다. 이것은 사용자가 직접 프로그램을 가져가도록 요구할 수도 있고, 사용자가 서버에 Hypertext Transfer Protocol로 접속하였을 경우 자바 에플릿 형태로 다운로드시켜서 수행할 수도 있다. 에플릿의 경우 클라이언트 프로그램이 비대해 지면 사용자 입장에서는 많은 다운로드 시간과 수행시간을 낭비하게 되므로 본 구현에서는 클라이언트 프로그램인 URL Sender의 크기를 줄이는데 촛점을 맞추었다. 실제로 URL Sender는 어플리케이션의 경우 5K 바이트, 에플릿의 경우 4K 바이트의 크기를 가진다.

URL Sender는 초기 실행시 사용자로 부터 2가지 입력을 요구한다. 하나는 사용자의 이메일(Email) 주소이다. 이것은 자신이 요구한 서비스의 결과를 받기 위한 것으로, 메시지 전달이 끝나면 서버에서는 백그라운드로 프로세스를 생성시켜 요구하는 작업을 수행하고 모든

수행이 종료하면 메시지로부터 받은 이메일 주소를 사용하여 성패여부를 사용자에게 메일로 보내지게 된다. 이메일 주소는 최초 수행시만 입력을 요구하며, 로컬의 디스크에 이메일에 대한 정보를 남겨두고, 두번째 이후에 사용부터는 디스크의 정보를 사용하여 패킷구조에 자동적으로 삽입하도록 한다.

두번째로 요구되는 입력은 서비스를 받고자 하는 URL 주소이다. 이것은 반드시 RFC 1738에서 요구하는 URL 형식을 따른다[10]. URL Sender에서는 사용자가 입력한 URL이 위의 형식에 일치하는지를 검사한다. 이때 URL을 파싱하여 각각의 요소별로 검사를 시도하여, 모든 것이 문제가 없다고 판단되면 서버에게 보내기 위한 패킷 구조에 이 정보를 포함시킨다. 마지막 정보는 패킷을 보내기 직전에 사용자 환경의 특성을 추출하여 서버에게 함께 보내지게 된다.

3.3 URL Receiver

URL Receiver는 서버 구조의 쓰레드 관리자, 로봇 에이전트 및 이메일 관리자로 구성되며, UNIX의 rc.local 화일에 등록되어 서버 시스템이 초기화 될 때 실행을 시작하여 시스템이 종료될 때까지 백그라운드로 수행된다. 클라이언트 구조의 URL Sender가 연결을 시도하면 쓰레드 관리자는 쓰레드를 생성시켜 메시지를 받을 준비를 한다. URL Sender로부터 메시지가 전달되면 에이전트를 수행시키고, 에이전트의 수행이 끝나면 형태소분석및 인덱스 생성을 위한 스크립트 생성기를 프로세스로 수행시킨다. 프로세스가 수행을 마치면 URL Receiver에게 메시지를 보내고, 다시 메시지의 결과를 이메일 관리자한테 보내서 최종적인 결과를 URL Sender를 수행시킨 사용자에게 메일로 전송함으로서 수행을 마치게 된다.

3.3.1 쓰레드 관리자

쓰레드 관리자는 대화형으로 동작하는 URL Receiver의 가장 상위에 존재하는 부분으로서 모든 클라이언트의 연결을 쓰레드로 관리하며, 생성된 쓰레드는 로봇 에이전트를 수행과 프로세스 생성을 담당한다. 최초 수행시 쓰레드 관리자는 자바 ServerSocket 클래스를 이용하여 소켓을 생성한뒤에 클라이언트가 연결하기를 기다린다. 클라이언트가 연결되면 Cloneable 클래스를 이용해서 복제본을 만들고, 다시 이 복제본은 Runnable 클래스를 통해서 쓰레드로 생성된다. 생성된 쓰레드는 자신의 생명주기(Life-Cycle)에 따라 수행된다.

[그림 3.3]은 스레드 관리자가 대기상태에서 새로운 클라이언트의 연결이 시도될때 마다 새로운 쓰레드로 연결을 생성하는 알고리즘을 보여준다. 서버 소켓을 생성하기 위해서는 서비스할 포트 번호와 연결 제한 시간을 필요로 한다. 클라이언트와 연결을 대기하기 위해서 소켓을 생성하고 accept() 메소드를 사용한다. clone() 메소드는 현재 클래스를 복제하고, Thread() 메소드를 통해서 새로운 쓰레드를 생성한 뒤에 start() 메소드는 쓰레드를 수행시킨다. 클라이언트로 부터의 메시지 처리는 handleSession() 메소드에서 이루어 진다.

```
// 서버 소켓을 생성한다
svrSock = new ServerSocket(port, delay);
// 클라이언트가 접속하기를 대기
sock = svrSock.accept();
. . .
// 새로운 쓰레드를 위한 복사본을 만든다
copy = (URLReceiver) clone();
twin = new Thread(copy);      // 새로운 쓰레드를 생성한다
twin.start();                 // 쓰레드를 수행
. . .
// 클라이언트 메세지를 받기 위한 스트림
in = sock.getInputStream();
handleSession(in);            // 실제로 작업을 위한 함수
shutdown();                   // 작업을 종료한다
```

[그림 3.3] 쓰레드 관리자 알고리즘

3.3.2 로봇 에이전트

로봇 에이전트를 크게 소켓 연결부, Request 수행부, Response 수행부, URL 파싱, URL 추출, 인덱스 관리 등으로 구분하였다. 소켓 연결부는 자바의 java.net.Socket 클래스를 이용하여 구현되었으며, Request/Response 수행부는 HTTP 규약을 따른 포멧으로 구성되며 URL 파싱은 java.net.URL 클래스로 구현되었다.

쓰레드 관리자에 의해 수행되는 로봇 에이전트는 단일 수행을 목적으로 한다. 고전적인 로봇 에이전트의 재귀적인 방법을 사용하지 않고 쓰레드 관리자가 수행을 요구할 때만 작동하며, 로봇 에이전트에 의해 수집된 문서는 로컬 파일 시스템에 일정한 형식을 따르는 파일명으로 저장된다. 쓰레드 관리자가 파라미터로 로봇 에이전트에게 수행해야 될 URL을 넘겨주면 로봇 에이전트는 자체적으로 URL을 파싱하여 호스트, 포트번호, 레퍼런스를 추출해 낸다. 이때 표준 URL형식에 대한 검사 과정도 같이 수행한다. 최종적으로 파싱된 요소들에 대해서 Request를 위한 데이터 타입으로 변환하여 소켓을 통해 서버에 메시지를 전달한다.

[그림 3.4]는 로봇 에이전트의 수행절차를 나타낸다. 원리는 소켓을 이용하여 웹 서버에 접속한 뒤에 스트림을 통하여 Request/Response 규약을 따른 통신을 위한 것이다. 일단 연결이 설정되면 스트림으로 Request 메시지를 보내고, 메시지 전달이 끝났음을 알리면, 서버는 다시 스트림을 통해 Response 메시지를 되돌려 준다. 하지만 여기서 상태 코드를 분석하여 유용한 정보일때만 파일로 저장한다

3.3.3 이메일 관리자

이메일 관리자는 쓰레드 관리자에게 요구한 서비스의 최종결과를 인터넷 메일 형태로 사용자에게 되돌려 주기 위한 목적으로 구현되었으며 여기서는 RFC 821, SMTP를 따르는 구조로 이루어져 있다.

SMTP에서 메시지 교환은 우선 메일 포트로 소켓을 연결하여 원격 메일 서버의 Response의 상태코드가 220이면 서비스를 받을 준비가 되었다는 의.미이다. 각 단계는 부분적으로 생략이 되어질 수 있으며 메일을 받을 사람의 주소는 생략될 수 없다. 이것은 RCPT TO:라는 키워드로 시작하며 메시지의 전달의 끝은 \r\n로 구분된다. 본문 내용은 DATA라는 메세지를 전달한 후에 이루어지며 본문의 끝은 \r\n . \r\n로 나타낸다. 본문 내용중에 마침표(.)가 나올 수 있기 때문에 반드시 위와 같은 형식일 때 본문의 끝을 의미한다. 본 구현에서는 본문의 내용을 이미 지정된 파일을 읽어서 보내주게 된다. URL Receiver는 수행하는 각 단계에서 에러가 검출되면 최종적인 결과는 수행이 실패 되었음을 내용으로 하는 파일을 전달하고, 모든 수행이 정상적으로 완료되면 성공되었음을 내용으로 하는 파일을 전달하게 된다. 이메일 관리자는 쓰레드 관리자에게 요구한 서비스의 최종결과를 인터넷 메일 형태로 사용자에게 되돌려 주기 위한 목적으로 구현되었으며 여기서는 RFC 821, SMTP를 따르는 구조로 이루어져 있다.

3.3.4 형태소 분석기

형태소 분석은 한국어 어절을 형태소라는 의미적 최소의 단위로 분리하는 것을 말한다. 한글 문서는 구조적인 특성상 띄어쓰기의 오류및 조사활용에 대한 변화가 다양하기 때문에 의미있는 단어를 찾아내기 위해서는 형태소 분석은 필수적인 요소라 할 수 있다. 본 연구에서는 HTML 문서에 대한 테그 파싱 처리가 이루어진 결과에 대해서 형태소 분석을 수행하여 유용한 단어를 찾아내는 방법을 사용하였다.

우선 불용어에 대한 선처리가 이루어 져야 한다. 불용어라는 것은 토큰단위로 구분된 문자열에 대해서 의미없는 조사들에 해당하며, 이때 조사사전을 이용한 매칭기법을 사용한다. 조사에 대한 검사가 잦고 조사의 갯수가 많지 않은 만큼 조사사전은 메모리에 상주시켜 사용함으로서 매칭 수행 속도를 향상시켰다. 하나의 토큰에 대해서 조사를 찾는 방법은 토큰의 뒤에서부터 매칭을 시도하여 제거하는 방법을 사용한다. 이때 속도를 높

```
try {
        sock = new Socket( host, port );                    // 지정된 호스트와 연결
        in = new DataInputStream( sock.getInputStream() );  // 서버와 통신하기 위한 스트림 설정
        out = new PrintStream( sock.getOutputStream() );
        out.println( Full-Request-Message );                // Request 메시지를 보내다

        // 서버 Response 메시지의 상태코드를 분석하여 2xx 계열이 아니면 소켓을 닫고 리턴한다
        if ( !Status-Code-Check ) {
                sock.close();
                return false;
        }

        for ( Header-Information ) outgoing = in.readLine(); // Response 메시지중에서 헤더정보는 스킵한다
        while ( End-of-Document ) content += outgoing;
        fileSave( content );                                // 문서 내용을 파일로 저장한다
        sock.close();
        return true;
}
catch ( UnknownHostException e ) { }
catch ( IOException e ) { }
catch ( Exception e ) { }
```

[그림 3.4] 로봇 에이전트 수행 알고리즘

이기 위해서 인덱스를 유지하는데 이때 한글 초성에 대한 인덱스는 효과를 보기가 어렵다. 본 연구에서 사용된 방법은 음절의 수를 이용한 인덱스 기법으로서 토큰 길이 - 1개의 음절수 이하를 대상으로 조사 사전과 비교하는 방법을 채택하고 있다.

불용어 제거가 끝난 결과에 대해서 다시 명사 사전과 비교해서 의미있는 단어를 추출하는 작업이 필요하다. 명사 사전은 방대한 양의 데이터를 유지하기 때문에 메모리에 상주할 수 없다. 그래서 외부 화일과 더불어 인덱스을 유지하여야만 한다. 인덱스의 구성은 한글 초성에 바탕을 두고 있으며, 불용어가 제거된 토큰에 대해서 첫 문자의 초성을 구한 뒤, 각 초성에 해당하는 위치에서부터 매칭작업에 들어가게 된다. 이 부분이 형태소 분석의 전체 과정에서 가장 많은 부하를 가져오는 부분으로서 인덱스 구조를 사용하여 수행 속도를 증가시켰다.

4. 실험 및 분석

로봇 에이전트의 수행 속도는 원격서버의 파일크기에 종속적이다. 즉 전형적인 로컬 파일시스템과 같은 성능을 같는다. 그러나 로봇 에이전트의 수행 대상이 인터넷이기 때문에 이러한 파일크기보다도 네트워크 속도에 더 많은 영향을 받는다. 실제로 테스트한 결과에 의하면 네트워크에 부하가 많이 걸리는 오후시간이 가장 낮은 성능을 보이며, 외국의 경우가 국내 사이트들 보다 낮은 성능을 보인다. 또한 휴일의 경우는 네트워크 부하가 현저히 감소하기 때문에 수행속도가 증가하게 된다.

문서를 인덱싱하여 스크립트를 생성하는 수행에서, 성능은 영문과 한글의 문서에서 상당한 속도의 차이를 가진다. 영문의 경우 조사사전의 크기가 작고 명사사전을 이용한 비교를 하지 않기 때문에 매우 빠른 속도를 가진다. 여기서 사용된 실험대상은 한글문서와 영문문서를 구분하여 각각 1K, 10K, 100K, 1M의 크기를 갖는 파일을 대상으로 UNIX의 time명령을 사용하여 측정하였다. [표 4.1]은 대상이 되는 문서의 크기에 따른 한글과 영문에 대한 성능분석을 나타낸다.

[표 4.1] 에서 보는 바와 같이 파일의 크기가 커질수록 한글을 처리하는 부분은 상당한 부하를 준다는 것을 알 수 있다. 그러나 영문의 경우도 1Mega byte 이상의 크기에서는 현저하게 성능이 감소되는 현상을 보인다.

문서 크기	문서 종류	수행 시간			전체 토큰 수	추출된 명사 수
		real time	user time	system time		
1K byte	한글	1.5	1.5	0.0	150	29
	영문	0.0	0.0	0.0	169	68
10K byte	한글	11.3	11.0	0.1	1,380	183
	영문	0.2	0.2	0.0	1,444	448
100K byte	한글	1:12.6	1:11.1	1.1	14,138	1,076
	영문	2.6	2.5	0.0	8,653	2,113
1M byte	한글	6:58.4	6:50.7	2.8	141,866	3,533
	영문	3:05.4	3:03.5	0.1	148,734	11,077

[그림 4.1] 한글과 영문에 따른 인덱싱 속도 비교

이러한 현상의 원인은 추출된 단어를 저장하기 위한 파일의 크기가 비례적으로 확장하기 때문이다. 본 연구에서는 실험대상을 한글과 영문을 구분하여 비교를 하였으나, 실제 문서는 한글과 영문이 혼합되어 있는 경우가 많기 때문에 인덱싱 과정에서 한글과 영문을 분리해서 처리한다는 것은 성능의 향상에 많은 영향을 준다. 초기 토크의 형태로 넘어오는 스트링에 대한 처리부분의 가장 상위에 한글과 영문을 검사하여 독립적인 인덱싱 루틴을 수행한다. 한글처리 부분과 영문처리 부분을 독립시키지 않을 경우 한글의 처리과정을 영문의 경우도 같은 형태로 수행되지 때문에 자원의 낭비를 초래한다. 실제로 대부분의 웹에서의 문서는 10K byte미만의 크기를 가지며 한글과 영문이 혼합되어 있기 때문에 대화형 인덱스 에이전트의 작업수행에 부하를 주지는 않는다.

5. 결론

로봇 에이전트에 대한 기존의 연구는 대부분 통계적인 목적이나 검색엔진을 위한 데이터의 수집을 목적으로 사용되었다. 그러나 World Wide Web의 기하급수적인 증가와 더불어 산재된 많은 정보를 수집하기 위해 더 높은 성능의 로봇 에이전트들이 제작되었고, 이러한 프로그램들이 팽창하면서 실제로 전체 네트워크를 과부하시키는 현상을 초래하게 되었다. 재귀적인 방법으로 수행되는 로봇 에이전트의 사용을 억제하기 위한 연구들이 많이 발표되었으나, 수동적인 방법에 의존하는 연구가 대부분이며 대표적인 것이 로봇 배제를 위한 표준안 정도이다. Harvest는 서버와 클라이언트 사이의 공조를 통한 정보 획득 방법을 제시하여 재귀적인 수행를 하지 않고도 정보를 수집할 수 있는 방법들을 제시하였다.

본 구현에서는 서버와 클라이언트간에 대화형으로 수행되는 인덱스 로봇 에이전트를 제안하여, 사용자가 요구에 따라 수행되는 로봇 에이전트에 의한 정보 획득의 방법을 시도하여 네트워크의 과부하를 억제하면서도 정보의 신뢰성과 정확성을 보장한다. 자바로 구현된 URL Receiver와 URL Sender로 불리우는 서버, 클라이언트 프로그램은 메시지를 전달방법으로 사용하며, 멀티쓰레드를 이용하여 서버는 클라이언트의 요청을 효과적으로 처리할 수 있으며, 예외상황 처리부를 통해 네트워크상에서 프로그램의 수행들의 안전한 종료를 보장한다. 자바는 프로그래밍 언어차원에서 멀티쓰레드를 지원하며 Blocking을 위한 록킹 메카니즘을 지원하여, 기존 커널차원에서의 멀티쓰레드를 관리하는 것보다 용이해졌다.

URL Receiver는 크게 쓰레드 관리자, 로봇 에이전트, 이메일 관리자로 구분되어 수행되어지며, 쓰레드 관리자는 인덱싱을 위한 형태소 분석기 및 검색을 위한 스크립트 생성을 수행한다. 형태소 분석은 실험 및 분석에서의 평가를 살펴보면 한글과 영문의 경우 수행 시간의 차이를 보이며, 파일크기에 따른 수행시간에서 비교되었듯이 파일크기가 증가함에 따라 영문의 경우 추출된 명사를 유지하기 위한 오버헤드가 증가함을 알수 있다. 그러나 실제로 웹을 통해 수집된 자료가 10K byte 미만이며, 한글과 영문이 혼합된 자료가 많기 때문에 수행속도는 서버에 큰 부하를 주지 않는다. 동시 사용자가 늘어나더라도 쓰레드 관리자에 의해 연결시에 발생하는 부하를 최소화 시키며, 인덱싱에 의한 부하도 색인을 이용한 사전관리를 통해 효율적인 수행을 보장한다. 본 구현에서는 서버 클라이언트간의 메시지를 통한 정보수집 과정에 촛점이 맞추어 졌으나, 향후 연구과제로서 질의어 처리부에서 구절검색및 구조적 질의를 통한 다양한 질의방법을 수용할 수 있어야 하겠다.

참고문헌

[1] Matthew Gray, "Internet Statistics: Web Growth, Internet Growth", http://www.mit.edu/afs/sipb/user/mkgray/ht/net/, Jun 20 1996

[2] Mary Campione, Kathy, Walrath, "The Java Tutorial", http://java.sun.com/tutorial Mar 4 1996

[3] J.Postel, "Simple Mail Transfer Protocol", RFC 821, August 1982

[4] David H. Croker, "Standard for the Format of ARPA Internet Text Messages", RFC 822, August 1982

[5] N.Borenstein, N.Freed, MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies., RFC 1521, September 1993

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

[7] Martijn Koster, "World Wide Web Robots, Wanders, and Spider" http://info.webcrawler.com/mak/project/robots/robots.html, 1995

[8] Martijn Koster, "Robots in the Web: threat or treat?", http://info.webcrawler.com/mak/project/robots/threat-or-treat.html, 1995

[9] Darren R. Hardy, Michael F. Schwartz, Duane Wessels Harvest Users Manual, University of Colorado at Boulder Technical Report CU-CS-743-94, January 31 1996

[10] T. Berners-Lee, L. Masinter, M. McCahill, Uniform Resource Locators, RFC 1738, December 1994