유사 질의 매칭 기반 데이터베이스 캐쉬 엔진 설계 및 구현

한윤희° 이정준
한국산업기술대 컴퓨터공학과
{yhhan°, jjlee}@jay.kpu.ac.kr

Design and Implementation of Database Cache engine based on Similarity Query Matching

Yun-Hee Han° Jeong-Joon Lee
Department of Computer Engineering, Korea Polytechnic University

요 약

인터넷 웹사이트의 급격한 증가와 함께 이용자도 증가 하고 있으며, 이용 목적은 주로 자료검색과 조회서비스 이다. 조회 요청이 많을수록 질의의 증가를 야기하며, 데이터베이스 서버의 질의 분석(Parse), 질의 실행 계획(Query Execution Plan)을 과도하게 발생시킨다. 즉 데이터베이스 서버에서 처리하는 작업량의 과부하로 인하여 병목현상을 초래한다. 데이터베이스 서버의 조회를 위한 질의처리량을 감소시키는 작업이 필요하다. 그리고 조회 대상이 데이터는 웹사이트에서 자주 갱신되지 않거나, 데이터가 주기적으로 갱신되는 특징이 있다. 이 데이터를 대상으로 데이터베이스 캐쉬 엔진을 구성하면 데이터베이스 서버의 과부하를 해소 할 수 있다. 본 논문에서는 유사 질의 매칭 기반 데이터베이스 캐쉬 엔진을 설계하고 구현한다. 유사 질의 매칭 기반으로 하여 적중률을 높여 데이터베이스 병목현상을 해결하여, 검색서비스에 더욱 효과적일 것으로 사료되며, 웹사이트의 성능 향상을 기대한다.

1. 서론

웹사이트의 증가와 함께 사용자도 꾸준히 증가 하고 있다. 사용자의 83%이상이 인터넷의 주요 목적으로 정보, 자료획득으로 조사되었다[1]. 이 지표는 조회서비스의 증가를 나타낸다. 즉, 데이터의 갱신을 요청하는 비율보다 데이터 조회 요청이 더 많이 발생한다. 이런 상황은 데이터 인출을 위해 질의처리 위한 작업의 증가를 일으켜 과부하로 인하여 데이터 갱신 요청의 지연을 초래하거나, 데이터베이스의 병목현상을 발생시킨다. 이 문제를 해결하기 위해 다음과 같은 방법들이 있다.

첫째, 추가 비용을 들여 데이터베이스를 성능 튜닝을 하거나 응용프로그램을 재개발 하는 경우가 있다. 이 방법은 경제적으로 고가의 비용이 들거나 재개발하는데 인력과 재개발 시간을 재투자해야 한다.

둘째, 기존의 서버를 대체할 대용량 데이터베이스 서버를 구매하는 경우이다. 대용량 데이터베이스 서버는 고가의 서버 구매 비용이 필요하다.

셋째, 대용량 하드웨어를 구입하여 데이터베이스 서버에 있는 데이터를 대용량 하드웨어로 복사하여 서비스하는 경우이다. 이 경우에는 데이터베이스의 모든 데이터를 복사하는데 소요되는 시간과 원본 데이터베이스와 일관성 문제가 제기된다. 대용량 하드웨어만 추가하여 처리 하는 방법에는 한계가 있다. 결과적으로 원본 데이터베이스의 모든 데이터에 대한 데이터베이스를 캐쉬로 적용하기에는 일관성 유지로 인하여 적용하기 어려운 문제점이 있다. 그래서 전체 데이터베이스를 대상으로 캐쉬를 적용하는 것이 아니라 일부 필요한 데이터만 캐쉬하는 것이 필요하다. 본 논문에서는 전체 데이터베이스에 대한 캐쉬가 아니라 조회서비스를 위한 데이터를 캐쉬 대상으로 한다.

한국인터넷진흥원에서 조사한 웹사이트를 이용하는 사용자의 주요 이용목적을 보면 자료검색 요청이 증가되는 것을 알 수 있다. 즉, 데이터의 갱신이 아닌 조회요청이 데이터베이스작업의 대부분을 차지하고 있다는 것이다. 수많은 조회서비스 요청은 데이터베이스 서버의 데이터 인출 작업을 위해 과도한 파싱작업과 쿼리 실행 계획 작업을 발생시켜 데이터베이스의 병목현상을 초래하기도 한다. 따라서 데이터베이스의 작업량을 감소시키고, 사용자의 조회서비스를 위한 데이터베이스 캐쉬 엔진이 필요하다.

주요 웹사이트(예 : 일기예보, 온라인 쇼핑몰, 음악 사이트, 신문데이터, 정책 사이트, 취업, 교육 사이트)의 특징을 살펴보면 다음과 같은 특징을 볼 수 있다.

첫째, DB의 데이터가 자주 갱신 되지 않는다. 둘째, DB의 데이터가 일정한 시점이나 주기적으로 갱신되는 것을 알 수 있다. 이러한 특징을 고려할 때 갱신이 자주 일어나지 않는 데이터나 주기적으로 갱신되는 데이터를 데이터베이스 캐쉬로 저장하여 서비스하면 데이터베이스의 병목 현상을 해결 할 수 있다.

본 논문에서는 일반적인 웹사이트 구성에서 애플리케이션 서버에 적용되는 데이터베이스 캐쉬 엔진을 설계하고 구현한다.

데이터베이스 캐쉬의 대상은 웹사이트 환경에 따라서 자주 갱신되지 않거나 주기적으로 갱신되는 데이터를 대상으로 하며 사용자의 주요 검색 대상이 되는 경우를 데이터베이스 캐쉬 대상으로 한다.

본 논문에서는 2장에서 데이터베이스 캐쉬와 관련한 연구를 살펴보고 3장에서 데이터베이스 캐쉬 엔진(DB Cache engine)의 구조와 설계를 제시한다. 4장에서는 3장에서 제시한 데이터베이스 캐쉬 엔진의 적중률을 높이기 위한 유사 질의 매칭에 대한 알고리즘을 설명한다. 5장에서는 구현환경과 구현결과를 표시하고 6장은 결론 및 향후 연구이다.

2. 관련연구

웹 애플리케이션을 위한 캐쉬 기술 중에는 웹서버와 브라우저 사이의 Proxy Cache가 있다. 이 기술은 웹페이지 전체를 캐쉬 하거나 이미지를 저장한다. 그러나 문제점은 HTML fragments를 캐쉬 할 수 없고, 데이터베이스 중심의 환경에서

는 적합하지 않고, 데이터가 다양하게 갱신되는 환경에는 적합하지 않다[2].

데이터베이스를 캐쉬의 대상으로 한 경우는 다음과 같다. MySQL 데이터베이스 서버[3]는 SQL Query Cache공간을 할당하여 데이터베이스 캐쉬를 적용하고 있다. IBM에서 개발한 DB2 데이터베이스 서버[4,5]는 Cache Tables의 개념을 적용하여 데이터베이스 캐쉬를 적용하고 있다. 오라클 사에서 만든 오라클(Oracle) 데이터베이스 서버[6]는 SQL문장에 대한 파싱(Parsing) 결과를 저장하고 SQL문장에 대한 결과를 저장하는 공간이 오라클 데이터베이스 서버 내부 메모리(System Global Area)에 할당되어 있다. 위에 제시한 데이터베이스 서버들은 데이터베이스 캐쉬 기능이 제공되고 있으나 공통적인 한계점은 데이터베이스서와 동일한 서버에서 실행되어야 한다. 즉 데이터베이스 서버 시스템에 영향을 받게 되어 데이터베이스 병목현상을 완전히 해소하는 것은 아니다. 따라서 데이터베이스 서버와 물리적으로 분리된 데이터베이스 캐쉬 구성이 필요하다. 본 논문에서는 데이터베이스 서버와 물리적으로 분리된 애플리케이션 서버에 적용되는 데이터베이스 캐쉬 엔진을 설계하고 구현한다.

오라클 데이터베이스 서버 내부에서 실행되는 데이터베이스 캐쉬를 살펴보면 SQL문장이 정확히 일치 시에만 데이터베이스 캐쉬 기능을 이용 할 수 있다. 즉, SQL문장 비교 시 대소문자가 정확히 일치 시에만 데이터베이스 캐쉬의 기능을 이용할 수 있다는 아쉬운 점이 있다. 이 기능을 보완하기 위해 쿼리 실행 계획(Query Execution Plan)이 유사한 경우는 데이터베이스 캐쉬를 이용하도록 하는 기능이 추가되었다. 이 경우는 SQL문장을 분석 한 후 쿼리 실행 계획이 세워진 상태에서 데이터베이스 캐쉬가 사용 가능한지를 판단 하게 된다.

본 논문에서는 쿼리 실행 계획을 세우기 전에 SQL문장을 비교하고자 한다. 앞에서 언급한 SQL문장의 대소문자 비교 외에도 문장을 비교하고, 문장 구조를 변경하여 같은 결과를 가지는 유사 질의와 매칭시켜 데이터베이스 캐쉬에 적용한다.

WDBAccel[7]은 e-commerce 사이트를 대상으로 한 데이터베이스 캐쉬 논문이다. SQL질의문장과 저장된 데이터베이스 캐쉬 질의 문장을 비교하고 데이터베이스 캐쉬에 대한 교체 알고리즘 제시하였다. 그러나 SQL질의 분석 기법에 대한 자세한 설계가 부족하여 본 논문에서는 유사 질의 매칭에 관한 방법 제시를 하고자 한다.

3. 데이터베이스 캐쉬 엔진(DB Cache engine) 구조

일반적인 웹사이트 시스템 환경은 그림 1과 같다. 그림 1에서 사용자(User)는 웹브라우저 프로그램을 통해 웹사이트에 접속한다. 웹서버(Web Server)는 웹페이지가 들어있는 파일을 사용자에 서비스 한다. 그 뒤에는 애플리케이션 서버와 마지막으로 데이터베이스 서버가 존재한다. 애플리케이션 서버의 기능은 미들웨어로서 트랜잭션과 비즈니스 로직을 담당한다. 데이터베이스 서버는 데이터를 관리하고, 애플리케이션 서버에서 요청한 SQL문장에 대한 처리를 담당한다.

본 논문은 일반적인 웹사이트 환경에서 애플리케이션 서버에 실행되는 데이터베이스 캐쉬 엔진을 설계한다. 그림 1에서 애플리케이션 서버와 데이터베이스 캐쉬 엔진을 연동하도록 구성하면, 그림 2와 같다. 본 논문에서는 데이터베이스 캐쉬 엔진의 설계에 대해 기술한다.

데이터베이스 캐쉬 엔진은 크게 두 부분으로 나뉜다. 쿼리 매칭기(Query Matcher) 와 캐쉬풀 관리자(Cache Pool manager)로 구성되어 있다. 캐쉬풀 관리자는 Cache Pool에 질의 처리 결과(Result Set)인 프래그먼트(Fragment)를 캐쉬 단위로 저장한다. 데이터베이스 캐쉬 엔진 구성도는 그림 3과 같다.

질의 매칭기는 사용자 혹은 상위프로그램의 질의를 받아 캐쉬풀 관리자를 이용하여 생성한 프래그먼트 혹은 원본 서버에서의 질의 처리 결과를 반환한다.

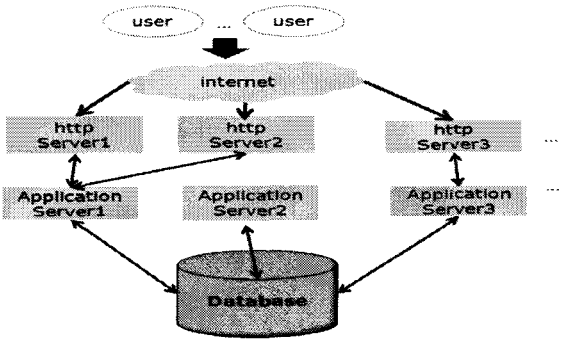

그림 1 일반적인 웹사이트 시스템

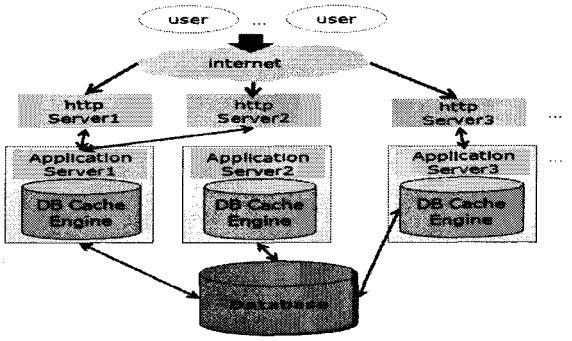

그림 2 데이터베이스 캐쉬 엔진을 이용한 웹서비스 시스템 구성

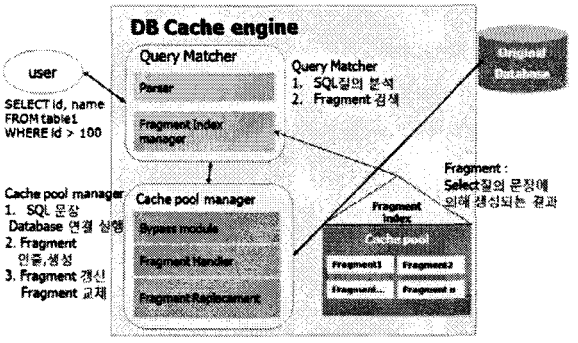

그림 3 데이터베이스 캐쉬 엔진 구성도

3.1 질의 매칭기 (Query Matcher)

질의 매칭기는 요청받은 질의를 처리하기 위해 프래그먼트를 검색한다. 이를 위해 요청받은 질의와 캐쉬풀에 저장된 프래그먼트를 생성한 질의 중 유사 질의를 검색한다.

유사 질의 검색을 위해 프래그먼트를 생성한 질의는 트라이(Trie)구조의 프래그먼트 색인(Fragment Index)에 저장된다. 그리고, 질의 매칭기는 질의를 파싱하고 생성하여 생성한 결과를 프래그먼트 색인에서 검색한다. 각 구성요소에 대해서 보다 자세히 살펴보면 다음과 같다.

• 파서 (Parser)

파서는 사용자 SQL질의에 대한 문법 체크를 한다. 오픈소스 기반의 DParser[8]를 이용한다. 먼저 SQL Grammar를 설정한 후 DParser를 이용하여 문법 확인(Syntax check)을 하고, 파스 트리(Parse tree)를 생성 해 준다.

파스 트리에서 불필요한 내용을 제거하여 생성된 추상구문트리(Abstract Syntax Tree)는 캐쉬풀에서 검색하기 위하여 프래그먼트 색인 관리자에 전달된다.

• 프래그먼트 색인 관리자 (Fragment Index manager)

파서에서 생성한 추상구문트리를 분석하고 SQL질의를 매칭한다. 매칭 하는 알고리즘은 4장 유사 질의 매칭에서 자세히 설명한다. 먼저 SQL문에서 Select 문 이외에는 질의 결과 결과를 생성하지 않으므로, 캐쉬의 대상이 아니다. 따라서 Select 이외의 SQL문은 프래그먼트 색인 관리자를 통하지 않고, 바로 캐쉬풀 관리자에 전달되어, 바이패스 모듈을 통하여 직접 원본 데이터베이스에서 실행되도록 한다.

캐쉬의 대상이 되는 Select 문은 파서에서 생성한 추상구문트리를 기준으로 Select절과 From절과 Where절로 분리 하여 그림 4의 형태로 저장한다. 그림 4에서 각 사각형은 하나의 클래스(Class)를 의미한다. QPFrom 클래스는 From절에 의해 QPFrom이 생성된다. QPSelect 클래스는 Select절에 의해 QPSelect가 생성된다. QPWhere 클래스는 Where절에 의해 QPWhere, RWhere, Where 클래스를 생성한다.

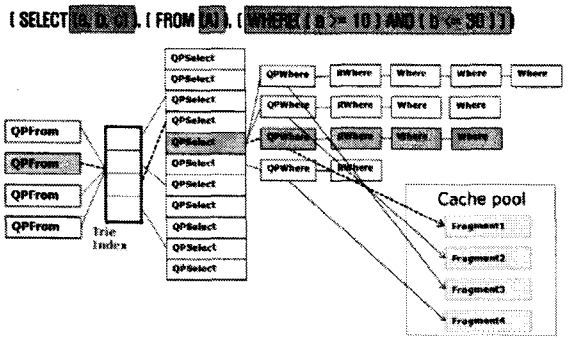

그림 4 SQL질의 저장 구조

먼저 질의 매칭의 흐름을 설명하면 다음과 같다. SQL문의 From절에서 동일한 테이블을 사용하는지 먼저 확인하고, Select절의 조회 컬럼에 따라 구성된 QPSelect절에서 요청한 질의의 조회 컬럼이 QPSelect 컬럼에 포함되는지를 확인하다. 그리고 조건들의 논리곱 형태(conjunctive normal form)을 저장된 QpWhere에 요청된 질의의 조건이 포함되는지를 검토한다. 포함되는 질의가 있으면 이에 해당하는 프래그먼트를 찾아 프래그먼트에서 질의 조건에 해당하는 행을 추출하고, 필요한 조회 컬럼만을 선택하여 반환한다. 질의 매칭의 흐름도는 그림 5와 같다.

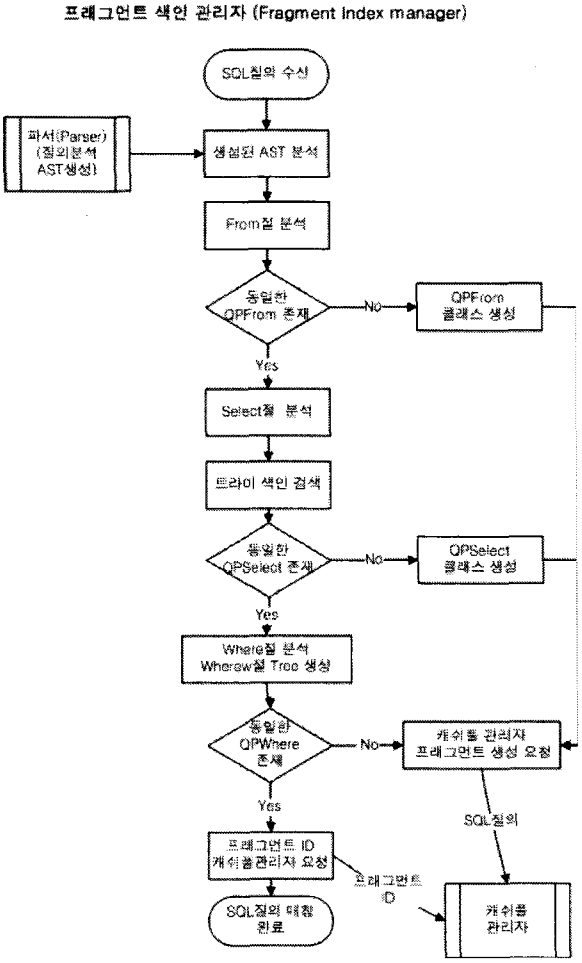

그림 5 질의 매칭 흐름도

이러한 매칭과정을 효율적으로 지원하기 위한 프래그먼트 색인의 구조를 살펴본다. 요청받은 질의는 From절에서 사용한 테이블들(뷰 포함)들에 따라 QPFrom절이 결정된다. 즉 동일한 테이블들을 사용한 Select문은 동일한 QPFrom 클래스에 속한다. 그리고 QPSelect는 조회하는 컬럼들에 의해 결정된다. 따라서, 동일한 QPFrom으로 구성된 문장은 수많은 형태의 QPSelect 클래스 갖는다. 그 이유는 테이블에서 조회할 수 있는 컬럼의 수는 여러 형태가 될 수 있기 때문이다. QPSelect 클래스는 많은 수의 클래스로 존재한다. 유사 질의 매칭을 하기 위해서는 QPFrom 클래스, QPSelect 클래스, QPWhere 클래스가 일치해야 한다. 수 많은 QPSelect 클래스 중에서 원하는 QPSelect 클래스를 빠르게 찾기 위한 색인이 필요하다. 이때의 색인 구조는 사전 찾기 색인방식인 트라이 색인(Trie Index)구조를 형성한다.

트라이 검색 구조는 그림 6에 표현되어 있다. 노드의 구성은 Select ID를 찾기 위한 컬럼의 값, 색인 결과 값인 Select ID, 자식 노드(Child node)를 가리키는 포인터, 형제 노드(Sibling node)를 가리키는 포인터로 구성되어 있다. 그림 6은 Q1, Q2,

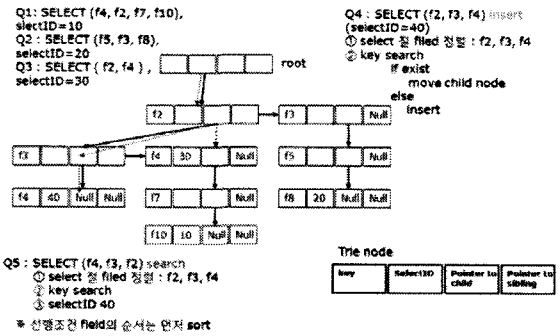

그림 6 QPSelect 클래스를 위한 트라이 색인 구조

Q3, Q4의 Select절의 컬럼 값이 저장되어 있고, Q5의 Select 절과 같은 문장이 있는지 검색하는 과정이다.

Where절은 SQL질의 비교를 위해 내부적으로 트리 형태로 저장된다. 그림 7의 문장(SELECT a, b, c FROM A WHERE (a >= 10) and (b<= 30)) 에서 QPFrom 클래스는 테이블(Table) 명인 A를 저장하고, QPSelect Class는 컬럼 a, b, c를 저장한다. Where절은 a >= 10 라는 조건과 b <= 30 조건을 Where 클래스에 저장되어 트리형태의 구조를 가지고 있다. 트리의 논리적 구조 형태는 그림 7에 나타나 있다. Where절은 캐쉬 결과인 프래그먼트 ID 를 가지고 있다.

select a, b, c from A where (a >= 10) and (b <= 30)

WHERE(and,(>=,(a,10)),(<=(b,30))

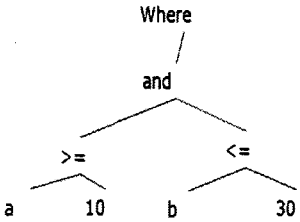

그림 7 Where절 트리 논리적 구조

프래그먼트 ID 부여는 캐쉬풀 관리자에서 프래그먼트 ID를 생성하여 질의 매칭기 에게 넘겨주면 QPWhere 클래스에 저장된다. 위에서 설명한 과정을 거쳐 SQL질의 문장은 분석되고 저장된다.

프래그먼트 색인 관리자는 요청한 SQL질의와 Fragment로 생성되어 저장된 SQL질의 문장들에 대한 검색을 시도한다. 유사한 문장을 찾은 경우는 Fragment Hit 라고 하고, 동일한 질의 문장이 없는 경우는 Fragment Miss 이다.

Fragment Hit인 경우는 사용자 SQL질의와 프래그먼트에 존재하는 SQL질의가 문장이 일치하지 않아도 같은 결과를 생성하는 경우이다. 유사 질의매칭을 사용하지 않을 경우에는 새로운 프래그먼트를 생성해야 하지만 유사 질의 매칭을 사용하면 새로운 프래그먼트를 생성하지 않아도 되기 때문에 캐쉬풀 관리자에게 프래그먼트 ID를 전달하여 해당하는 Fragment에 대한 결과 값을 전달 받아 사용자 혹은 상위 프로그램에게 결과 값을 전달한다.

Fragment Miss인 경우는 사용자 질의와 프래그먼트에 존재하는 질의의 문장이 일치하거나 유사하지 않은 경우로서 사용자 질의에 대한 결과를 프래그먼트로 생성 할 수 있도록 캐쉬풀 관리자에게 사용자 질의를 전달한다. 캐쉬풀 관리자는 프래그먼트 생성 후 프래그먼트 ID를 새로 부여 한 후 질의 매칭기에 전달하게 되면 최종적으로 QPWhere 클래스에 프래그먼트 ID가 저장되게 된다.

3.2 캐쉬풀 관리자 (Cache Pool Manager)

캐쉬의 대상이 아닌 경우에 바이패스 모듈을 통해 원본 데이터베이스 서버와 연결하여 사용자SQL을 실행시키고 결과 값을 전달받는다. 프래그먼트 핸들러(Fragment Handler)는 프래그먼트를 생성하거나, 생성된 프래그먼트의 만기시간을 적용하여 필요한 경우에 갱신하고, 프래그먼트 결과 값을 질의 매칭기로 인출하여 전달한다. 프래그먼트 교체(Fragment Replacement)는 Cache Pool의 용량을 초과했을 경우 생성된 프래그먼트를 삭제하기 위한 모듈이다.

• 바이패스 모듈 (Bypass module)

캐쉬의 대상이 아닌 경우에 바이패스 모듈을 사용한다. 사용자 SQL문장을 질의 매칭기로부터 전달 받아 원본 데이터베이스 서버에 연결 실행한 후 결과를 캐쉬풀 관리자를 통해 질의 매칭기에 전달한다.

• 프래그먼트 핸들러 (Fragment Handler)

질의 매칭기에서 결정된 상황에 따라 실행이 달라진다. Fragment Hit인 경우와 Fragment Miss의 경우로 나뉜다.

Fragment Hit인 경우는 질의 매칭기에서 전달 받은 프래그먼트 ID에 해당하는 프래그먼트를 검색하고 인출하여 질의 결과 값을 질의 매칭기로 전달한다. 이 결과 값을 전달하기 전에는 프래그먼트 생성 시간과 현재 시점을 비교한다. 미리 설정된 만기시간을 요청시간 기준으로 초과했을 경우는 프래그먼트 질의를 재실행 한다. 즉 원본 데이터베이스 서버의 실행 결과를 프래그먼트 재생성 후 질의 매칭기로 전달한다.

캐쉬풀에 저장된 프래그먼트 예는 그림 8과 같다. 만기시간을 3분이라고 설정, 현재 10시 5분이라고 가정한다. 질의 매칭기가 프래그먼트 ID 1 요청 시 생성 시점으로부터 만기시간 5분을 초과하였으므로 원본 데이터베이스 서버로부터 데이터 재실행 후 질의 매칭기로 전달하고 요청횟수(Hitcnt)를 증가한다. 요청횟수는 프래그먼트 요청 시마다 1씩 증가한다. 또 다른 예는 다음과 같다. 만기시간과 현재시간이 같은 조건이고, 프래그먼트 ID 4 요청 시 프래그먼트 ID4는 10시 3분에 생성되었으므로 만기시간을 초과하지 않았다. 그러므로 프래그먼트 ID 4는 갱신 없이 그대로 인출하여 질의 매칭기에 전달하고 요청횟수를 증가 한다.

Fragment Miss인 경우는 질의 매칭기로부터 전달받은 사용자 질의를 원본 데이터베이스 서버에서 실행하여 결과 값을 프래그먼트로 생성한 후 프래그먼트 ID 를 부여하여 ID와 프래그먼트 결과를 캐쉬풀 관리자를 통해 질의 매칭기로 전달한다.

각 프래그먼트는 SQL질의, 프래그먼트 ID, 생성시간, 갱신 시간, 프래그먼트 요청횟수, 프래그먼트 사이즈를 가지고 있다. 이 정보는 프래그먼트를 생성시킨 질의의 재실행 여부의 기준이며 프래그먼트 교체 시 정보로도 사용된다.

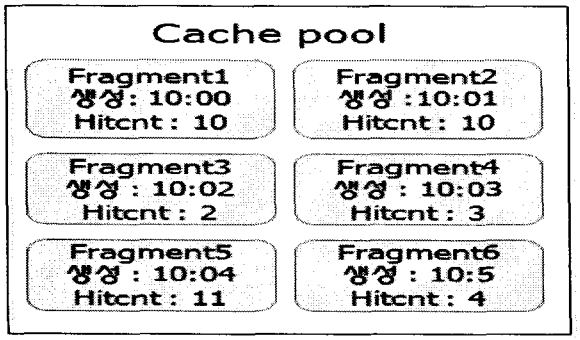

그림 8 Cache Pool에 저장된 Fragment의 예

• 프래그먼트 교체 (Fragment Replacement)

캐쉬풀의 용량을 초과했을 경우 기존 프래그먼트를 삭제하기 위한 모듈이다. 프래그먼트 교체 모듈은 생성된 프래그먼트들 중에서 삭제할 대상을 선정한다. LRU, LFU, Function 기반 등 다양한 방법이 있으나 본 논문에서는 만기시간과 교체전략으로 주로 이용되는 LFU를 이용하였다[9,10]

첫 번째 기준으로 만기시간이 경과된 프래그먼트가 존재할 경우를 우선순위로 하여 프래그먼트를 삭제 대상으로 한다. 만약 만기시간 경과된 프래그먼트가 존재하지 않을 경우에는 두 번째 기준으로 LFU(Least Frequently Used)를 적용한다. 즉 프래그먼트에 저장된 요청횟수가 적은 것을 삭제 대상으로 한다.

4. 유사 질의 매칭

Select문장으로 이루어진 SQL문장은 캐쉬의 대상이다. Select문장에 대한 분류를 한다. SQL문장에서 Where절이 and 연산으로 이루어진 경우와 Where절이 or 연산으로 이루어진 두 가지의 경우로 분리한다.

• SQL질의 유사 질의 매칭 (and 연산자 경우)

표 1의 두 개의 질의는 동일한 결과를 가지는 질의의 예이다. From절, Select절, Where절을 각각 분석 비교하여 동일한 문장일 경우에는 유사 질의로 판단한다.

첫째, From절을 먼저 분석한다.

Q1 : table1 Q2 : table1

둘째, Select절을 분석한다. Select절을 분석할 때는 컬럼의 순서로 저장하는 것이 아니라 먼저 컬럼의 이름으로 먼저 소팅을 한다.

Q1 : a , b 로 되고, Q2 : a, b

셋째, Where절을 분석 한다. Where절 에서는 연산자(피연산자,피연산자) 형태로 갱신 한 후 피연산자 문자열 정렬하여 필요 시 피연산자 교환한다.

(1) 연산자(피연산자,피연산자) 순서로 변경

Q1 : and(>(a,10),<(b,20))

Q2 : and(>(b,20),>(a,10))

(2) 피연산자 교환

아래의 예에서는 Q1은 a와 b 순서로 되어 있어 변경 필요 없고, Q2에서 b, a 순서로 되어 있기 때문에 a, b 순서로 변경한다.

Q1 : and(>(a,10),<(b,20))

Q2 : and(>(a,10),>(b,20))

표 1 유사 질의 예

Q1	select a, b from where table1 a > 10 and b > 20;
Q2	select b, a from where table1 b > 20 and 10 < a;

• SQL질의 유사 질의 매칭 (or 연산자 경우)

표 2의 두 개의 질의는 동일한 결과를 가지는 질의의 예이다.

첫째, From절을 먼저 분석한다.

Q1 : table1 Q2 : table1

둘째, Select절을 분석한다. Select절을 분석할 때는 컬럼의 순서로 저장하는 것이 아니라 먼저 컬럼의 이름으로 먼저 소팅을 한다.

Q1 : a , b 로 되고, Q2 : a, b

셋째, Where절을 분석한다. Where절에 or 연산자가 있을 경우는 먼저 and 연산으로 변경 한다. 드모르간 법칙을 이용하여 변경한다.

(1) or 연산자를 and 연산자로 변경한다.

Q1 : not(b >= 20 and a <= 10)

Q2 : not(a <= 10 and b <= 20)

(2) 연산자(피연산자,피연산자) 순서로 변경

Q1 : not(and(>=(b,20), <=(a,10)))

Q2 : not(and(<=(a,10), <=(b,20)))

(2) 피연산자 교환

Q1 : not(and(<=(a,10), <=(b,20)))

Q2 : not(and(<=(a,10), <=(b,20)))

Where절 까지 분석한 후 두 문장을 비교해 보면 Q1과 Q2문장은 같은 결과를 가지는 질의이다.

표 2 유사 질의 예

Q1	select a, b from table1 where b > 20 or a > 10
Q2	select a, b from table1 where not(a <= 10 and b <= 20)

5. 구현환경 및 구현 결과

5.1 구현환경

운영체제는 Linux (CentOS 5.0) 환경에서 데이터베이스는 MySQL 5.0을 사용 하였다. 데이터베 베이스 캐쉬의 구성단위인 Fragment는 파일로 처리 하여 인출하거나 삭제 하였다.

개발환경은 Cpu는 intel(R) Pentium(R) 4 CPU 2.8GHz 이고 Memory는 512MByte이며, C, C++를 이용하여 개발 하였고 gcc 컴파일러 버전은 3.4.6 이다.

5.2 구현 결과

파서를 통한 SQL질의 분석 후 추상 구문 트리는 그림 9와 같다. 그림 9에서 추상구문트리를 출력했으며, QPFrom 클래스, QPSelect 클래스, QPWhere클래스를 생성한 결과이다.

SQL질의 유사 질의 매칭 결과는 그림 10 이다. Query[2]번과 Query[3]번 문장은 다른 문장이지만 유사 질의 매칭 과정 후 같은 결과를 가지는 질의임을 알 수 있다. Query[2]문장은 새로운 질의로 인식되어 프래그먼트 ID 1이 생성 된다. 그 후 Query[3]의 문장은 Query[2]의 결과와 동일한 문장으로 인식되어 프래그먼트 ID 1로 인식된다.

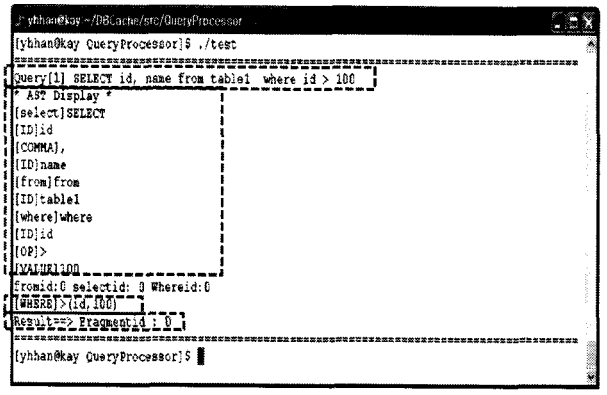

그림 9 파서에서 생성한 추상구문트리

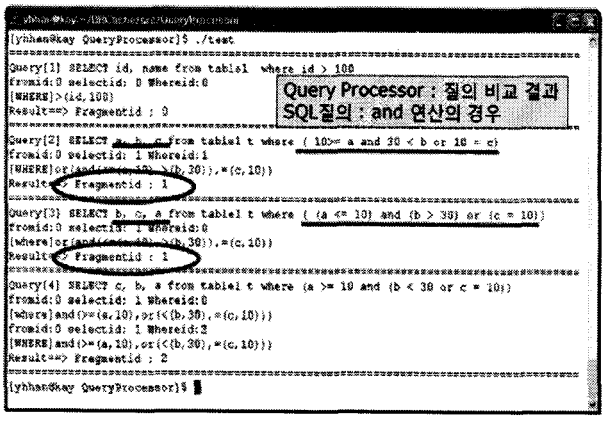

그림 10 SQL질의 유사 질의 매칭(and 연산자 경우)

6. 결론 및 향후 연구

웹사이트 증가와 더불어 이용자도 증가하고 있다. 그리고 사용자들의 조회서비스 증가는 데이터베이스 서버의 SQL질의 수행을 위한 작업의 과부하를 발생시킨다. 본 논문에서는 유사 질의 기반 데이터베이스 캐쉬 엔진을 설계하고 구현하여 문제를 해결하고자 한다.

조회 대상이 되는 데이터는 변경이 자주 갱신되지 않거나 주기적으로 갱신된다는 특징을 기준으로 데이터베이스 캐쉬의 대상으로 하였다.

향후 연구는 유사 질의 매칭 범위를 확장하여 다양한 유사 질의 매칭을 기법을 제시한 후 구현하고자 하며 다양한 실험계획을 세워 평가하는 것이다. 그리고 데이터베이스 캐쉬 교체 전략인 만기 시간 방식을 다양화하여 알고리즘을 제시하고자 한다.

7. 참고문헌

[1] 한국인터넷진흥원, 인터넷 백서 2007 http://isis.nida.or.kr

[2] "Tutorial : Caching Technologies for Scaling and Speeding Up Web Applications", Dr.C.Mohan ,IBM insight Exchange Webcast, 2003

[3] MySQL 데이터베이스 서버 : http://dev.mysql.com/doc/refman/4.1/en/

[4] Christof Bornhövd, Mehmet Altinel, C. Mohan, Hamid Pirahesh, Berthold Reinwald: Adaptive Database Caching with DBCache. IEEE Data Eng. Bull. 27(2): 11-18 ,2004

[5] M.Altinel, C.Bornhövd,S.Krishnamurthy,C.Mohan,H.Pirahesh,B.Renwald : Cache Tables: Paving the Way for an Adaptive Database Cache, VLDB'03, berlin, germany, 2003.

[6] 오라클(Oracle) 데이터베이스 서버 : http://download.oracle.com/docs/cd/A91202_01/901_doc/server.901/a90117.pdf

[7] Seunglak Choi, Jinwon Lee, su Myeon Kim, Junehwa Song, and Yoon_Joon Lee, Accelerating Database Processing at e-Commerce Sites, E-Commerce and Web Technologies 5th International Conference, EC-Web 2004, Zaragoza, Spain, 2004.

[8] Dparser http://dparser.sourceforge.net/

[9] S.Podlipnig, L.Böszörmenyi: A Survey of Web Cache Replacement Strategies, ACM Computig Surveys, Vol. 35, No.4, pp.374-398, 2003.

[10] 최승락,김미영,박창섭,조대현,이윤준, 웹 프락시 서버를 위한 적응형 캐시 교체 정책, 정보과학회논문지, 시스템 및 이론 제 29권, 2002.