대용량 데이터 색인에 적합한 역파일의 구현

임 성 채
동덕여자대학교 정보대학 컴퓨터전공
e-mail:sclim@dongduk.ac.kr

Implementation of the Inverted File for Indexing Large-volume Data

Sung Chae Lim
Dept. of Computer Science, Dongduk Women's University

요 약

대용량 문서에 대한 키워드 검색을 위해 역파일(inverted-file) 색인 기법이 널리 쓰이고 있다. 역파일 색인 기법을 구현함에 있어 고려되어야 할 점은 키워드 검색 처리 시에 디스크 사용을 최소로 할 수 있는 방법이다. 크기가 작은 역파일이라면 디스크 I/O 사용도 작고 필요시 역파일을 메모리에 적재하여 둠으로써 디스크 사용을 크게 줄일 수 있다. 하지만, 웹 검색이나 규모가 큰 도서관 시스템에서와 같이 색인 데이터 크기가 매우 큰 경우 역파일을 읽는 디스크 비용이 급격히 증가할 수 있다. 본 논문에서는 매우 큰 크기의 역파일을 사용하는 검색 환경에서 디스크 사용을 최소로 할 수 있는 역파일 구조를 제안한다. 제안된 구조는 질의 처리 과정을 고려해 계층 구조로 설계되며 실제 상용 시스템에 적용되어 안정성 및 성능을 입증했다.

1. 서론

역파일 구조의 색인 기법에서는 검색 대상이 되는 문서에서 키워드로 사용할 수 있는 단어들을 모두 추출해 내고, 추출된 각 키워드에 대해 키워드가 출현한 문서 식별자 정보 및 문서 내 키워드 위치 정보를 저장하여 사용한다. 역파일을 사용한 키워드 검색 시스템은 개인용 PC나 소규모 회사에서 널리 쓰이고 있으며, 역파일의 크기가 작은 경우에는 적절한 크기의 디스크 사용 및 메모리 자원 이용만으로도 효과적으로 동작할 수 있다[1]. 하지만 색인하는 문서 수에 비례하여 그 크기가 빠르게 커지는 역파일의 특성으로 인해 많은 수의 문서가 색인되어야 하는 환경에서는 과도한 디스크 사용량으로 인해 질의 처리 시스템 전체의 성능에 큰 문제를 야기할 수 있다[2]. 예를 들어 인터넷 상의 웹 문서 검색 시스템, 규모가 큰 도서관의 전문(full text) 검색 시스템, 특허 전문 검색 시스템이 이에 속할 수 있다.

본 논문에서는 이렇게 색인되는 문서의 수가 매우 많은 경우에 문제가 될 수 있는 역파일 색인 성능을 개선하기 위한 방법을 소개한다. 본 연구에서는 키워드 검색이 처리됨에 있어 수행되는 2단계 과정인 키워드 조인(join) 단계와 랭킹 계산 단계를 고려하여 이에 맞도록 역파일 구조 역시 계층화하여 설계한다. 이런 계층화를 통해 얻을 수 있는 장점은 키워드 조인 단계에서 사용되는 색인 파일과 랭킹 계산 단계에서 사용되는 색인 파일을 분리함으로써 불필요한 색인 파일 읽기를 최소화 할 수 있다는 점이다. 이와 함께 색인 파일의 일부를 메모리에 캐싱하는 기법을 제안한다. 이때 캐시 데이터 역시 계층성을 가지며, 메모리 캐시가 가지는 계층성은 키워드의 사용자 질의 내 출현 빈도 및 색인 크기에 의존한다. 이런 계층적이고 메모리 캐시를 고려한 역파일 구조를 통해 질의 처리 시에 요구되는 디스크 읽기 비용을 최소화 시킬 수 있었다.

본 논문의 구성은 다음과 같다. 2장에서는 역파일을 사용한 키워드 검색 일반에 대해서 기술한다. 3장에서는 고안한 역파일의 계층적 구조에 대해서 기술한다. 4장에서는 디스크 사용을 줄이기 위해 적용된 메모리 캐시 기법을 소개한다. 그리고 5장에서 결론을 맺는다.

2. 역파일을 이용한 키워드 검색

키워드를 이용한 문서 검색 시스템에서는 검색엔진 사용자가 자신이 찾고자 하는 문서의 특성을 잘 반영한다고 생각한 키워드를 하나 이상 입력하면 입력된 질의에 대해 가장 관련성이 높다고 판단되는 문서 순으로 검색 결과가 반환되고 이 결과를 사용자가 확인함으로써 질의가 처리된다. 이런 응용을 위해 역파일을 사용한 색인 기법이 널리 쓰이고 있다[2,3]. 역파일은 색인될 문서에서 사용자 질의에 사용될 수 있는 키워드들을 추출한 후 키워드 별로 추출된 문서의 식별자들을 리스트 형태로 기록한 파일이다[5]. 즉, 각 키워드가 어떤 문서에 출현했는지를 색인한 파일이다. 역파일의 문서 식별자 정보를 이용하여 입력된 질의어에 포함된 키워드들을 모두 포함하고 있는 문서 집합을 구함으로써 사용자 질의를 만족시키는 문서 집합을 결정할 수 있다. 이런 과정이 키워드 조인이라 할 수 있다. 논문에서는 모든 질의가 AND 질의임을 가정하고 이는 웹 검색과 같이 대용량 문서 검색이 가지는 특성을 반

★ 본 논문은 지식경제부 한국산업기술평가원 지정 한국항공대학교 부설 인터넷정보검색 연구센터의 지원에 의함.

영한 것이다.

역파일에 단지 문서 식별자 정보만이 존재한다면 문서의 질의 적합도에 따라 계산되는 랭크 값을 구할 수 없을 것이다. 따라서 질의에 대한 문서 연관성을 평가하는 랭킹 작업에 사용할 색인 정보가 역파일에 저장되어야 한다. 이런 색인 정보로서 가장 일반적인 것이 키워드들이 문서 내 위치(offset) 정보이다. 사용자 질의를 구성하는 키워들의 문서 내에서 어떤 인접도를 가지며, 어떤 패턴으로 출현하는 지는 질의와 문서간의 연관성을 평가하는데 있어 매우 중요한 요소로 작용한다.

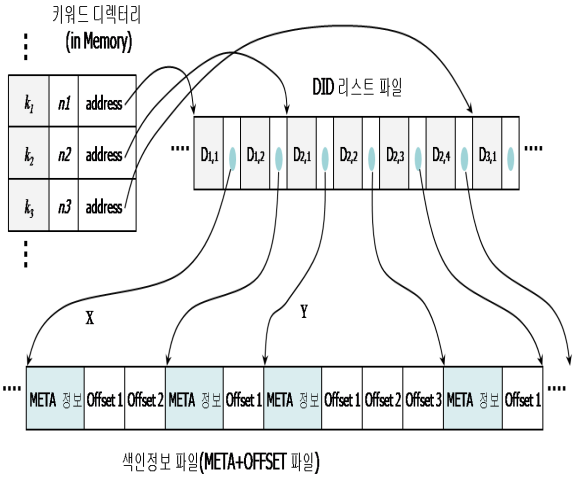
이 때 한국어의 경우는 이런 키워드의 위치 정보 저장 시에 형태소(morpheme) 분석 정보도 함께 저장되는 것이 보통이다. 이런 형태소 분석 정보를 저장하기 위해 키워드 위치를 표시하는 바이트 중 일부 비트를 사용한다. 논문에서는 키워드 위치 정보와 형태소 분석 정보를 합쳐 OFFSET 정보라 부른다. 이런 OFFSET 정보는 키워드가 특정 문서에 출현할 때 마다 생성되므로, 문서에 자주 출현하는 키워드는 OFFSET 정보 저장을 위해 보다 많은 디스크 공간을 필요로 한다.

OFFSET 정보 외에도 역파일에는 키워드의 문서 내 중요도를 나타내는 정보도 함께 저장된다. 예를 들어 키워드 *k*가 출현한 문서가 웹 문서일 때, 키워드 *k*가 해당 웹 문서의 제목에 있거나, 이탤릭체 혹은 큰 사이즈의 폰트로 표현되었다면 그렇지 않은 보통의 키워드에 비해 그 중요도가 클 확률이 높다고 볼 수 있다. 이외에도 해당 문서를 가리키는 외부 앵커 텍스트 내의 존재 유무, 문서 내 앵커 텍스트에서의 존재 유무, URL 내의 존재 유무 및 URL 내의 위치 등은 랭킹 계산에 있어 중요 정보이다. 이와 함께 각 문서의 문서중요도 역시 역파일에 저장된다. 문서중요도란 Google의 PageRank[4] 정보와 동일한 성격의 정보이다. 이런 정보들을 모두 합쳐 논문에서 색인 META 정보라 부른다.

OFFSET 정보와 META 정보는 모두 랭킹 작업에 사용되는 색인 정보로서 이 데이터는 문서 식별자 정보를 이용해 질의어에 부합(matching)하는 문서 집합을 선택하는 단계인 키워드 조인 연산 이후에 읽혀진다. 키워드 조인은 입력된 키워드를 모두 포함한 문서 집합을 결정하는 과정을 의미한다. 또한, 랭킹 작업은 키워드에 부합하는 문서 집합이 결정된 후 이들 간에 질의 분합도 순서를 평가하는 과정이며, 이 때 OFFSET 정보와 META 색인 정보가 필요하다. 이런 두 단계의 질의 처리 과정에서 가급적 역파일을 적게 읽어 들이는 것이 좋으며 이를 고려한 역파일 구조가 요구된다.

3. 고안된 역파일 구조

주어진 사용자 질의 Q에 대한 랭킹 계산 과정을 자세히 살펴보기로 한다. 이를 위해 질의 Q가 키워드 *k1*과 *k2*로 구성된다고 가정한다. 질의 처리 1단계인 키워드 조인 단계에서는 *k1*과 *k2*가 출현한 문서들의 식별자 리스트를 각각 읽은 후에 두 개 키워드가 모두 출현한 문서를 찾기 위해 동일조인(equi-join)을 수행된다. 동일조인의 결과로 질의 Q에 부합하는 문서들의 식별자 리스트가 구해진다.



(그림 1) 3계층 구조의 역파일.

다음에 이어서 2단계 작업으로 이들 문서 집합에 대한 질의 연관성의 상대적인 크기를 결정하는 랭킹 작업이 수행된다. 즉, 얼마나 자주 문서에 출현하고 있는 지, *k1*과 *k2*의 문서 내 인접 빈도수 등이 계산된다. 이와 함께 HTML 문서의 특성을 반영한 여러 정보들도 질의 연관성 계산에 함께 고려된다. 이런 랭킹 연산에는 앞서 언급한 OFFSET 및 색인 META 정보가 사용된다.

키워드 조인 단계에서 *k1*과 *k2*의 문서 식별자 리스트가 모두 읽혀지고, 동등조인을 빠르게 수행하기 위해 문서 식별자 값은 미리 정렬되어 저장된다. OFFSET+META 데이터는 문서 식별자 데이터에 비해 상대적으로 그 크기가 매우 크고, 질의에 부합하는 문서들에 대해서만 관련 데이터를 읽으면 되기 때문에 전체 데이터를 다 읽어 들이는 방식이 아닌 부분적인 읽기가 용이하게 구성되어야 한다.

문서 식별자 데이터와 OFFSET+META 데이터의 서로 다른 읽기 특성을 고려해서 3단계 계층구조의 역파일을 사용하며 이에 대한 개략적인 모습을 (그림 1)에서 보인다. 그림에서 가장 상위 계층에는 키워드 디렉터리가 있으며, 각 키워드 디렉터리 엔트리에는 키워드 별로 문서 식별자 리스트의 디스크 위치 정보 및 키워드가 출현한 전체 무서의 개수가 기록된다. 이런 키워드 디렉터리 엔트리들은 키워드에 대해 정렬되어 저장되며 초기 시점에는 디스크에 저장되어 있다가 이후 검색 엔진의 메모리로 적재되어 상주한다. 다음 계층인 문서 식별자 리스트 데이터는 이런 키워드 디렉터리에 의해 접근되며 리스트 내의 각 엔트리에는 출현한 문서의 식별자 값과 해당 문서에서의 세부 색인 정보를 담고 있는 파일의 디스크 주소가 저장된다. 즉, OFFSET+META 데이터의 디스크 위치가 저장된다. (그림 1)의 예에서 *k1*이 출현한 문서 식별자는 $\{D_{1,1}, D_{1,2}\}$이고, *k2*의 경우는 $\{D_{2,1}, D_{2,2}, D_{2,3}, D_{2,4}\}$의 네 개이다. (그림 1)에서는 화살표 X는 키워드 *k1*에 대한 META+OFFSET 데이터 위치 정보를, 화살표 Y는 *k2*에

대한 위치 정보를 나타낸다.

META+OFFSET 데이터는 (그림 1)에서 보이듯이 가장 하부 계층으로 존재한다. 각 문서에 대해서 해당 키워드가 갖는 META 색인 정보는 고정된 크기의 데이터 구조체로 정의되며 이런 META 색인 정보를 담은 데이터 구조체에 바로 이어서 OFFSET 데이터가 저장되어 있다. META 색인 데이터와는 다르게 OFFSET 데이터는 문서 내 키워드 출현 회수가 많아지면 그 크기도 함께 증가한다. META+OFFSET 데이터도 문서 식별자 데이터와 동일하게 디스크에 물리적으로 연속적으로 저장되며, 연관된 문서 식별자 순서로 데이터가 저장된다.

*k1*과 *k2*의 키워드 조인에 있어, 만약 $D_{1,1} = D_{2,1}$이고 $D_{1,2} = D_{2,4}$이라면, 질의에 부합하는 문서 수는 두 개가 될 것이다. 또한 이런 두 문서의 랭킹 값을 계산하기 위해 META+OFFSET 데이터를 읽기 위해서는 전체 4회의 디스크 탐색 지연이 발생할 수 있다. 이처럼 IDX 파일은 선택적인 데이터 접근이 요구되기 때문에 디스크 탐색 지연이 발생하기 쉽다. 물론 구현된 시스템은 항상 부분적인 데이터 읽기만을 하는 것은 아니고 읽어야 할 데이터와 데이터 사이가 그리 큰 크기가 아니라면 연속된 모든 데이터를 읽는 방법을 취함으로써 탐색 지연을 줄인다.

4. 역파일 캐시 기법

본 장에서는 디스크 자원의 사용을 줄이기 위해 메모리에 역파일 일부를 캐시하여 사용하는 방법을 기술한다.

4.1 캐시 사용의 필요성

웹 검색 질의를 처리하기 위해 생성된 역파일의 경우에는 색인된 문서 수가 적게는 수천만 개에서 많게는 수십억 개에 달하기 때문에 키워드 조인과 랭킹 계산을 위해 읽어야할 디스크 데이터의 크기가 매우 크다. 따라서 검색엔진이 매 사용자 질의 처리 시마다 역파일을 디스크로부터 읽어 랭킹 계산을 수행한다면 과도한 I/O 사용이 초래될 것이다. 이런 디스크 과다 사용에 따른 성능 저하를 막기 위해 캐시 기법을 적용하였다.

앞의 (그림 1)에서 키워드 디렉터리 부분은 메모리에 상주된 부분이고, 문서식별자 리스트와 META+OFFSET 파일은 일반적으로 디스크로부터 읽혀져야 할 데이터들이다. 구현된 캐시 기법은 디스크에 저장될 데이터 중에서 문서 식별자 리스트 일부를 메모리에 캐싱하는 정책을 취한다. 이와 달리 같은 디스크에 저장된 데이터이지만 META+OFFSET 데이터는 메모리에 캐시되지 않도록 강제한다.

이런 정책을 사용하는 것은 META+OFFSET 데이터가 상대적으로 크기가 크고, 데이터 특성상 캐시 히트가 발생하기 매우 어렵다는 특성 때문이다. 예를 들어 키워드 *k1*에 대응되는 META+OFFSET가 있다고 할 때, 이 데이터 중 어떤 일부 데이터가 읽혀질 것이냐는 것은 *k1*과 함께 질의어를 이루는 다른 키워드 구성에 따라 변화가 발생한다. 이에 반해 문서식별자 리스트는 다른 키워드 집합과는 관계없이 질의어에 *k1*이 포함된다면 늘 일정하게 읽혀야 하는 데이터이다. 이런 이유로 *k1*에 대한 META+OFFSET 데이터의 특정 일부를 캐시할 것을 결정하기가 어렵고, 또 이 데이터의 크기가 매우 크기 때문에 전체를 모두 메모리에 캐시할 수도 없다. 그러므로 제안한 방법에서는 META+OFFSET 데이터는 캐시 영역에 기록되는 것을 완전히 봉쇄하는 정책을 취한다. 메모리에 캐시되는 문서 식별자 리스트의 경우에도 예상되는 캐시 적중률 및 사용될 메모리 크기를 고려해서 2단계의 메모리 캐시 계층으로 구성된다. 이에 대한 사항은 다음 절에 기술한다.

4.2 계층적 메모리 캐시

색인된 모든 n개의 키워드에 대한 문서식별자 데이터 집합을 $D = \{d_1, d_2, d_3, ,.. ,d_n\}$이라 할 때, 키워드 k_j 에 대한 문서식별자 리스트 d_j이 읽혀질 확률은 입력되는 전체 질의들에서 k_j 인기도에 비례한다. 즉, 인기 키워드의 문서식별자 리스트가 핫스팟(hot spot)이 된다. 논문에서는 이런 인기 키워드 집합을 핫스팟 키워드라고 칭하며 이런 핫스팟 키워드 집합은 예전 입력된 사용자 질의를 저장하는 로그 데이터를 분석함으로써 얻을 수 있다. 이런 정보에 따른 핫스팟 키워드에 대해서 적정 개수만큼 해당 문서식별자 리스트를 메모리로 적재하여 사용한다면 질의 처리 시에 디스크 사용량을 크게 줄일 수 있다. 이런 아이디어에 따라 핫스팟 키워드에 대한 정보를 메모리에 캐시하고 이를 상위 캐시 메모리라 칭한다.

이런 캐시 기법을 사용할 때 고려되어야 할 사항이 있다. 실제 질의 처리과정에 있어서는 이런 핫스팟 키워드들만이 입력되는 것이 아니기 때문에 다른 다양한 키워드들의 문서식별자 리스트도 역시 메모리로 읽혀진다는 점이다. 이는 일반적인 디스크 I/O 처리과정을 살펴보면 이해할 수 있다. 디스크로부터 읽혀진 데이터는 응용 프로그램의 메모리 영역으로 직접 들어오는 것이 아니고, 일단 운영체제 커널의 버퍼 메모리 영역으로 읽혀진 후 응용 프로그램 메모리로 복사된다. 응용 프로그램에서 질의 처리를 수행한 후 해당 데이터를 삭제한다고 하여도 커널 내의 메모리 영역에서는 해당 데이터가 즉각 삭제되지는 않는다. 이때 다소 빠른 속도로 질의어가 처리되는 상황이 발생하면 커널 메모리에서 발견되지 않는 문서식별자 리스트들을 버퍼 메모리로 읽어오기 위해 커널 메모리 사용이 자연 증가하는 현상이 발생한다.

이런 현상은 입력되는 질의 수가 증가할수록 심화되면으로 인해 사용 가능한 메모리 영역 대부분을 커널의 버퍼 공간이 차지하게 되고 핫스팟 키워드에 할당했던 메모리 공간이 줄어들어 캐시 히트가 발생하지 않는 현상이 생길 수 있다. 이런 현상이 발생하면 질의 처리 시간이 급격하게 증가하여 시스템 전체에 병목현상을 초래할 수 있다.

이런 문제점을 해결하기 위해 구현된 시스템에서는 커널 내부의 버퍼 공간이 빠르게 확장되지 않고 일정 수준 유지될 수 있도록 커널 버퍼공간을 한단계 낮은 수준의 캐시 공간으로서 관리한다. 즉, 일정한 조건을 만족시키는 문서식별자 리스트들만 커널 공간으로 읽혀 질수 있게 하고 나머지 문서식별자 데이터는 커널 공간으로 읽혀 지지 않고, 디스크에서 곧바로 응용 프로그램의 메모리로 읽혀

지도록 한다. 이를 위해 Windows Server 에디션에서 제공되는 non-blocked I/O 기능을 사용하여 운영체제 커널을 거치지 않는 디스크 데이터 읽기를 수행한다. 이런 방법을 통해 커널 버퍼 공간이 점차 확장되는 현상을 막음과 동시에 메모리 복사로 인한 시간 낭비도 함께 없앨 수 있다는 장점이 있다.

앞서 설명한 구현 방식을 정리하면 다음과 같다. 메모리 내의 캐시 데이터 영역은 두 개로 분류될 수 있다. 상위 계층은 응용 프로그램 메모리 영역에서 관리되는 캐시 영역으로, 핫스팟 키워드에 대한 문서 식별자 리스트가 저장된다. 다음의 하위 계층은 커널 메모리 영역에 위치한 캐시 데이터로 이는 운영체제 커널의 버퍼 메모리 관리 기법에 의해 캐시 데이터가 자동 저장되거나 삭제된다. 즉, 응용 프로그램인 검색 엔진이 직접 캐시 데이터를 관리하지는 않지만, 질의 빈도에 따라 일정 정도의 메모리 크기를 확보하고 자동 관리되는 데이터 영역이라 할 수 있다.

부연하면, 문서식별자 데이터는 크게 두 개 종류의 메모리 공간에 저장될 수 있다. 첫 번째는 핫스팟 키워드를 위한 일반 응용 프로그램 메모리 공간이고, 두 번째는 핫스팟 키워드를 제외한 키워드 중에서 캐싱 조건을 만족시키는 키워드들을 위한 커널 메모리 공간으로 구분할 수 있다. 물론 두 번째의 메모리 공간은 랭킹 계산을 수행하는 사용자 프로그램이 세부 사항을 컨트롤할 수 있는 공간은 아니며, 커널의 버퍼 공간 관리 알고리즘에 의해 통제되는 공간이다. 구현한 시스템은 이런 서로 다른 형태의 메모리 공간에 캐시되는 데이터를 2계층의 데이터로 구별해 관리한다.

아래 (그림 2)는 이런 2 계층의 캐시 공간을 사용해서 랭킹 계산을 하는 알고리즘을 보인다. 알고리즘의 라인 4-16에서는 문서 식별자 리스트를 읽어오기 위해 함수 콜이 수행되며, 이때 캐시 데이터가 사용되거나 갱신된다. 문서 식별자 정보를 모두 읽어 들인 후에는 라인 17에서 실제 랭킹 계산이 수행되며, 이때 *CalcRanks()* 함수에서 META+OFFSET 데이터를 읽을 때는 항상 non-blocked I/O 콜이 수행된다. 라인 8에서는 키워드 k가 캐싱할 키워드 인지를 체크되며, 해당 조건을 만족할 때는 라인 9가 수행됨으로써 해당 문서식별자 리스트가 커널 메모리에 캐싱된다.

---

procedure GetRankValues()

Input:: Q. // 질의어. 키워드 집합.
H. // 핫스팟 키워드 집합

Output:: R. // 랭킹 결과. (문서식별자, 랭킹값) 리스트

1. j ← 1. // 키워드 인덱스 초기화.
2. $P \leftarrow \varnothing$. // 포인터 초기화.
3. **foreach** $k \in Q$ // 질의의 키워드에 대한 문서식별자 리스트를 읽고 이에 대한 메모리 주소를 p_j에 기록.
4. **if**($k \in H$) // k가 핫스팟 키워드 인지 체크
5. $p_j \leftarrow$ *GetPtrCache*(k); // k의 캐시 메모리 주소반환
6. **else** {
7. (addr, n) ← *Lookup*(k). // 키워드 디렉터를 검색. 해당 엔트리를 반환.
8. **if**(n < max_size **and** *QueryFreq*(k) > min_frequency)
9. $p_j \leftarrow$ *ReadCached*(addr, n). // Blocked I/O 콜 사용하여 메모리로 적재. 커널 메모리에 캐시됨.
10. **else**
11. $p_j \leftarrow$ *ReadNotCached*(addr, n). // Non-blocked I/O 콜 사용하여 메모리로 적재. 캐시되지 않음.
12. **end_if**
13. }
14. $P \leftarrow P \cup \{ p_j \}$.
15. $j \leftarrow j + 1$. // 다음 키워드 인덱스
16. **end_foreach**
17. $R \leftarrow$ *CalcRanks*(Q, P). // 랭킹 계산 결과를 R에 반환

(그림 2) 랭킹 계산 알고리즘.

5. 결론

본 논문에서는 키워드 검색에서 널리 사용되는 색인 기법인 역파일을 효과적으로 사용할 수 있는 방법에 대해서 기술하였다. 대용량의 문서에 대한 역파일 구성시 문제가 될 수 있는 과도한 디스크 자원 사용을 막기 위해 색인 파일을 3단계로 구성하여 키워드 조인 단계와 랭킹 계산 단계에서 읽혀질 데이터를 분리 저장하였다. 이와 함께 문서식별자 리스트를 메모리에 적재함으로써 디스크 사용 가능성을 줄이는 방식을 함께 취한다. 이를 위해 메모리 내에 2개 계층의 캐시 영역을 구성하였으며 상위 계층은 응용 프로그램의 메모리 영역에, 하위 계층은 운영체제의 커널 메모리 영역에 두어 성능을 높일 수 있게 했다. 논문의 역파일 구조는 실제 상용 웹 검색 엔진에 적용되어 사용된 바가 있다. 실제 적용 사례를 통해 검색 엔진의 안전성 및 디스크 사용을 줄이는 효과가 검증되었다.

참고문헌

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

[2] 임성채, "계층적 캐싱 기법을 이용한 대용량 웹 검색 엔진의 구현", 2007년 KCC 논문집, Vol. 34, No. 1(C), pp. 87-91.

[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] 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.