계층적 캐시 기법을 이용한 대용량 웹 검색 질의 처리 시스템의 구현

(Implementation of a Large-scale Web Query Processing System Using the Multi-level Cache Scheme)

임 성 채 †

(Lim Sung Chae)

요 약 웹을 이용한 정보 공개 및 검색이 확대됨에 따라 웹 검색 엔진도 지속적인 주목을 받고 있다. 이에 따라 웹 검색 엔진의 다양한 기술적 문제를 해결하고자 하는 연구가 있었음에도 웹 검색 엔진의 질의 처리 시스템에 대한 기술적 내용은 잘 다뤄지지 않았다. 질의 처리 시스템의 경우 소프트웨어 아키텍처나 운영 기법을 고안하기 어렵기 때문에 본 논문에서는 구현된 상용 시스템을 바탕으로 관련 기술을 소개하고자 한다. 구현된 질의 처리 시스템은 6,500 만개 웹 문서를 색인하여 일 500만개 이상의 사용자 질의 요청을 수행하는 큰 규모의 시스템이다. 구현한 시스템은 질의 처리 결과를 재사용하기 위해 계층적 캐시 기법을 적용했으며, 저장된 캐시 데이타는 4계층으로 구성된 데이타 저장소에 분산 저장되는 것이 특징이다. 계층적 캐시 기법을 통해 질의 처리 용량을 400% 정도로 향상 시킬 수 있었으며 이를 통해 서버 구축 비용을 70% 정도 절감할 수 있었다.

키워드 : 웹 검색 엔진, 캐시 기법, 서버 클러스터, 웹 질의 처리

Abstract With the increasing demands of information sharing and searches via the web, the web search engine has drawn much attention. Although many researches have been done to solve technical challenges to build the web search engine, the issue regarding its query processing system is rarely dealt with. Since the software architecture and operational schemes of the query processing system are hard to elaborate, we here present related techniques implemented on a commercial system. The implemented system is a very large-scale system that can process 5-million user queries per day by using index files built on about 65-million web pages. We implement a multi-level cache scheme to save already returned query results for performance considerations, and the multi-level cache is managed in 4-level cache storage areas. Using the multi-level cache, we can improve the system throughput by a factor of 4, thereby reducing around 70% of the server cost

Key words : web search engine, cache scheme, server cluster, web query processing

1. 서 론

지난 십 년의 기간 동안 웹을 이용한 정보 공개 및 공유가 빠르게 확대됨에 따라 웹 검색 엔진의 중요성도 함께 증가했다. 웹 검색 엔진은 웹 로봇 프로그램이 인터넷에서 접근되는 HTML 문서를 포함한 다양한 문서를 자동 수집하여 키워드 기반의 검색 서비스를 제공하기 위한 제반 시스템을 의미한다. 이런 웹 검색 서비스는 90년대 초반 라이코스(Lycos) 포탈 사이트에서 최초로 상용 서비스를 시작함으로써 본격화 되었다. 당시 색인된 웹 문서의 양이 5만 페이지를 조금 넘는 수준이었으며, 현재는 그 규모가 수십억 페이지로 확대되었다[1,2].

증가하는 웹 검색 엔진의 중요성에 따라 관련 연구도 다양한 분야에서 진행되었다. 즉, 웹 문서 자동수집에 대한 연구[3,4], 색인 파일 구성에 대한 연구[5,6], 링크 분석에 관한 연구[7-9], 사용자 검색 의도에 기반한 검

· 이 논문은 2006년도 동덕여자대학교 학술연구비 지원에 의해 수행된 것임

† 정 회 원 : 동덕여자대학교 컴퓨터학과 교수
sclim@dongduk.ac.kr

논문접수 : 2007년 7월 5일
심사완료 : 2008년 8월 18일

Copyright@2008 한국정보과학회 : 개인 목적이나 교육 목적인 경우, 이 저작물의 전체 또는 일부에 대한 복사본 혹은 디지털 사본의 제작을 허가합니다. 이 때, 사본은 상업적 수단으로 사용할 수 없으며 첫 페이지에 본 문구와 출처를 반드시 명시해야 합니다. 이 외의 목적으로 복제, 배포, 출판, 전송 등 모든 유형의 사용행위를 하는 경우에 대하여는 사전에 허가를 얻고 비용을 지불해야 합니다.

정보과학회논문지: 컴퓨팅의 실제 및 레터 제14권 제7호(2008.10)

색 기법[10,11], 주제별 색인 관련 연구[7,12] 등을 예로 들 수 있다. 이들 연구들은 웹 검색 서비스를 위한 데이타 준비 단계에 관련된 것들이며, 사용자 웹 질의를 실제로 처리하는 시스템에 대한 연구는 부족한 것이 사실이다. 이에 따라 웹 검색 서비스나 유사 시스템을 구현하고자 할 때 관련 기술 자료를 얻기에 어려움이 있다. 이런 어려움을 해소하기 위해 본 논문에서는 대용량의 상용 웹 검색 엔진의 구현 및 운영 경험을 바탕으로 웹 검색 엔진 시스템 전반적인 사항과, 기존 논문에서는 자주 다뤄지지 않았던 질의 처리 시스템에 대해 기술한다. 특히, 핵심 주제라 할 수 있는 계층적 캐시(cache) 기법에 대해 상술한다.

대용량 데이타를 빠른 시간 안에 처리하는 웹 검색의 특성상 질의 처리 시스템은 단일 서버로는 구현이 불가능하며 고속 LAN에 기반한 분산 구조가 필수적이다[13]. 또한 질의 처리에 드는 디스크와 CPU 비용이 크기 때문에 자원 요구량을 상당 부분 줄이지 못한다면 과도한 하드웨어 비용이 초래된다. 이런 문제점을 해결하기 위한 것이 웹 검색 시스템에 맞게 구현된 캐시 기법이다. 논문의 캐시 기법은 모두 4개 계층의 캐시 데이타를 사용한다. 이중 최상위 계층의 캐시는 메모리에 저장되며 나머지 3계층의 캐시들은 디스크를 사용하여 많은 수의 웹 질의 결과를 저장할 수 있다. 논문에서 구현한 계층적 캐시 기법을 사용하여 6,500만 개의 웹 문서가 색인되고 하루 500만 번 이상의 사용자 질의를 처리하는 시스템에서, 4배 가까운 성능 향상을 볼 수 있었으며 이를 통해 70% 정도의 서버 비용을 절감했다. 본 논문의 시스템 구조와 설계시의 고려 사항들은 다른 유사 시스템의 개발에 참고 자료가 될 수 있을 것이라 기대한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구를 설명하고, 3장에서는 질의 처리를 위한 색인 구조에 대해서 기술한다. 4장에서는 계층적 캐시를 사용한 질의 처리 시스템의 구조 및 알고리즘에 대해서 설명한다. 5장에서는 계층적 캐시 사용을 통한 성능 향상 효과에 대해 서술하고, 이어지는 6장에서 결론을 맺는다.

2. 관련 연구

웹 검색 기술 관련된 연구들은 크게 두 가지로 분류 가능하다. 첫째는 웹 검색 엔진에서 사용할 데이타의 생성에 관련된 연구들로서 웹 문서 크롤링(crawling) 기법[3,4], 웹 문서 중요도 계산방법[7-9,12], 색인 데이타의 처리 기법[5,6]들이 여기에 속한다. 웹 문서 중요도 계산에 있어 웹 문서간의 링크 연결을 이용한 연구들이 구글(Google.com)의 성공으로 가장 활발하였다. 확장된 형태로는 사이트의 주제 연관성을 고려한 문서 중요도 계산에 대한 연구가 있다. 예를 들어, 웹 사이트 S가 있을 때 S가 가지는 중요도는 서로 다른 주제인 영화나 여행에 대해서는 상대적인 차이가 있다고 보는 것이 좀더 정확하다. 이런 주제에 따른 랭킹 부여 방식은 이후 의미 기반의 질의 처리에도 유용하게 사용될 수 있다[10].

둘째는 웹 검색 엔진에 사용될 데이타가 생성된 후 이를 사용하여 효과적으로 질의를 처리할 수 있는 방법에 대한 연구이다. 웹 검색 엔진에서 질의 처리 기술이 중요시되는 것은 웹 검색의 특성상 생성된 데이타가 매우 대용량이기 때문이다. 국내 웹 문서 검색 서비스를 구현한다고 할 때에도 문서의 수가 최소 5,000만 개 이상을 넘어야 한다. 해외 사이트 일부를 함께 서비스 하는 경우 6,000만 개 이상의 웹 문서를 수집하고 색인해야 하는데, 이 때 생성되는 데이타의 크기는 테라(tera) 바이트 이상이다. 이런 데이타를 읽어 실시간으로 질의 결과를 내기 위해서는 고성능의 분산 처리 기술 및 캐시 기법이 요구된다. 여기서 캐시 기법이란 한번 처리한 질의 결과를 시스템의 저장 공간에 기록해두고 다시 사용함을 의미한다.

웹 질의 처리 과정에서 수십 메가 바이트의 디스크 데이타가 읽혀져야 하기 때문에 질의 처리시에 이런 크기의 데이타가 매번 읽혀 진다면 수백 개의 동시 질의가 처리되는 시스템 특성상 과도한 서버 자원이 요구된다. 여기에 대해 캐시 기법은 질의 처리에 소모되는 디스크 자원이나 CPU 자원의 중요 하드웨어 자원을 최소로 사용하면서 질의 응답시간을 크게 줄일 수 있는 장점이 있다[14]. 이런 이유로 기존 연구에서도 로그 데이타를 분석하여 보다 효과적인 캐시 데이타를 생성하고 유지하는 기법에 대한 연구[15], 기존 LRU(Least Recently Used) 알고리즘의 효율성을 개선한 캐시 알고리즘들이 연구된 바 있다[14,16].

이런 기존 연구의 목적은 다양한 방식을 통해 최적의 캐시 집합을 결정하는 것이었다. 캐시 공간은 제한적일 수 있기 때문에 가능하면 재사용할 수 있는 확률이 높은 질의들을 선택함으로써 전체적인 캐시 적중률(hit rate)를 높이는데 관심이 있다. 따라서 질의 로그 데이타를 통해 미리 질의 입력 특성을 분석한다거나, 캐시를 2-3개의 계층으로 구성하여 두고 캐시 계층간에 저장되는 질의를 동적으로 이동시키는 알고리즘을 고안하는데 관심이 있었다.

하지만 본 논문에서 구현한 캐시 기법은 기존의 연구와는 다른 문제 접근 방식을 취한다. 하루 500만 질의를 처리하는 시스템을 구현하기 위해 사용한 분산 서버 구조에서 질의가 처리되는 과정을 고려한 현실성 있는 캐시 기법을 기술하려고 한다. 기존 연구들은 상용 서비스를 위한 시스템에 적용된 것이 아니고 다소 이론적 측면에 관심이 많다. 하지만 이런 연구들은 실제 시스템의

성능향상에는 큰 도움을 주기 어렵다. 일반적으로 웹 검색 엔진에서 하나의 질의 처리에 수 백 번의 디스크 읽기 연산이 필요하다는 것을 감안한다면, 가끔 얻게 되는 몇 번의 디스크 읽기 감소가 시스템 전체 성능향상에 큰 영향을 미치지는 못한다.

본 논문에서는 이런 관점에서 고속 LAN으로 연결된 분산 서버 환경에서 어떻게 캐시 데이타를 배치하고, 어떤 캐시 데이타를 유지해야 하는 지에 관심을 가진다. 물론 필요에 따라 캐시 레코드의 교체(replacement)는 이뤄져야 하지만 단순한 형태의 알고리즘을 사용하는 것만으로도 충분하다는 것을 실제 운영을 통해 알 수 있었다. 이런 것이 가능 한 것은 몇 개의 계층으로 이루어진 캐시 데이타들 간에 저장하는 질의 집합이 공유될 수 있고, 하위 캐시 계층에서는 저장하는 데이타의 크기를 작게 하여 충분한 디스크 공간을 확보할 수 있기 때문이다. 이런 방식을 사용하면 비록 상위 계층에서 캐시 레코드가 발견되지 않아 몇 번의 I/O 연산이 더 발생할 수 있지만, 캐시 데이타 없이 질의를 처리했을 때의 수 백 번의 디스크 읽기 동작은 막을 수 있다. 즉, 상위 레벨에서의 캐시 레코드 교체 알고리즘의 효율성은 전체 시스템의 관점에서 보았을 때 큰 영향은 끼치지 않기 때문에 본 논문의 중요한 관심은 되지 못했다. 물론 기존 논문의 특정 알고리즘을 적용하여 구현 시스템의 캐시 레코드를 관리하는 방식을 취하는 것에는 문제가 없다. 다만 주 관심을 어떤 형태의 캐시 레코드를 몇 단계로 구성하여 관리할 것인지, 어떤 서버에 관리할 것인지, 그리고 이를 통해 어떤 성능 향상을 기대할 수 있는지에 둔다는 것이다.

3. 질의 처리를 위한 색인구조

크롤링 기술에 따라 웹 문서를 자동 수집하고 수집된 문서에 대해서는 키워드 검색할 수 있도록 역파일(inverted file) 색인을 생성한다. 역파일은 웹 문서에서 색인할 키워드를 추출한 후 키워드 별로 추출되었던 문서 식별자 리스트를 기록한 파일이다. 즉, 각 키워드가 어떤 문서에 출현했는지를 색인한 파일이다. 웹 검색은 단순한 키워드 검색이 주이고 갱신 작업이 거의 없기 때문에 역파일이 색인기법으로 널리 쓰이고 있다[2,7].

이런 역파일에는 질의의 문서 연관성을 평가하는 랭킹 작업에 사용될 색인 정보도 함께 저장된다. 이런 색인 정보로서 키워드 위치(offset) 정보가 있다. 질의를 이루는 키워드가 여러 개일 때 이들 키워드가 문서 내에서 어떤 인접도를 갖는지는 질의 연관성 평가에 있어 매우 중요한 요소이다. 따라서 이런 정보가 역파일에 저장되어야 하며, 일반적으로 키워드 위치 정보와 함께 한국어 형태소(morpheme) 정보도 함께 저장된다. 예를 들어 복합명사 C가 단일명사 A와 B로 구성된다면, A, B, C 세 개 명사가 모두 동일한 키워드 위치로 색인된다. 하지만 이들 키워드에는 서로 다른 형태소 분석 정보가 부여되어야 하며, 이런 형태소 분석 정보를 저장하기 위해 키워드 위치를 표시하는 바이트 중 일부 상위 비트를 사용한다. 논문에서는 *키워드 위치 정보와 형태소 분석 정보를 합쳐* ***OFFSET 정보***라 부른다.

OFFSET 정보 외에 키워드의 문서 내 중요도를 나타내는 정보도 존재한다. 예를 들어 키워드 *k*가 웹 문서 제목에 있거나, 이탤릭체 혹은 큰 사이즈의 폰트로 표현되었다면 보통의 키워드에 비해 그 중요도가 클 확률이 높다고 할 수 있다. 이외에도 키워드의 문서 내 앵커 텍스트 포함 유무, 해당 문서를 가리키는 외부 앵커(incoming anchor) 텍스트 내의 포함 유무, URL 내의 존재 유무 및 URL 내의 위치 등은 랭킹 계산에 있어 중요 정보이다. 이와 함께 각 문서의 문서중요도 역시 역파일에 저장된다. 문서중요도란 구글의 PageRank[7] 정보와 동일한 성격의 정보이다. 이런 정보들을 모두 합쳐 논문에서는 ***색인 META 정보***라 칭한다.

색인 정보를 저장하는 역파일의 저장구조는 랭킹 계산시 디스크 사용의 효율성을 고려하여 설계되어야 한다. 먼저 랭킹 과정을 간단히 살펴본다. 설명을 위해 키워드 *k1*과 *k2*로 이루어진 질의를 고려해 보자. 랭킹 계산의 첫 단계로서 키워드 조인 단계가 있다. 이 단계에서는 *k1*과 *k2*가 출현한 문서 식별자 리스트(list)를 각각 읽은 후에 이들 간에 동일조인(equi-join)이 수행된다. 이를 통해 질의에 부합(matching)하는 문서 집합 식별자들을 구한다. 다음 단계로 여기서 구한 문서 집합에 대해서 랭킹 계산을 수행한다. 이때 사용될 정보가 앞서의 OFFSET 및 색인 META 정보이다.

이런 두 단계의 랭킹 과정 중 첫 단계에서는 *k1*과 *k2*의 문서 식별자 정보가 필요하며, 두 번째 단계에서는 관련 색인정보가 필요하다. 이런 OFFSET+META 색인 정보는 문서 식별자 정보에 비해 그 크기가 크며, 질의 부합하는 문서들에 대해서만 관련 데이타를 읽을 수 있어야 한다. 이런 이유로 OFFSET+META 정보는 선별적 읽기가 가능해야 한다.

이런 점을 고려해 사용된 역파일은 그림 1과 같이 세 계층으로 설계되었다. 가장 상위 계층인 키워드 디렉터리 파일 계층은 생성 후 메모리에 적재되며, 주어진 키워드의 디렉터리 엔트리(entry)는 이진탐색으로 빠르게 접근된다. 그림 1의 키워드 디렉터리 엔트리에는 해당 키워드가 출현한 문서의 전체 개수가 저장되며, 이 데이타 바로 뒤에 관련 DID(Document ID) 리스트가 기록된 파일 주소 정보가 있다. 그림에서 키워드 *k1*의 경우 출현문서수는 3이어야 하며, 관련 DID 리스트는 $D_{1,1}$에서 $D_{2,1}$으로

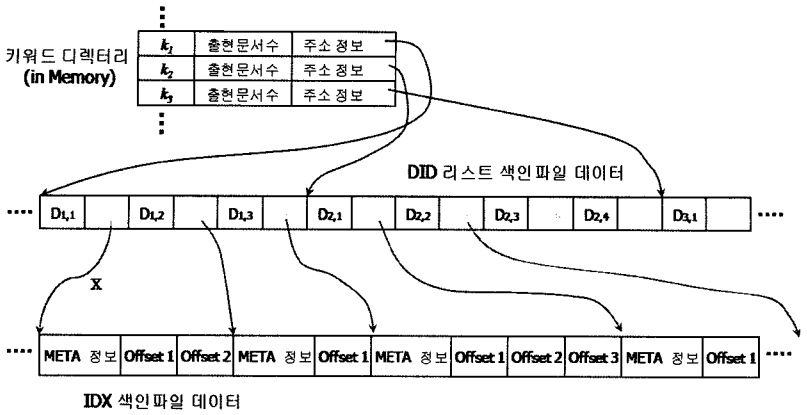

그림 1 사용되는 3계층의 역파일 구조

표시된 곳 사이 공간에 위치한다. *k2*의 경우는 4개 문서에 출현했으며 관련된 DID 리스트는 $D_{2,1}$에서 $D_{3,1}$ 사이에 저장되었다. 이 두 DID 리스트에 정렬된 DID 값을 비교함으로써 질의 부합하는 DID 집합을 정할 수 있다.

DID 리스트에는 각 DID에 대한 META+OFFSET 정보가 저장된 파일(즉, IDX 색인파일)의 위치 정보가 함께 저장된다. 예를 들어 그림 1에서 *k1*의 $D_{1,1}$ 문서에 대한 META+OFFSET 정보 저장 위치는 화살표 *X*가 가리킨다. 그림에 따르면 *k1*은 $D_{1,1}$ 문서에서 2회 출현했음을 알 수 있다. 앞에 위치한 META 데이타는 모든 문서에 대해 동일한 크기를 가지며 OFFSET 데이타는 출현 횟수에 따라 크기가 변한다. *k1*과 *k2*의 키워드 조인에 있어, 만약 $D_{1,1} = D_{2,1}$이고 $D_{1,3} = D_{2,4}$이라면, 질의에 부합하는 문서 수는 2가 될 것이다. 또한 이런 두 문서에 대해 랭킹 값을 계산하기 위해 META+OFFSET 데이타를 읽기 위해서는 전체 4회의 디스크 탐색 지연이 발생할 수 있다. 이처럼 IDX 파일은 선택적인 데이타 접근이 발생하기 때문에 디스크 탐색 지연이 발생하기 쉽다. 물론 구현된 시스템은 항상 부분적인 데이타 읽기만을 하는 것은 아니고 읽어야 할 데이타와 데이타 사이가 그리 큰 크기가 아니라면 연속된 모든 데이타를 읽는 방법을 취함으로써 탐색 지연을 줄인다. 이에 대한 자세한 사항은 본 논문의 중요 범위가 아니기에 생략한다.

4. 질의 처리를 위한 계층적 캐시 기법

본 장에서는 웹 검색 엔진의 질의 처리 시스템을 위한 하드웨어 구성 및 중요 주제인 계층적 캐시 기법에 대해서 서술한다.

4.1 시스템 구조

웹 질의는 크게 2 단계에 걸쳐 처리된다. 1 단계는 랭킹 계산 단계로서 그림 1의 역파일을 사용하여 질의와 웹 문서간의 연관성 크기에 따라 랭킹값이 부여된다. 즉, 질의 부합하는 문서들에 대해 <DID, 랭크값>을 구하는 단계다. 2 단계는 1 단계에서 얻은 랭킹 순위에 따라 SRT(Search Result Text)를 구하고 검색 결과를 생성하는 단계이다. 여기서 SRT라 함은 각 DID에 대한 검색 결과 요약문으로서, URL 정보, 제목, 그리고 하이라이팅된 요약문으로 구성된다.

이런 2 단계 질의 처리 과정을 고려해 구현된 시스템의 하드웨어 구성도를 그림 2에서 보인다. 그림 2는 시스템의 하드웨어 구성도이며 시스템은 유료 IDC(Internet Data Center)에 설치된다. 외부 인터넷과는 100 Mbps IDC 포트로 연결되며, 포트 바로 뒤 단에 보안을 위해 방화벽 장비가 있다. 방화벽 뒤에는 CISCO Layer-4(L4) 장비가 있어 웹 서버들 간의 작업량 분산 및 장애극복(fail-over) 기능을 수행한다. L4 스위치를 거쳐 웹 서버로 사용자 질의가 입력되면 웹 서버는 해당 질의를 분석하여 키워드들과 검색 범위를 결정한다. 여기서 검색 범위라 함은 사용자의 검색 결과 페이지에 보여질 웹 문서의 랭킹 범위를 의미한다. 일반적으로 사용자들은 상위 10개의 랭킹 범위를 초기 검색범위로 하고, 이후 링크 연결을 통해 하위 검색범위로 이동한다.

앞서 언급한 2 단계 질의 처리 작업은 그림 2의 중간단에 위치한 6대의 중계자(coordinator) 서버에 의해 진행된다. 아랫단의 랭커 서버 클러스터(cluster)는 1 단계 작업인 랭킹 계산을 위한 서버들이고, SRT 서버 클러스터는 2 단계 SRT 생성용 서버들의 모임이다. 랭커 클러스터는 모두 4개가 있으며, 각 클러스터는 4대의 랭커 서버로 구성되었다. 동일한 랭커 클러스터 내의 서버들은 모두 같은 색인 파일을 저장하며, 이들 서버들 사이에는 작업 분산 및 장애극복 기능이 수행된다. 수집된 웹 문서는 비슷한 크기의 4개 집합으로 분할되어 각각 색인되며 각 분할된 색인 데이타가 서로 다른 랭커 클러스터로 저장된다. 따라서 하나의 질의 처리를 위해서

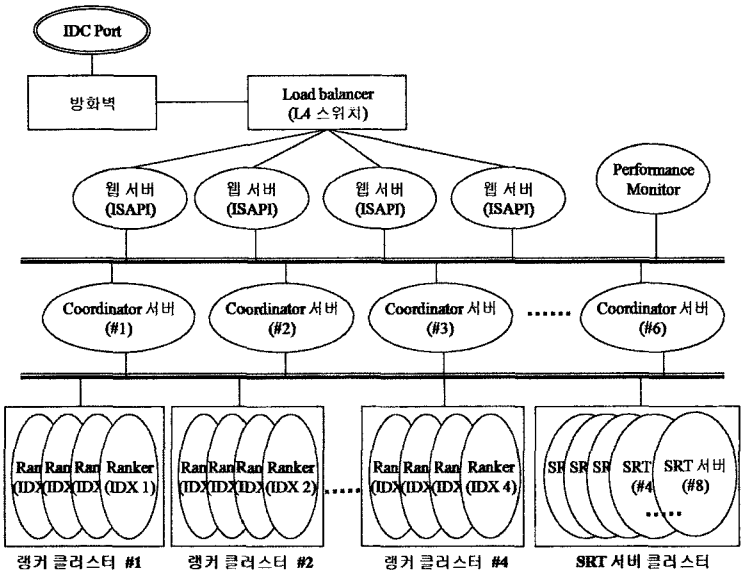

그림 2 질의 처리 시스템의 하드웨어 구성도

는 4개 랭커 클러스터들로부터 <DID, 랭크값> 리스트들을 모으고 다시 재정렬되어야 한다. SRT 클러스터는 모두 8대 서버로 구성하며각 서버는 웹 문서에서 HTML 태그를 삭제한 텍스트 데이타들을 압축해 저장하며, 서로 다른 서버들에는 동일한 데이타 집합을 중복 저장한다.

4.2 캐시 기법의 필요성

캐시 필요성은 질의 처리 과정을 이해함으로써 알 수 있다. 그림 3은 3.1 절에서 언급한 질의 처리단계를 중계자 서버 측면에서 기술한 것으로, 1 단계는 라인 1-7을 통해, 2 단계는 라인 8-16을 통해 처리된다. 그림 3에는 캐시 데이타 사용은 고려하지 않았다.

Input: *Query_url*. /* 질의 URL. *Query_url*에 키워드들과 질의 범위가 포함됨 */

Output: *Q*. /* 질의 결과 페이지. */

1. (*Keywords*, *Query_range*) ← Parse(*Query_url*). /* 키워드와 질의범위 추출. */
2. *R* ← ∅. /* 초기화. */
3. **foreach** *S*, *s* ∈ *S* /* *S* 는 랭커 서버 클러스터. */
4. *X* ← Send(Keywords, s). /* 랭커 서버 s로 질의를 보내고 랭킹 결과를 *X*에 저장 */
5. *R* ← *R* ∪ *X*. /* *X*는 <DID, rank value> 엔트리로 구성됨. */
6. **end_for**
7. *R* ← Sort(*R*). /* *R*을 rank value 크기의 내림차순으로 정렬 */
8. *Top_K* = Select(*R*, DUP_K). /* DUP_K개 상위랭크 엔트리를 선택하여 *Top_K*에 저장 */
9. **for** (*N*=0; | *Top_K* | > 0; *N*++) /* SRT 서버로 보낼 DID 목록을 준비 */
10. *Request*[*N*] ← Extract(*Top_K*, 20). /* 최대 랭킹 상위 20개 엔트리를 추출 */
11. Select randomly *N* SRT servers in the SRT cluster and let *s*[0], *s*[2], .., *s*[*N-1*] be them.
12. *X*[*0, ..., N-1*] ← Send(*Keywords, Request[0, ..., N-1], s[0, ..., N-1]*,). /* 서버 *s*[*i*]로 SRT 생성 요청을 보내고 결과는 *X*[*i*]에 저장. *X*[*i*]는 <DID, SRT> 엔트리로 구성됨. /*
13. *T* ← *X*[0] ∪ *X*[2] ∪ ... *X*[*N* − 1]. /* SRT 데이터를 랭킹순서로 합병. */
14. *Obsolete* ← CheckDuplication(*T*). /* SRT 데이터의 비교를 통해 제거될 중복 DID를 찾고 이를 *Obsolete*에 저장. */
15. *T* ← Delete(*T*, *Obsolete*); *R* ← Delete(*R*, *Obsolete*). /* 중복된 DID를 *T*와 *R*에서 제거 */
16. Create a query result page *Q* using *T*, *R* and *Query_range*. /* 검색 결과페이지 생성 */

그림 3 중계자 서버에 의한 질의 처리 세부과정

그림 3에서 자원 사용이 많은 과정은 라인 3-6과 라인 12이다. 라인 3-6에서의 디스크 자원 사용을 보기 위해 키워드 *k1*과 *k2*의 질의 Q를 가정해 본다. 이 두 키워드가 하나의 랭커 서버의 문서에 출현 하는 회수가 각각 5만과 10만이라면, 그림 1의 색인 파일로 동일조인을 수행할 때 약 1.2 MB 크기의 DID 리스트 파일을 읽어야 한다(DID와 주소정보 각각 4 바이트 사용). 질의 처리에 4대의 랭커 서버가 사용되므로 전체 4.8 MB 정도의 디스크 읽기가 필요하다. 여기에 더해, 랭킹 계산에는 디스크 탐색지연과 크기가 큰 IDX 데이타로 인해 보통 동일조인의 2배 정도의 디스크 비용이 필요하다. 따라서 질의 Q의 처리에는 14.4 MB 크기의 디스크 데이타를 읽는 비용이 필요하다. 이는 적지 않은 디스크 사용량이며 질의의 키워드 개수가 많을수록 문서 출현 빈도가 높을 수록 그 크기는 증가한다.

라인 12에서도 매우 큰 컴퓨터 자원이 초래된다. SRT 서버가 임의의 DID에 대해 SRT 데이타를 생성할 때, 6,500 만개의 문서 중 어떤 하나를 읽는 것이므로 디스크 임의 접근이 발생한다. 또한 라인 14-15에서 중복제거 작업이 수행되어야 함으로, 질의의 검색범위와 관계없이 상위 랭크값을 갖는 DUP_K개의 SRT가 항상 생성된다. DUP_K는 100 이상의 값으로 고정되기 때문에 많은 수의 디스크 임의 접근이 일어난다. ***중복제거*** 작업이란 수집한 SRT 간의 문자열 비교를 통해 수행되며, 상위 랭크값의 검색 결과에서 중복된 내용으로 보이는 DID들을 제거하는 작업이다.

이 작업은 검색 품질 유지에 매우 중요한 작업이며 색인 작업 시작 전에 수집된 웹 문서에 대해서도 중복성 여부가 체크되고 상당한 웹 문서가 제거된다. 하지만 이때의 제거 작업만으로는 완벽한 제거가 불가능하며, 추후 검색 결과를 생성할 때 라인 14-15에서와 같은 추가적 중복 제거 작업이 필수적이다. 그 결과 라인 12에서 질의범위에 관계없이 일정 개수 이상의 SRT가 생성되며 이로 인해 디스크 자원 소모가 커진다.

디스크 사용 외에도 라인 12에서 SRT 서버의 CPU 비용도 매우 크다. 요약문을 생성하는 과정에서 웹 문서에서 키워드가 많이 출현한 부분을 찾아야 하기 때문에 키워드 숫자만큼 문자열 탐색이 필요하고, 키워드 개수가 3개 이상인 경우는 상당한 CPU 시간이 소모된다. 이때 만약 일반적인 문자열 탐색 함수가 사용된다면 CPU에 병목현상이 발생할 수도 있다. 이를 피하기 위해 오토마타 기법을 적용한 별도의 하이라이팅 알고리즘을 사용하며[17], 이를 통해 STR 서버에서의 CPU 병목 현상을 막는다.

다시 부연하면 라인 3-6와 라인 12에서의 디스크 자원 사용량은 매우 크며, 이런 디스크 사용이 매 질의마다 발생한다면 시스템의 서버 요구량이 과도해 진다. 특히, 시스템의 요구 사항이었던 하루 500만개 질의 처리와 평균 0.7초 내의 응답 시간을 만족시키는 시스템을 구성하는 것은 비용 측면에서 그 구현이 불가능했다. 이에 따라 이전 검색 결과를 캐싱하여 재사용하는 방법이 고려되었다.

4.3 계층적 캐시 기법

본 절에서는 구현된 계층적 캐시 기법에 대해 기술한다. 캐시 데이타는 모두 4개의 계층으로 구성되었으며 서로 다른 서버의 메모리 및 디스크에 저장된다.

4.3.1 메모리 캐시 데이타

캐시 계층 중 가장 상위 계층 캐시인 CL1(Cache Level 1) 캐시는 웹 서버 메모리에 저장되며, 다른 하위 계층 캐시인 CL2, CL3, CL4는 중계자 서버의 디스크에 저장된다. CL1 캐시를 제외한 대부분의 캐시 데이타를 중계자 서버에 두는 것은 2 가지 측면을 고려한 선택이다. 첫째로는 그림 3에서와 같이 질의 처리의 중간 결과들이 중계자 서버에 의해 수집되고 있기 때문이다. 중계자 서버는 컴퓨팅 자원의 소모가 큰 라인 3-6과 라인 12의 작업결과를 효과적으로 저장하고 재사용할 수 있기에 캐시 저장 서버로 적당하다. 둘째로는 하드웨어 비용 측면에서의 고려이다. 랭커 서버와 SRT 서버는 매우 많은 I/O를 수행하고 저장할 데이타 역시 크기 때문에 15개의 SCSI 디스크가 장착된 고가 서버가 사용된다. 반면에 중계자 서버는 이런 고가의 디스크 자원이 요구되지 않기 때문에 저가 서버로도 대치 가능하다. 따라서 캐시 데이타 관리와 같은 작업을 중계자 서버에 둠으로써 다른 고가 서버의 자원 사용을 최대한 억제하였다.

예외가 되는 것은 CL1 데이타로 웹 서버의 메모리에 질의 결과 페이지 자체를 압축하여 저장한다. 하나의 질의 결과 페이지는 유일한 URL로 지정되며, URL을 접근키로 사용하는 CL1 캐시 레코드가 저장된다. 메모리 크기 문제로 인해 CL1 캐시 레코드 수는 2만개 정도로 제한적이며, 전체 캐시 계층에서 가장 접근 빈도가 높은 캐시 레코드를 저장한다고 할 수 있다.

그림 4는 CL1의 관리 알고리즘을 보인다. 라인 1-2에서 검색 결과 페이지에 해당하는 URL을 사용하여 해쉬키를 만들고, 이를 사용하여 URL에 대응되는 캐시 레코드가 저장될 수 있는 논리적 메모리 뱅크의 식별자를 구한다. 각 논리적 메모리 뱅크 안에 여러 개의 메모리 슬롯이 존재하며 각 슬롯은 하나의 캐시 레코드를 저장할 수 있다. 메모리 뱅크 접근 후에는 해쉬키를 사용하여 원하는 캐시 레코드를 검색하며 이런 과정은 라인 4-8에서 보인다. 이런 캐시 레코드 검색시에 각 메모리 슬롯에 저장된 인기도를 조정하여 향후의 캐시 레

```
Input:  Query_url.  /* 질의 URL. 검색결과 페이지를 유일하게 지정함. */
Output: R. /* 질의 URL에 해당하는 결과 페이지. */
1.  H_key = Hash(Query_url).  /* 해쉬 함수를 적용. */
2.  Id = H_key mod NUM_OF_LBANK. /* NUM_OF_LBANK는 논리적 메모리 뱅크수 */
3.  Found = nil. /* 초기화 */
4.  foreach  Slot ∈ LBank[Id]  /* 뱅크에는 여러 개의 슬롯이 존재. */
5.      if( Slot.h_key != H_key) then Slot.populairy--.  /* 인기도를 낮춤. */
6.      else Found = Slot;  Slot.populairy++.  /* 저장된 캐시 레코드 발견 */
7.      end_if
8.  end_for
9.  if(Found != nil) then R ← Decompress (Found.data) and exit. /* 발견된 레코드를 압축해제 하여
    결과 페이지로 사용하고 종료 */.
10. Select randomly a server in the coordinator cluster and let s be that server. /* 중계자 서버 s 선택 */
11. R ← Send(Query_url, s). /* 중계자 서버로 질의를 보내고 결과를 R에 저장. */
12. If needed, compress R and save it into a slot in LBank[Id].  /* 이때 자유 슬롯이 없다면 저장된
    인기도(popularity) 정보로 victim 슬롯을 선택하여 저장함. */
```

그림 4 메모리 CL1 캐시 관리 알고리즘

코드 교체에 이용한다.

라인 9에서 캐시 적중이 발생한 경우 해당 캐시 레코드를 사용하여 질의 결과를 보내고 그렇지 않은 경우는 중계자 서버로 질의를 보내 처리 결과를 기다린다. 중계자 서버로부터 받은 처리 결과는 라인 12에서 CL1에 저장될 수 있다. CL1의 캐시 레코드 수가 제한적이기 때문에 일정한 랭킹 범위 안에 속한 검색 결과 페이지만이 저장된다. 구현된 시스템에서는 질의 결과 페이지 중 재사용 확률이 높은 최상위 4 페이지까지 저장되도록 했다[15]. 이를 벗어나는 검색 범위들은 CL1에 캐시되지 않고, 중계자 서버의 디스크 캐시에 저장된다.

4.3.2 디스크 캐시 데이타

CL1 데이타의 경우 제한된 메모리 크기로 인해 저장되는 질의 개수나 검색범위가 매우 제한적이다. 이를 보완하기 위해 중계자 서버는 디스크에 캐시 레코드를 저장하며 이런 디스크 캐시 중 CL2와 CL3에 대해서 먼저 설명한다.

이미 언급되었듯이 그림 3의 라인 3-6과 라인 12의 작업에 많은 시스템 자원이 소모되며 작업의 결과는 라인 14-15의 중복제거 작업을 통해 마무리된다고 할 수 있다. 이를 통해 얻어진 데이타는 <DID, SRT> 집합의 형태를 가진다. CL2와 CL3 캐시 레코드는 이런 DID+SRT 데이타를 최대 DUP_K개 저장하는데 사용된다. 이 두 캐시의 차이는 저장하는 질의 종류에 있다. CL2는 고정된 인기 질의의 DID+SRT를 저장하며, 인기 질의란 사용자 질의 로그를 분석하여 얻을 수 있는 검색 빈도가 높은 질의를 말한다. 급격한 질의 패턴의 변화는 자주 발생하지 않기 때문에 이전에 자주 검색된 질의들이 일정 기간 동안에도 그러한 것이 보통이다. 따라서 인기 질의 결과를 미리 생성하여 저장함으로써 높은 캐시 적중률을 보장할 수 있다[15]. CL2의 캐시 레코드는 검색만이 가능하고 수정되지 않는 반면 CL3 캐시에서는 캐시 레코드의 생성과 삭제가 발생할 수 있다. 즉 캐시 교체 알고리즘에 의해 캐시 히트가 낮은 질의들은 캐시에서 삭제될 수 있다.

CL2와 CL3에 저장하는 SRT 데이타는 그 검색 가능 범위가 최대 DUP_K개의 상위 랭크값을 가지는 DID에 대해서이며, 이 범위를 벗어난 하위 검색범위에 대해서는 결과를 저장하지 않는다. 이렇게 한 것은 실제 사용자가 요청하는 검색 범위는 이런 상위 검색 범위에 주로 집중되고, SRT 데이타는 저장 비용이 상대적으로 크기 때문이다.

하지만 다소 빈도는 낮지만 하위 검색범위에 대해서도 캐시 데이타는 있어야 한다. 이를 고려한 것이 CL4 캐시이다. CL4는 SRT 데이타는 저장하지 않지만 2,000개까지의 상위 랭크 DID 정보를 가진다. 이 DID 리스트는 그림 3의 라인 15에서 재구성된 R에 대응된다. CL4는 인기 질의를 포함한 모든 질의들에 대해 랭크값으로 정렬된 DID를 가지며 이때 DID 리스트는 중복제거 후의 값이다. SRT 데이타가 없기 때문에 CL4에 캐시 적중이 생기는 경우에도 검색범위에 속하는 DID들의 SRT를 생성하는 비용은 발생한다. 하지만 이 비용은 그림 3의 전체 과정이 수행될 때의 비용에 비하면 그 크기가 매우 작다. 이런 이유로 CL4 캐시를 사용함으로써 적은 비용으로 성능 향상을 기대할 수 있다.

디스크 캐시인 CL2, CL3, CL4에 대한 캐시 관리 과

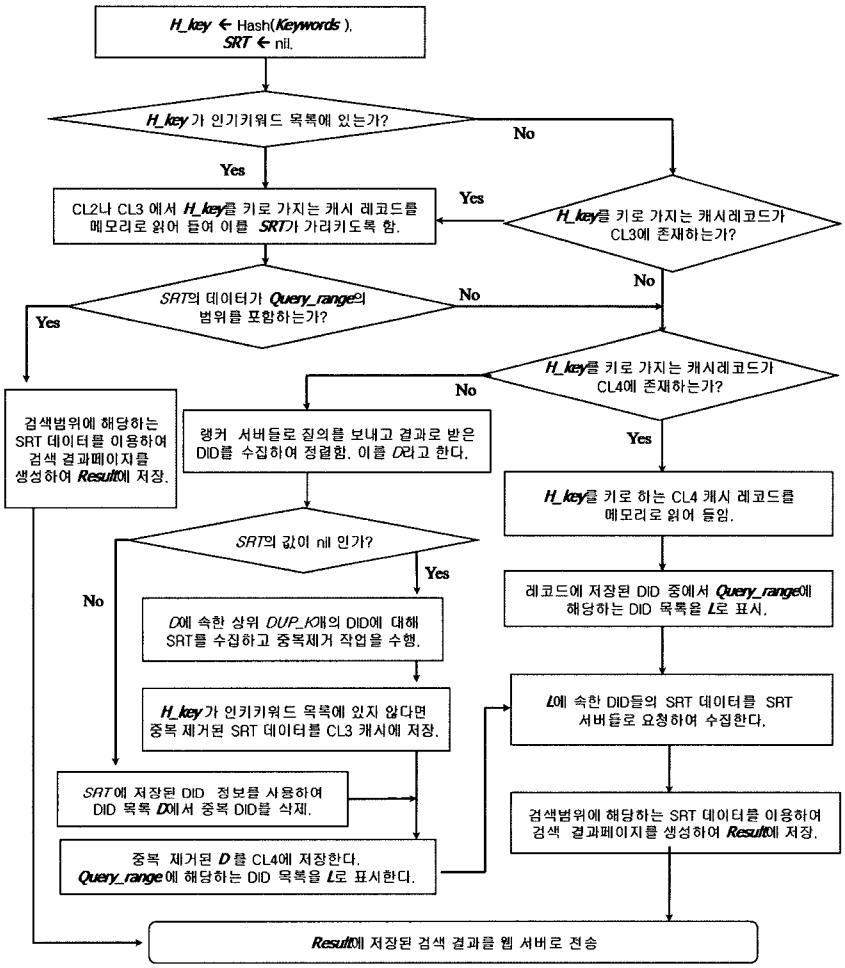

그림 5 중계자 서버의 디스크 캐시 관리 순서도

정을 그림 5에 보였다. 편의상 순서도 형태로 표현했으며, 키워드(*Keywords*) 검색범위(*Query_range*)를 사용하여 수행된다. *Keywords*에 대해 해쉬키를 생성하고, 이를 사용하여 CL2, CL3, CL4의 순서대로 관련 캐시 레코드가 있는지 검색한다. CL2나 CL3에서 캐시 적중이 생기는 경우에도, 저장된 SRT 범위가 검색범위인 *Query_range*를 포함하지 않는다면 CL4에서 캐시 레코드를 검색한다.

CL4에서도 캐시 히트가 없는 경우라면 랭커 서버들에 의한 랭킹 계산이 수행된다. 랭킹 계산이 수행된 후 중복제거 작업이 필요한데, 만약 CL2나 CL3에서 해당 질의를 캐시하고 있는 경우라면(즉, 그림 5의 변수 *SRT*의 값이 *nil*이 아닌 경우) 중복제거 작업을 위한 SRT 수집은 생략될 수 있다. 이에 대한 사항은 그림 5의 왼쪽 하단부에 보인다. 최종적으로 *Query_range*에 해당하는 SRT 데이타가 수집되고 이에 따라 검색 결과 페이지가 생성되어 변수 *Result*에 저장된다. 이 데이타는 결국 웹 서버를 거쳐 사용자에게 반환될 것이고, 질의 처리결과들은 적절히 캐시 레코드에 저장된다.

5. 성능 고찰

웹 질의 처리의 경우 디스크 및 CPU 요구량이 매우 크기 때문에 멀티쓰레드 프로그래밍, 최적화된 계산 알고리즘이 필수적이다. 이런 기술을 통해 일정 정도 자원 사용량을 줄일 수 있지만, 큰 수준으로 낮추기는 어렵다. 이런 시스템 비용 측면에 대한 가장 좋은 해결책이 캐시 기법이라 할 수 있다. 본 장에서는 이런 캐시 사용에 의한 비용 절감분을 분석하기 위해 서버 비용 모델을 사용하기로 한다.

분석에 적용되는 중요한 가정은 서버의 처리 용량을 결정하는 것은 I/O 작업이라는 것이다. 이는 실제 운영 및 예비 실험을 통해 얻어진 결과이며 시스템의 중요

서버인 랭커 서버와 SRT 서버가 I/O에 제약되는(I/O bounded) 특성에 기인한다. 비용 계산을 위해 연속된 8KB 크기의 디스크 블록을 임의 접근 했을 때의 디스크 비용을 디스크 단위 비용(UDC: Unit of Disk Costs)라 하고 서버의 중요 I/O 작업을 UDC 단위로 표시한다. UDC 정의에 따르면 디스크 읽기 비용은 읽어 들이는 디스크 데이타 크기에 단순 비례하지 않는다. 예를 들어, 16KB 크기 디스크 데이타를 읽어 들이는 비용은 이 데이타가 연속된 경우와 두 개의 분리된 디스크 블록으로 존재하는 경우 서로 다른 UDC 값을 가진다. 분리된 경우라면 2 UDC 정도의 디스크 비용이 소모되지만, 전자의 경우라면 그 비용을 1 UDC로 계산해도 무방하다. 이는 디스크 탐색 지연이 없는 경우 8KB 정도의 작은 데이타를 추가적으로 읽는 비용은 현재와 같은 고속의 디스크에서는 무시할 수준이기 때문이다[18].

아래 표 1은 UDC를 기본 단위로 했을 때 캐시 없이 질의를 처리할 때 발생하는 평균적 I/O 비용을 정리한 표이다. 표 1에서 소켓(socket)을 사용한 네트워크 비용도 I/O 비용으로 함께 고려했다. LAN을 사용한 통신의 경우 소켓 연결풀을 사용하기 때문에 소켓 생성 및 제거 비용이 없어 디스크 I/O가 발생하는 것과 비용면에서 차이가 없음으로 1 UDC로 계산했으며, 사용자와의 외부 통신 비용은 상당한 추가 비용이 있기에 이보다 높은 10 UDC로 했다. 그리고 여러 서버 단에서의 디스크 사용 비용은 실제 동작 환경에서의 실험치이다.

표 1의 비용은 하나의 AND 질의(2개 이상의 키워드로 이루어진 질의)를 처리할 때의 비용이다. 랭커 클러스터의 경우는 4개의 랭커 서버에서 발생하는 비용을 계산했고, 100개 SRT 데이타 생성을 위해 5개의 SRT 서버로 SRT 요청 메시지가 전송됨을 고려했다. 이는 구현된 시스템의 구조를 반영한 것이다.

SRT 서버 측의 I/O 비용 중에서 웹 데이타 읽기는 저장된 HTML 텍스트 데이타를 디스크에서 읽기 위한 것이며, 하나의 SRT 데이타 생성을 위해서 디렉터리 정보 및 실제 텍스트 데이타 정보를 순차적으로 읽기 위해 두 번의 디스크 블록 접근이 필요하다. 즉, 하나의 SRT 데이타 생성을 위해서 2 UDC의 비용이 발생한다. 하지만 디렉터리 정보와 HTML 데이타를 동시에 읽어 들일 수 있는 경우가 빈번히 발생하여, 실제로 2번의 디스크 접근이 생기는 경우는 전체의 32% 정도 수준이다. 이를 고려한 값이 표 1의 값들이다.

표 1에 따르면 하나의 AND 질의 처리 시에 462 UDC 정도의 I/O 비용이 필요하며, 웹 서버, 중계자 서버, 랭커 서버, SRT 서버 순으로 대략 1:1:30:14 정도로 비용이 요구됨을 알 수 있다. 주로 랭커 서버와 SRT 서버 쪽에 작업량이 많이 발생하며, 사용된 캐시 기법도 이 두 서버 단의 작업량을 줄일 수 있도록 설계되었다.

이제 캐시 히트로 인한 I/O 사용의 감소를 알아 보기 위해 간단한 확률적 분석을 수행한다. 사용자 질의 처리 시에 CL1에서 CL4까지 캐시 데이타가 검색되며 이들 각각의 캐시 레벨에서의 캐시 적중률을 p_1, p_2, p_3, p_4 라고 하자. 질의가 어떤 계층의 캐시에서든 캐시 적중이 발생할 확률을 P_{hit} 이라 하면 이는 아래 식으로 얻을 수 있다.

$$P_{hit} = p_1 + (1 - p_1) \cdot p_2 + (1 - p_1) \cdot (1 - p_2) \cdot p_3 + (1 - p_1) \cdot (1 - p_2) \cdot (1 - p_3) \cdot p_4 \quad (1)$$

계층적 캐시를 사용할 때 어느 레벨의 캐시에서 캐시 적중되었는가에 따라 질의 처리 비용의 감소 정도가 결정된다. 표 2는 각 캐시 계층에서 캐시 적중되었을 사용된 I/O 비용을 정리한 표이며, 표 1의 데이타를 기반으로 계산된 것이다. 표 2에서 A1에 해당하는 비용은 CL1 캐시에서 캐시 적중되었을 때의 I/O 비용을 나타낸다. CL2 캐시에 캐시 적중되었을 때는 A1 + A2의 비용이 사용된다. 동일한 방식으로 캐시 계층 k에서 (즉, CL_k 캐시) 캐시 적중이 발생할 때의 I/O 비용은 $\sum_i^k A_i$ 로 구할 수 있으며, C_k 로 나타낸다. 예를 들어 CL4에 캐시 히트가 발생할 때의 비용은 33 UDC가 될 것이다.

표 1 캐시없이 하나의 질의 처리를 위한 평균적 I/O 사용량

웹 서버의 I/O 비용					
사용자와의 통신 비용	10 UDC	중계자 서버와의 통신비용			1 UDC
중계자 서버의 I/O 비용					
웹 서버와의 통신비용	1 UDC	랭커 및 SRT 서버와의 통신비용			9 UDC
랭커 클러스터의 I/O 비용 (4개 랭커 서버 기준)					
중계자 서버와의 통신비용	4 UDC	동일조인 비용	100 UDC	랭킹계산비용	200 UDC
SRT 서버 I/O 비용 (DUP_K = 100, 5개 SRT 서버 기준)					
중계자 서버와의 통신비용	5 UDC	웹 데이타 읽기 비용			132 UDC

표 2 캐시 적중되었을 때의 I/O 비용

비용 이름	비용 발생 항목	I/O 비용
A_1	웹 서버/사용자 간 통신비용	10 UDC
A_2	웹서버/중계자 서버 간 통신비용	2 UDC
	CL2 캐시 레코드 읽기 비용	2 UDC
A_3	CL3 캐시 레코드 읽기 비용	2 UDC
A_4	SRT/중계자 서버 간 통신비용	2 UDC
	CL4 캐시 레코드 읽기 비용	2 UDC
	SRT 데이타 생성 비용 (10개 생성)	13 UDC

표 2의 기호와 식 (1)의 캐시 적중률 계산법에 따라 제안된 캐시를 사용했을 때의 평균비용 C_{cache}는 아래의 식과 같다. 식에서 C_{nohit}는 캐시 적중되지 않았을 때의 비용으로서 표 1에서 그 값은 462 UDC이고, 기호 $\overline{p_i}$는 $1 - p_i$를 표시한다.

$$C_{cache} = (1 - P_{hit}) \cdot C_{nohit} + p_1 \cdot C_1 + \overline{p_1} \cdot p_2 \cdot C_2 + \overline{p_1} \cdot \overline{p_2} \cdot p_3 \cdot C_3 + \overline{p_1} \cdot \overline{p_2} \cdot \overline{p_3} \cdot p_4 \cdot C_4 \quad (2)$$

캐시 사용에 의한 I/O 비용 감소비율은 $1 - C_{cache} / C_{nohit}$로 계산된다. 시스템 운영을 통해 각 계층의 캐시 적중률로 $p_1 = 0.62$, $p_2 = 0.19$, $p_3 = 0.24$, $p_4 = 0.13$을 얻었다. 이 값과 식 (1)과 (2)를 이용하면 C_{cache}의 값 103을 구할 수 있고, $1 - C_{cache} / C_{nohit}$의 값으로 0.77이 계산된다. 이는 전체 I/O 비용에서 77% 정도의 비용을 캐시 사용으로 절감할 수 있음을 의미한다.

시스템 구현에 앞서 p_1의 값을 0.5 이하로 잡았고 캐시 적중률의 범위를 70%-80% 정도로 예상했었다. 이에 따라 시스템의 안정성 차원에서 70% 정도의 캐시 적중률을 기준으로 하드웨어를 구매하는 전략을 취했다. 이 정도의 캐시 적중률에서 C_{cache} 값은 145 UDC 정도이며 이 경우 대략 70% 정도의 I/O 비용 절감을 예상할 수 있었다. 이런 예측 및 사전 실험에 따라 캐시를 사용하지 않는 경우 필요한 서버 자원 대비 30% 정도의 서버를 가지는 시스템을 구축하여 적은 비용으로 서비스를 제공할 수 있었다.

시스템 운영 전에는 CL2 캐시의 적중률인 p_2의 값을 매우 높게 예상하였는데, 실제로는 p_1에 비해 작은 값을 보였다. 하지만 CL1에서 캐시 적중되는 질의를 분석한 결과 상당수 질의가 CL2에도 있음을 알 수 있었다. 따라서 실제 적인 CL2의 캐시 적중률은 매우 높은 것이며 CL2에 캐시 적중했다면 표 2에서 보이듯이 매우 낮은 I/O 비용만으로도 질의 처리가 가능하다. 이런 특성은 실제 웹 검색 서비스 운영에 있어 매우 유용하게 사용된다. 웹 데이타를 새로운 데이타로 갱신하려고 할 때에는, CL1 캐시는 모두 삭제되어야 한다. 하지만 CL2 캐시는 이전 질의 로그를 사용하여 데이타 교체 작업 전에 미리 생성해 둘 수 있기 때문에 CL1이 삭제된 상황에서도 시스템 과부하에 의한 문제는 방지할 수 있다. 물론 데이타가 교체되는 시간은 사용자의 질의 부하가 가장 적은 새벽 시간이나 이른 오전 시간을 이용한다. 교체 후에는 CL2 데이타를 사용한 결과 페이지가 CL1에 점차 저장된다.

6. 결 론

본 논문에서는 6,500만 개의 웹 페이지를 색인하고 하루 500만개 이상의 웹 검색 질의를 처리하는 상용 웹 검색 엔진의 구현 경험을 바탕으로 시스템 구조, 색인 데이타의 형태, 질의 처리 시스템 설계시의 고려 사항, 캐시 기법에 대해서 기술하였다. 질의 처리 시스템에서 하나의 사용자 웹 질의를 처리할 때 사용되는 디스크 및 CPU 자원의 크기는 매우 크다. 만약 모든 사용자 질의 처리 시 마다 이런 자원 소모가 요구되었다면 비용 문제로 시스템 운영이 불가능했을 것이다. 이런 점을 극복하기 위해 구현 시스템에서는 계층적 캐시 데이타를 사용하였다. 구현된 기법에서는 사용 빈도나 캐시 레코드의 특성에 맞춰 캐시 데이타를 서버 별로 분산하거나 특정 계층의 캐시 데이타의 경우 메모리에 항상 적재하여 사용할 수 있다. 시스템을 운영을 통해 얻은 통계에 따르면 구현된 캐시 기법을 통해 약 4배 정도의 시스템 성능 향상을 얻을 수 있었다.

시스템 성능 향상은 캐시 적중률에 의존하게 되는데 적중률은 저장된 캐시 레코드의 개수 외에 입력되는 사용자 질의의 특성에도 크게 좌우된다. 따라서 논문에서 얻은 성능 향상치가 모든 경우에 해당되진 않을 것이다. 하지만 이런 차이에도 불구하고 구현된 계층적 캐시 기법은 시스템 성능 향상에 매우 좋은 효과를 낸다는 사실은 분명하다 하겠다. 본 논문의 구현 사례를 통해 상용 웹 검색 시스템의 구조나 특성을 보다 잘 이해할 수 있으며, 이에 따라 유사한 시스템을 구현할 때 큰 도움이 될 것이라고 생각한다.

참 고 문 헌

[1] Search Engine Report, http://www.searchenginewatch.com, 2005.
[2] Arvind Arasu, et al., Searching the Web, ACM Trans. on Internet Technology, Vol. 1(1), pp. 2-43, August 2001.
[3] Sriram Raghvan and Hector Garcia-Molina. Crawling the Hidden Web. In Proc. of the VLDB Conference, pp. 129-138, 2001.
[4] Andrei Z. Broder, Marc Najork, and Janet L. Wiener, Efficient URL Caching for World Wide Crawling, In Proc. of the 12th WWW Conference,

Budapest, Hungary, 2003.
[5] Maxim Lifantsev and Tzi-cker Chiueh, I/O-Conscious Data Preparation for Large-Scale Web Search Engines, In Proc. of the 28th VLDB Conf., pp. Hong Kong, 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 10th International World Wide Web Conference. pp. 396-406, 2001.
[7] Larry Page, Sergey Brin, R. Motwani, and T. Winograd, The PageRank Citation Ranking: Bring Order to the Web, Stanford Univ. Technical Report, 1998.
[8] Zheng Chen, Shengping Liu, Liu Wenyin, Geguang Pu, and Wei-Ying Ma, Building a web thesaurus from web link structure, In Proc. of the ACM SIGIR' 03, pp. 48-55, Toronto, Canada, 2003.
[9] C. Lee, G. Golub and S. Zenios. A Fast Two Stage Algorithm for Computing PageRank, Technical report, Stanford University, 2003.
[10] Steve Lawrence, Context in Web Search, IEEE Data Engineering Bulletin, Vol. 23(3), pp. 25-32, 2000.
[11] Reiner Kraft, Chi Chao Chang, Farzin Maghoul, and Ravi Kumar, Searching with Context, In Proc. of the WWW Conf., pp. 477-486, Edinburgh, Scotland, 2006.
[12] Taher H. Haveliwala. Topic-sensitive PageRank, In Proc. of the 11th International Conf. on World Wide Web, 2002.
[13] Maxim Lifantsev and Tzi-cker Chiueh, Implementation of a modern web search engine cluster, In Proc. of the USENIX Annual Technical Conference, Texas, 2003.
[14] Ronny Lempel and Shlomo Moran, Predictive Caching and Prefetching of Query Results in Search Engines, In Proc. of the 12th International Conf. on World Wide Web, pp. 19-28, New York, 2003.
[15] BoostingCraig Silverstein, Hannes Marais, Monika Henzinger, and Michael Moricz, Analysis of a very large web search engine query log, ACM SIGIR Forum, Vol. 33(1), pp. 6-12, 1999.
[16] Tiziano Fagni, Raffaele Perego, Fabrizio Silvestri, and Salvatore Orlando, Boosting the performance of Web search engines: Caching and prefetching query results by exploiting historical usage data, ACM Trans. on Information Systems, Vol. 24(1), pp. 51-78, 2006.
[17] Alfred V. Aho and Margaret J. Corasick, Efficient String Matching: An Aid to Bibliographic Search, Communication of the ACM, Vol. 18(6), pp. 333-340, 1975.
[18] C. Ruemmler and J. Wilkes, An Introduction to Disk Modeling, IEEE Computer, Vol. 17, No. 3, pp. 17-28, 1994.

임 성 채
1992년 서울대학교 컴퓨터공학과(학사) 1994년 KAIST 대학원 전산학과(이학석사). 2004년 KAIST 전산학과(공학박사) 2000년~2000년 7월 서울정보시스템 기술부 부장. 2000년~2005년 2월 코리아와이즈넛 연구소 수석연구원 및 이사 2005년~현재 동덕여자대학교 컴퓨터학과 조교수. 관심분야는 WEB IR, 고성능 색인기법, 데이타 마이닝