역파일에 기반한 웹 검색 엔진의 랭킹 시스템 구현*

임성채°, 안준선
동덕여자대학교 정보대학 컴퓨터전공°, 한국항공대학교 항공전자및정보통신공학부 정보통신전공
sclim@dongduk.ac.kr, jsahn@kau.ac.kr

Implementation of a Ranking System for the Web Search Engine based on Inverted Files

Sungchae Lim, Joonseon Ahn
Dongduk Women's University, Korea Aerospace University

요 약

역파일을 사용한 색인 기법은 정보 검색 분야에서 널리 사용되었으며, 최근 대용량 검색 시스템으로 사용되고 있는 웹 검색 엔진에서도 적용되고 있다. 본 논문에서는 웹 검색 엔진의 특성에 맞춰 구현된 역파일 기법 기반의 웹 문서 색인 파일의 구조와 디스크에 저장된 대용량의 역파일 색인을 기반으로 웹 페이지의 검색 적합도를 계산하는 랭킹 시스템을 설명한다. 이를 통하여 상용 웹 검색 엔진의 랭킹 시스템과 디스크 자원 사용의 최소화 기법을 제시한다.

1. 서 론

1994년 미국 라이코스 사가 최초의 상용 웹 검색 포탈 서비스를 시작한 이후로 Infoseek, Inktomi, Altavista, Teoma 등 여러 인터넷 비즈니스 회사에서 자사의 웹 사이트를 통해 웹 검색 서비스를 제공하였다. 이런 초기 서비스를 거쳐 현재는 Google, Yahoo, MSN 등의 포탈에서 제공하는 웹 검색 서비스를 통해 전세계의 다양한 언어로 작성된 웹 페이지를 검색할 수 있다[1,2]. 국내에서는 네이버, 엠파스의 포탈 사이트에서 자사의 웹 검색 엔진을 통한 웹 검색 서비스가 제공되고 있다.

웹 검색 엔진이란 웹 페이지 자동 수집 로봇을 통해 자동 수집한 웹 페이지를 저장한 후에, 사용자의 질의에 부합하는 웹 페이지를 검색 결과로서 제공하는 시스템을 의미한다. 웹 검색 엔진은 한 개 이상의 키워드로 구성된 사용자 질의를 처리하기 위해 자동 수집된 웹 문서에 대해서 색인 파일을 생성하여 두고 주어진 사용자 질의 키워드에 대해서 가장 연관성이 높다고 판단되는 웹 문서의 순으로 검색 결과를 출력한다. 랭킹 시스템은 질의된 키워드를 모두 포함한 웹 문서 중에서 질의 적합도가 높은 순서를 정하기 위한 시스템이며, 웹 검색에서와 같이 색인된 문서의 수가 수천만에서 수십억 페이지를 상회하는 상황에서는 그 중요성이 매우 높다[3,4,5].

웹 검색에서와 같이 대용량의 문서를 색인하고 색인 데이터가 자주 바뀌지 않는 키워드 기반의 질의 시스템에서는 색인 파일 구조를 역파일로 구성하는 것이 매우 효과적인 것으로 알려져 있다[3]. 여기서 역파일이라 함은 문서 내에 출현한 단어, 즉 키워드들을 추출하고 추출된 키워드에 대해서 출현한 문서들을 합병 나열한 색인 파일 구조를 말한다. 문서 내용의 수정이나 문서의 추가 및 삭제가 빈번히 발생할 때에는 동적인 변경이 힘든 역파일의 특성으로 인해 색인 파일로 사용하기 어렵다는 단점이 있다. 하지만 웹 검색 엔진에서와 같이 주기적으로 웹 문서를 수집한 후에 생성한 색인 데이터가 일정 기간 동안에는 갱신 연산 없이 읽기 위주의 접근이 주로 발생하는 환경에서는 색인 파일의 크기를 작게 할 수 있도, 키워드 기반 검색에 효과적인 역 파일이 색인 색인 기법으로 사용된다.

본 논문에서 구현한 검색 엔진은 6,000만 개의 웹 페이지를 색인하여 검색하는 웹 검색 엔진이며, 수집된 웹 페이지의 전체 크기는 HTML 코드를 포함한 상태에서 2 테라바이트를 넘는 매우 큰 규모이다. 색인 작업에서 각 웹 페이지에 존재하는 키워드들이 추출되어 역파일을 생성하며, 역파일에는 질의 적합도 수준을 나타내는 랭크값 계산에 사용하기 위해 문서 내 단어의 오프셋 정보뿐만 아니라 다양한 색인 메타 정보가 함께 저장된다. 랭킹 시스템에서는 조인 연산을 수행하여 질의를 만족하는 웹 문서를 구한 뒤에 랭크값을 계산하여 질의 적합도 순에 따라 사용자에게 보여줄 검색 결과의 순서를 결정하게 된다.

색인 파일의 크기가 매우 크기 때문에 복수개의 서버를 사용한 분산 처리 기법이 필요하다[6,7,8]. 만약 하나의 서버로 동작하는 랭킹 시스템을 사용하여 이러한 대용량의 색인 파일을 처리할 경우에는 원하는 수준의 사용자 질의 응답 시간을 제공할 수 없다. 구현된 시스템에서는 색일 파일을 4개로 파티션하고 이를 독립된 각각의 서버에 저장하여 운영한다. 또한 각 서버에서도 랭크값 계산에 소요되는 디스크 시간을 최소로 하기 위해 분할된 색인 파일을 다시 키워드 별로 12개의 디스크에 분산 저장하는 방식을 사용한다. 이렇게 서버 및 디스크 자원

* 본 논문은 산업자원부 한국산업기술평가원 지정 한국항공대학교 부설 인터넷정보검색 연구센터의 지원에 의함.

병렬화를 통해 하나의 질의 처리에 소모되는 최대 디스크 시간을 줄이는 한편, 생성하는 역파일을 계층적 구조로 만들어 디스크 읽기 비용을 줄이는 기법을 사용하였다.

본 논문에서 소개하는 웹 검색 엔진을 위한 역파일 구조의 색인 파일과 이를 이용한 랭킹 시스템을 통해 대용량의 데이터에 대해 키워드 질의를 수행해야 할 때 고려해야할 기술적인 제약 사항 및 적용된 기술을 제시한다. 이어지는 논문의 구성은 다음과 같다. 2장에서는 구현된 계층적 구조의 역파일 색인 구조에 대해서 기술한다. 3장에서는 구현된 랭킹 시스템의 전체적인 구조 및 랭크값 계산에 사용하는 알고리즘에 대해서 소개한다. 그리고 4장에서 결론을 맺는다.

2. 계층적 역파일 구조

사용자 질의 처리 과정은 질의를 이루는 모든 키워드를 포함한 웹 페이지를 찾는 조인 연산과 이들에 대한 질의 적합도 계산 과정으로 이루어진다. 질의 적합도 계산에 따라 질의 연관성이 높은 웹 페이지가 검색 결과의 상위에 노출되며 하위의 결과들은 검색 결과 페이지 내에 보여지는 링크 연결을 따라 접근된다. 이러 질의 처리 과정의 효율을 위해서는 수집한 웹 문서에 대해서 색인 파일을 구성하는 것이 일반적이다.

웹 검색의 색인 기법과 기존 DBMS의 색인 기법에는 요구사항에 있어 다소 차이가 있다. 기존 DBMS에서는 저장된 레코드의 삭제 및 수정이 빈번히 발생할 수 있기 때문에 색인 기법 역시 이런 갱신 연산에 효과적이어야 한다. 또한, 범위 검색이나 like 검색 등과 같이 단순 키워드 매칭 이외의 검색 연산이 지원되어야 한다. 하지만 웹 검색에서는 단순 키워드 매칭이 전부이고, 갱신 연산이 거의 발생하지 않는 것이 특징이다. 즉, 다음 데이터 갱신 시점 까지는 갱신 없이 사용되기 때문에 검색 속도가 빠르고 색인 파일 크기가 상대적으로 작은 역파일 구조가 웹 검색에 사용된다. 구현한 웹 검색 시스템 역시 역파일을 색인 파일의 기본 구조로 사용하며, 역파일 안에는 색인 단어의 오프셋(문서 내 위치) 정보 및 다양한 색인 메타 정보가 기록된다.

구현된 시스템에서 사용한 역파일은 그림 1과 같은 구조를 가진다. 색인 파일은 조인 연산 및 질의 적합도 계산의 효율성을 위해 3단계로 분리되어 구성되며, 가장 상위에 키워드 디렉터리가 메모리에 상주하며, 그 아래 계층으로 키워드 별로 출현한 문서의 DID(Document ID)와 관련 세부 색인 데이터의 저장 위치를 기록한 DID 리스트 파일이 존재한다. 가장 아래 계층에는 각 문서 내에서 해당 키워드가 위치한 오프셋 정보 및 색인 메타 정보를 담은 IDX 파일이 생성된다. DID 리스트 파일과 IDX 파일은 디스크에 존재함을 가정한다.

키워드 디렉터리에는 <키워드 DID 리스트 주소 엔트리가 키워드를 정렬 필드로 하여 저장되며, DID 리스트 주소는 해당 키워드를 포함한 웹 문서의 DID 모음(list)에 대한 주소 정보이다. 이런 DID 모음을 DID 리스트라고 부르며 DID 리스트는 다시 DID, 관련IDX 파일 주소 엔트리로 구성된다DID 리스트는 빠른 조인 연산을 위해 DID 순으로 정렬되어 있다. 주어진 키워드 k에 대한 DID 리스트를 찾고자 한다면 먼저 메모리

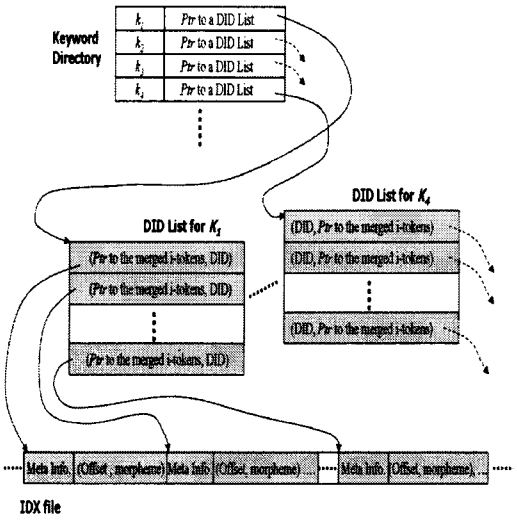

그림 1. 색인 파일의 3단계 구조.

에 적재된 키워드 디렉터리에서 k를 검색하여 해당 엔트리를 찾고, 여기에 저장되어 있는 DID 리스트 주소를 사용하면 된다.

가장 하위에 존재하는 IDX 파일은 가장 큰 파일 크기를 가지며 이곳에 오프셋 데이터 및 색인 메타 데이터가 저장된다. 이런 색인 정보는 각 웹 문서로부터 추출된 i-token(index token)들을 합병하여 생성한다. 문서 내의 각 키워드 출현 시마다 하나의 i-token이 생성되며, i-token에는 단어 위치 정보, 형태소 분석 정보, 폰트 크기나 종료, 타이틀 단어인지의 유무, URL에 나오는 단어인지의 유무, 앵커 텍스트 단어인지의 유무, HTML 태그 정보 등이 포함되는데, 이중에서 단어 위치 정보 및 형태소 분석 정보는 오프셋 정보라 칭한다. 나머지 정보에 대한 데이터는 색인 메타 정보라 칭한다. 이런 i-token 들이 문서 별로 병합되고, 다시 키워드 별로 병합되어 IDX 파일에 저장된다.

색인 파일을 세 계층으로 분리함으로써 사용자 질의 처리 시에 디스크 I/O의 사용을 크게 줄일 수 있다. 우선 크기가 작은 키워드 디렉터리는 메모리에 적재하여 둠으로써 디스크 I/O를 없앤다. 조인 연산 시에는 각 키워드 별로 DID 리스트 만을 읽은 후, 조인 연산을 통해 질의 적합도 계산을 수행할 DID들을 선택한다. 조인 연산에 의해 크게 줄어든 DID에 대해서만 관련된 IDX 파일을 읽어 들임으로써 불필요한 IDX 파일 읽기를 피할 수 있다.

3. 구현한 랭킹 시스템

본 장에서는 앞서 기술한 색인 파일을 이용한 랭킹 시스템에 대해 설명한다. 구현한 랭킹 시스템은 하나 이상의 질의어로 구성된 사용자 질의를 실시간으로 처리하며, 질의 적합도가 높은 순서로 검색 결과를 생성한다.

3.1 랭킹 시스템 개요

앞서 설명하였듯이 하나의 색인 파일은 4개의 서버로 분할

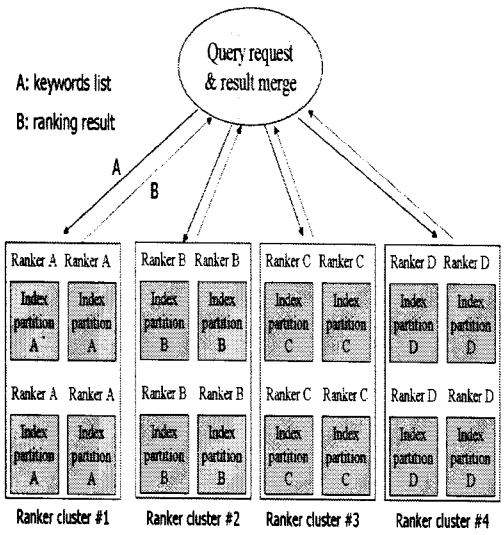

그림 2. 랭킹 시스템의 랭커 클러스터와 서버.

저장된다. 각 서버에는 사용자 질의어에 대해 조인 연산과 랭킹 작업을 수행하는 랭커 서버가 존재한다. 이런 랭커 서버를 클러스터링 함으로써 랭킹 시스템을 구성한다. 그림 2는 구현된 랭킹 시스템에서 사용한 클러스터의 모습이다.

그림 2에서 보이는 색인 분할 A에서 D까지는 네개의 서버로 분할된 색인 파일들을 나타낸다. 하나의 질의에 대해 이런 분할된 색인 파일을 이용하여 4개의 랭커 서버가 동시에 랭킹 작업을 수행하며, 랭킹 작업을 통해서 Rank Level, DID> 의 쌍으로 구성된 랭크값 리스트가 작성되어 결과 값으로 전달된다. 이들 각 랭커 서버에 의해 작성된 랭크값 리스트는 하나의 서버에서 수집되며, 수집된 4개의 랭크값 리스트를 랭크값 순으로 합병한다. 이렇게 합쳐진 검색 결과를 이용하여 상위 랭크의 DID들이 검색 결과 페이지의 상위 페이지에 노출된다.

구현한 웹 검색 엔진의 작업 용량을 늘이기 위해 동일한 IDX 파티션을 가지는 랭커 서버 4개를 동일한 서버 클러스터로 묶어 구성한다. 작업 분배자의 입장에서는 이런 4개의 랭커 클러스터에서 각기 한 개의 랭커 서버를 선택하고, 선택된 4개의 서버로 동시에 동일한 질의어를 보내고 결과를 합병하고 정렬함으로써 검색 결과를 출력한다. 이런 서버 클러스터링을 통해 임의의 랭커 서버에서 발생할 수 있는 하드웨어 고장에 대비한다. 즉, 특정 랭커 서버에 고장이 발생하는 경우, 동일한 랭커 클러스터 내의 다른 서버들에 질의를 보냄으로써 고장이 발생하는 경우에도 랭킹 작업이 중단되지 않는다.

랭커 서버에 의한 사용자 질의 처리는 다음의 단계에 따라 진행되며, 다음 처리 단계를 통해 진행되는 사용자 질의는 (*k1*, *k2*, ..., *kn*)의 1개 이상의 키워드로 구성됨을 가정한다.

① 사용자 질의에 속한 키워드 *ki*에 해당하는 키워드 디 렉터리 엔트리를 찾기 위해 키워드 디렉터리를 이진 검색함. 검색된 엔트리에서 DID 리스트에 대한 주소 정보를 획득.

② 주소 정보를 사용하여 모든 키워드 *ki*에 대한 DID 리스트를 메모리로 읽어 들임. 읽어 들인 모든 키워드들의 DID 리스트를 순차적으로 스캔하며 조인 연산 수행. 조인 연산을 만족하는 DID 만을 남겨둠.

③ 조인 연산으로 구한 키워드 별 DID 리스트를 이용하여 해당 문서를 위한 색인 메타 정보와 오프셋 데이터를 읽어 들임.

④ 색인 정보에 따라 랭킹 알고리즘을 적용하여 질의 적합도를 계산하고 이를 랭크 값으로 출력함.

⑤ 모든 DID에 대해 랭크 값을 출력한 후에 랭크값에 의거하여 내림차순으로 소팅된 Rank Level, DID> 쌍을 검색 결과 리스트로 보냄.

위의 과정 1에서 사용되는 키워드 디렉터리는 메모리에 상주하는 색인 데이터이며, 나머지 데이터는 디스크 저장 데이터이다. 이들 디스크 데이터들은 그 사이즈가 크기 때문에 일부 데이터에 대해서는 메모리에 캐시하여 사용한다. 이런 캐쉬 기법이 적용되는 데이터는 DID 리스트 데이터이며 IDX 데이터의 경우 그 특성상 캐시 적중률이 매우 낮기 때문에 캐시 기법을 적용하지 않는다.

그림 3은 구현된 랭킹 시스템에서 사용하는 DID 리스트 캐시 기법을 간략히 보이는 구성도이다. 그림을 통해 보이듯이 DID 리스트를 캐쉬할 수 있는 메모리 공간은 크게 두 종류로 분리된다. 하나는 랭커 프로세스와 같은 응용 프로그램이 사용하는 메모리 공간으로서 초기 프로세스가 생성되면서 확보하는 메모리 공간이다. 다른 하나는 운영체제 내에서 관리되는 시스템 버퍼 공간으로 커널 메모리 공간에 위치한다. 구현된 시스템은 Windows 2000 서버를 운영체제로 하기에, Windows의 파일 시스템에서 관리되는 커널 버퍼링 공간을 하나의 캐시 공간으로 가정하여 구현된다. 서로 다른 두 메모리 공간이 캐시 공간으로 사용되며 랭커 서버내의 공간에 생성되는 캐시를 Level 1, 커널 공간에 생성되는 캐시는 Level 2 캐시라 부른다.

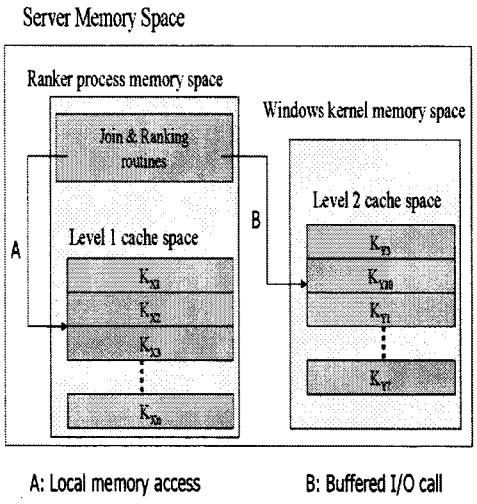

그림 3. 2계층 구조의 DID 리스트 캐시.

Level 1 공간에 저장되는 키워드는 웹 검색 엔진에 입력된 사용자의 질의어 로그에 따른 매우 자주 입력되는 키워드 들이다. 커널 공간에 위치하는 레벨 2 캐쉬의 경우, 메모리 공간의 점유 및 해제는 랭커 프로세스가 직접 관여하지는 않는다. 이 공간의 캐시 데이터는 운영체제의 버퍼 메모리 관리 알고리즘에 의해 간접적으로 관리되며 이곳으로의 캐시 데이터를 올리기 위해 I/O 콜을 호출한다. 레벨 2에 기록될 수 있는 DID 리스트는 Level 1 캐시에 저장되는 키워드에 비해 보다 참조 확률이 낮은 키워드 들이다. Level 1 캐시에 위치한 키워드 집합은 고정적이며 Level 2의 경우는 메모리 상황에 따라 동적으로 변할 수 있다.

Level 1과 Level 2에 저장되는 키워드 외의 키워드에 대해서는 캐시를 적용하지 않는다. 이를 위해 비 캐쉬 키워드에 대한 DID 리스트를 메모리로 읽을 때는 Windows 운영체제에서 지원하는 Non-Buffered I/O 콜을 사용한다. 이 I/O 콜을 통해 읽혀지는 디스크 데이터는 커널의 메모리 버퍼 영역으로 복사되지 않고, 함수를 호출한 응용 프로세스의 지정된 메모리로 직접 읽혀진다. 이런 방식을 통해 Level 2 캐시에 원하지 않는 키워드에 대한 DID 리스트가 기록되는 것을 막는다.

Non-Buffered I/O 콜에 의한 디스크 데이터 읽기는 모든 IDX 파일에 속한 데이터를 읽을 때 일괄적으로 적용된다. IDX 파일의 경우 조인 연산에 의해 선택되는 DID에 의해, 읽혀지는 데이터 위치가 매우 가변적이기 때문에 캐시 기법을 통한 성능 향상을 기대하기 어렵다. 이런 이유로 IDX 파일을 읽을 때는 항상 Non-Buffered I/O를 사용한다.

이처럼 랭커 서버가 조인 연산 및 랭킹 작업을 수행할 때 커널의 버퍼링 기능을 사용하지 않음으로써 두 가지 성능 향상을 기대할 수 있다. 첫째는 불필요한 메모리 사용 증가를 막을 수 있다는 점이다. 재차 사용되지 않은 IDX 파일이나 DID 리스트를 많이 읽어 들일 경우 커널의 버퍼 메모리가 증가하고 이를 통해 랭커 프로세스가 사용할 수 있는 메모리가 빠르게 감소할 수 있다. 둘째는, 커널 메모리로 복사되는 과정을 거치지 않음으로써 CPU 사용을 줄일 수 있다는 점이다.

실제 구현 시스템이 이런 Non-Buffered I/O 콜을 사용하지 않는 경우 처리할 질의수가 증가할 때 랭커 서버의 가용 메모리가 급속도로 감소하여, 시스템에 thrashing 현상이 발생하는 것을 관찰할 수 있었다. 이를 막기 위해 색인 데이터의 많은 부분을 Non-Buffered I/O 방식으로 읽어 들였으며 이를 통해 성능 향상 및 성능 안정성을 높일 수 있었다.

3.2 랭킹 알고리즘

조인 연산을 통해 질의 적합도를 계산할 DID들이 결정되며, 각 키워드 별로 IDX 파일에 위치 정보가 준비된다. 이후에는 랭킹 알고리즘을 적용하여 각 DID 별로 랭크 값을 정해주어야 하는데 이를 위해서는 IDX 파일에 저장된 색인 메타 정보 및 키워드의 문서 내 위치 정보인 오프셋 데이터가 읽혀져야 한다. 사용되는 오프셋 데이터는 2바이트 데이터 공간 중에 상위 14 비트를 사용하며, 나머지 2비트는 형태소(morpheme) 정보를 저장하는데 사용한다. 메타 정보로는 앞서 2장에서 나열한 정보들이 기록된다.

그림 4는 구현된 랭킹 알고리즘을 설명하는 의사 코드이다. 알고리즘은 조인 연산이 끝난 후에 수행되며 알고리즘의 수행 결과는 계산된 랭크값과 해당 DID 값을 하나의 원소로 갖는 RankLevels에 저장되어 반환된다. 먼저 키워드의 오프셋 정보를 이용하여 문서에 나타난 키워드들 간의 거리 계산이 수행되며, 여기서 얻은 정보는 알고리즘의 라인 (8)의 랭크 계산에 사용된다. 색인 메타 정보를 이용한 랭크 계산은 라인 (9)에서 수행되며, 이 두 개의 랭크값들은 이후 라인 (11)에서 최종 랭크값을 구하는데 입력값으로 사용된다. 이때 최종 계산에는 링크 분석을 통해 미리 구해놓은 웹 페이지의 중요도 값인 PageWeight 값이 함께 사용된다. 최종 랭크값은 해당 페이지의 DID 값과 함께 RankLevels의 원소로 저장된다. 알고리즘의 랭크값 계산 및 인접도 계산 함수들은 공간의 제약 상 세부 알고리즘은 기술하진 않는다.

구현된 랭킹 코드는 질의에 속한 키워드 개수에 따라 오프셋 데이터에 대한 가중치를 크게 변화시킨다. 예를 들어 2개의 키워드로 구성된 질의어 Q 과4 개의 키워드로 구성된 질의어 Q 가 있다고 가정해 보자Q 에 대한 랭킹 계산 시에는 두 질의어 간에 단어 인접도는 Q 에 대한 랭킹 계산에 비해서 그 중요성이 떨어진다. 2개의 단어가 인접하는 경우에 비해 4개의 단어가 인접할 때가 문서의 질의 연관성을 상대적으로 강하게 암시한다. Q 과 같은 경우 키워드의 빈도Url 에 나오는 지의 여부, 타이틀 문장에 속하는 지의 여부, 사용한 폰트의 특징이 단어 인접도에 비해 그 중요도가 높을 수 있다. 이에 비해 Q 의 경우에는 다른 어떤 요소에 비해서 단어 인접도가 질의 연관성에 큰 영향을 미침을 알 수 있었다.

3.3 디스크 시간 최소화

웹 검색 엔진의 색인 파일은 그 크기가 매우 큰 것이 특징이다. 만약 하나의 랭커 프로세스에서 하나의 사용자 질의에 관련된 모든 색인 데이터를 읽는다면 대부분의 질의가 너무 긴 처리 시간을 요구하게 된다. 이런 점을 해결하고자 구현한 시스템은 4대의 서버로 색인 파일을 분할하였고 하나의 서버 내에서도 키워드에 대해 분할 정책을 사용하여 12개의 물리적으로 독립된 디스크에 분산 시키는 방식을 취했다.

이처럼 병렬적인 하드웨어 구성을 통한 디스크 시간 최소화 기법 외에도 앞서 언급한 DID 리스트의 캐싱을 적용하였다. 이를 통해 최고 빈도로 입력되는 키워드들은 Level 1 캐시에, 이보다 다소 적은 빈도의 일부 키워드들은 Level 2 캐시에 저장되도록 하여 가급적 메모리로부터 해당 DID 리스트 정보를 읽는 방식을 사용하였다. 이외의 키워드에 대해서는 디스크로부터 사용자 메모리로 직접 읽혀지도록 하여 I/O 콜에 블록되는 시간을 줄이는 방식을 사용하였다.

이외의 방식으로는 색인 파일을 압축함으로써 디스크 시간을 줄일 수 있다. 하지만 이 경우 압축 해제를 위해서 CPU 시간이 필요하기 때문에 이 둘 사이의 trade-off를 적절히 고려해야 한다. 또한 랭크값 계산 시에 IDX 파일의 경우는 전체 데이터가 처음부터 순차적으로 읽혀지는 것이 아니고 임의 접근에

Algorithm EvaluateRankLevel()

INPUT: NoJoinResults /*number of DIDs satisfying the join condition */
NoKeywords /*number of keywords contained in the user query */
JoinDID[NoKeywords][NoJoinResults] /*DID lists resultin g from the join operation */

OUTPUT: RankLevels[NoJoinResults] /*rank lev els evaluated for each DID in JoinDID[][] */

(1) count =1.
(2) **While** [count ≤ NoJoinResults] **Do**
(3) **For** [each *k* such that 1 ≤ *k* ≤ NoKeywords] **Do**
(4) Determine the disk address to the data portion of the IDX file for *k*-th keyword using JoinDID[*k*][count], and save those address into the variable, IdxAddress[*k*].
(5) Fetch the meta data and offset data pointed to by IdxAddress[*k*] into the variables MetaInfo[*k*] and Offsets[k], respectively.
(6) **EndFor**
(7) Call *EvalProximity(NoKeywords, Offset[], ProximityInfo)* to evaluate the proximities among the keyword occurrences in a web page and save them in ProximityInfo.
(8) Call *Rank1 = CalcRankPart1(ProxmityInfo, NoKeywords)* to calculate the rank level using the proximity information among the keyword occurrences.
(9) Call *Rank2 = CalcRankPart2(MetaInfo[], NoKeywords)* to calculate the rank lever using the index meta information of the keywords.
(10) Save MetaInfo[1].PageWeight to variable PageWeight, which is calculated based on the hyperlink analysis on all the crawled Web pages.
(11) Finally, calculate the rank level of the *count*-th DID by calling *RankLevel = CalcRankMixed(NoKeywords, PageWeight, Rank1, Rank2).*
(12) Save the current DID and RankLevel values into fields RankLevels[count].DID and RankLevels[count].Rank, respectively.
(13) count ← count +1.
(14) **EndWhile**

그림 4. 랭킹 알고리즘 EvaluateRankLevel.

의하여 읽혀지므로 압축 기법을 사용하는 것이 용이하지 않으며 압축을 통한 성능 향상을 거의 기대할 수 없다.

이 보다는 DID 리스트에 대해서 압축 기법을 적용하는 것이 바람직한데, DID 리스트는 조인 연산시에 전체가 읽혀져야 하며 특히 DID 리스트 데이터는 DID 순으로 정렬되어 있기 때문에 DID 값의 차이를 코딩하는 방식으로 데이터를 간단히 압축할 수 있기 때문이다. 또한 조인 연산을 수행하는 과정에서도 조인 조건을 만족시키는 DID에 대해서만 압축 해제된 값을 저장함으로써 압축 해제 시에 요구되는 추가 메모리 확보도 쉽게 할 수 있다. 계산에 의하면 DID 리스트 압축을 통해 DID 정보 기록을 위한 공간은 50% 이상 줄일 수 있으며 이를 통한 성능 향상 효과를 기대할 수 있다.

4. 결론

본 논문에서는 실제 구현된 웹 검색 엔진에서의 계층적 역파일 구조에 대해 기술하였다. 계층적 구조를 사용한 것은 가장 접근 빈도가 높고 상대적으로 크기가 작은 키워드 디렉터리 부분은 메모리에 상주시키고 나머지 부분은 필요에 따라 디스크 I/O를 통해 선택적으로 읽어 들일 수 있게 하기 위해서였다. 오프셋 데이터는 최하위 계층에 기록하여 둠으로써 조인 연산이 수행된 후에 메모리로 읽혀지게 하여 가능한 불필요한 디스크 요청을 없애고자 했다. 또한 DID 리스트의 경우에는 캐시 기법을 적용하여 사용자 질의에 나타나는 키워드의 빈도수에 의거하여 2계층 캐시 메모리를 사용할 수 있도록 구현하였다.

랭킹 계산을 수행하는 랭킹 시스템은 4개의 랭커 클러스터와, 각 클러스터 내에서는 4개의 랭커 서버로 구성된다. 각 랭커 서버는 계발된 랭킹 알고리즘에 따라 조인 연산이 수행된 DID에 대해서 랭킹 값을 계산하고 이들 값들은 이후 검색 적합도가 높은 DID를 우선으로 소팅되어 사용되었다. 본 연구를 통해 상용 웹 검색 엔진에서의 색인 파일 구성 및 랭킹 계산에 사용되는 기술 및 알고리즘에 대해서 보다 깊이 이해할 수 있을 것이라 생각한다.

[참고문헌]

[1] 이주남, Google과 함께 떠오르는 검색엔진, 소프트웨어진흥원 시장이슈 보고서, 2004.

[2] Search Engine Report, http://www.searchenginewatch.com , 2005.

[3] Sergey Brin and Lawrence Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. Computer Networks and ISDN Systems, 30(1-7), pp. 107-117.

[4] C. Lee, G. Golub and S. Zenios. A Fast Two-Stage Algorithm for Computing PageRank, Technical report, Stanford University, 2003.

[5] Taher H. Haveliwala. Topic-sensitive PageRank, In Proc. of the 11th International World Wide Web Conference, 2002.

[6] Sergey Melnik, Sriram Raghavan, Beverly Yang, and Hector Garcia-Molina. Building a Distributed Full-text Index for the Web, In Proc. of the Tenth International World Wide Web Conference. pp. 396-406, 2001.

[7] Maxim Lifantsev and Tzi-Cker Chiueh. Implementation of a Modern Web Search Engine Cluster. http://citeseer.ist.psu.edu/561246.html

[8] Luiz Andre Barroso, Jeffrey Dean, and Urs Holzle. Web Search for a Planet: The Google Cluster Architecture, IEEE Micro, 23(2), pp. 22-28, 2003.

[9] Soumen Chakrabarti, Kunal Punera, and Mallela Subramanyam., Accelerated focused crawling through online relevance feedback. In Proc. of the 11th international conference on World Wide Web, pp. 148-159, 2002.

[10] Sriram Raghvan and Hector Garcia-Molina. Crawling the Hidden Web. In Proc. of VLDB, pp. 129-138, 2001.

[11] Krishna Bharat, Andrei Z. Broder, Jeffrey Dean, and Monika Rauch Henzinger. A comparison of techniques to find mirrored hosts on the WWW. Journal of the American Society of Information Science. 51(12), pp.1114-1122, 2000.