LOB 캐쉬를 위한 SQL CLI의 확장

이 종 민[†]·강 현 철[††]

요 약

SQL CLI(Call Level Interface)는 클라이언트-서버 환경에 적합한 데이타베이스 응용 프로그래밍 인터페이스(API)로서 ODBC, JDBC 등의 업계 표준이 여러 응용 분야에 걸쳐 널리 사용되고 있다. 그러나 현재의 표준안에서는 멀티미디어 데이타와 같은 대용량 데이타를 효율적으로 검색하기 위한 기능을 다양하게 제공해주지 못하고 있다. 본 논문에서는 멀티미디어 데이타를 구성하는 LOB(Large OBject)의 효율적 검색을 위하여 SQL CLI 상에서 LOB의 캐쉬를 제안하고, 이를 위한 SQL CLI의 확장을 제안한다. 제안한 내용을 한국전자통신연구원에서 개발한 바다-II DBMS를 위한 SQL CLI 라이브러리 상에 구현하며, 실험을 통하여 캐쉬로부터의 LOB 검색 성능을 기존의 SQL CLI 표준 사양에 따른 LOB 검색의 성능과 비교 평가한다.

키워드 : SQL CLI, LOB, 클라이언트 데이타 캐싱

Extending SQL CLI for Large Object Caching

JongMin Lee[†]·Hyunchul Kang[††]

ABSTRACT

The SQL CLI (Call Level Interface) is a database application programming interface (API) that fits the client-server environment, and its de facto standards such as ODBC and JDBC are widely employed in various applications. The current SQL CLI standards, however, do not specify enough features for efficient retrieval of large objects (LOBs) that constitute the multimedia data. In this paper, we propose the LOB caching through the SQL CLI and describe the extension of the SQL CLI to achieve such a goal. We implement our proposal by extending the SQL CLI library for BADA-II DBMS developed at ETRI, and evaluate the performance of LOB retrieval through the cache compared to that of LOB retrieval conducted solely with the functions specified in the current SQL CLI standards.

Key word : SQL CLI, LOB, client data caching

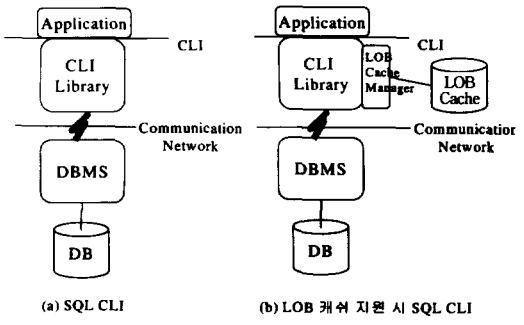
1. 서 론

근래의 컴퓨팅 환경은 클라이언트-서버 소프트웨어 구조가 주류를 이루고 있다. 데이타베이스 시스템도 기존 중앙 집중 환경에서 운용되는 것보다 망 환경에서 데이타 서버로서 클라이언트의 응용 프로그램을 지원하는 기능이 중요하게 되었다. 데이타베이스 시스템이 제공하는 응용 프로그래밍 인터페이스(API)들 중 클라이언트-서버 환경에 적합한 것으로 SQL Call Level Interface(이하, CLI)가 있다 [1-5].

CLI는 클라이언트의 응용 프로그램이 서버의 데이타베이스에 연결되어 SQL 명령어를 수행하고, 그 결과를 받아보기 위한 함수들로 구성된 인터페이스이다. (그림 1-a)에서 보듯이 CLI를 지원하는 DBMS는 자신의 SQL 처리기의 기능을 호출하는 CLI 함수 라이브러리를 갖추고 있다. 이는 응용 프로그램의 실행 시에 링크 및 적재(load)되는 실행 시간(run time) 라이브러리이다. 따라서 응용 프로그램은 데이타베이스 접근이 필요할 때 해당 CLI 함수를 호출하고 호출된 CLI 함수는 이 라이브러리를 통하여 수행된다. 따라서 CLI의 표준화가 전제된다면 응용 프로그램은 목적(target) DBMS에 상관없이 독립적으로 개발이 가능하다. 이러한 특성으로 인해 CLI를 이용하여 작성된 응용 프로그램은 동일한 사양의 CLI를 지원하는 서로 다른 DBMS들 간에 실행 모듈 수준의 호환성(binary compatibility)을 갖는다. X/Open에서는 1992년 SQL Access Group과 함께 CLI 표준 사양을 제정 발표하였고[2], ISO/ANSI에서도 1993년 표준 사양을 제정하였다[3]. 업계 표준으로는 Microsoft의 ODBC[4]와 Sun의 JDBC [5]가 여러 응용 분야에 걸쳐 널리 사용되고 있다. ODBC는 데이타베이스 연결을 위한 C 인터페이스로서 API 함수들은 크게 core, level 1, level 2 등으로 분류되고, 지원되는 SQL 문법도 minimum, core, extended로 등급을 분류하여 ODBC 인터페이스의 준수 수준(conformance level)을 규정하고 있다. X/Open, ISO

※ 본 논문은 정통부의 정보통신 우수시범학교 지원사업에 의한 것임.
† 준 회 원 : 중앙대학교 대학원 컴퓨터공학과(공학석사)
†† 정 회 원 : 중앙대학교 컴퓨터공학과 교수
논문접수 : 2000년 10월 30일, 심사완료 : 2001년 1월 30일

등의 표준화 기구에서 발표한 CLI 사양은 대체로 ODBC의 core level에 해당된다. JDBC도 목적 DBMS에 무관한 데이타베이스 연결을 제공하기 위한 Java 인터페이스인데 JDBC 드라이버는 ODBC 드라이버와 연동하기도 한다.



(그림 1) SQL CLI와 LOB 캐쉬

현재 데이타베이스 서버로부터 이미지, 동영상 등과 같이 멀티미디어 데이타를 구성하는 LOB(Large OBject)을 검색하는 응용은 계속 증대하는 추세에 있으며, 이에 따라 데이타베이스 응용 프로그래밍의 기능으로서 더욱 효율적이고 신속한 LOB 검색을 지원해 줄 필요가 있다.

기존의 CLI 응용 프로그램에서는 LOB을 검색하기 위하여 X/Open이 명시한 CLI 표준의 경우, GetCol이라는 CLI 함수(ODBC의 경우, GetData)를 호출한다. GetCol 함수는 클라이언트 응용 프로그램의 작업 영역에 주어진 버퍼의 크기 제약으로 인해 한번의 호출로 서버로부터 LOB 전체를 검색하지 못할 경우, 여러 번 반복 호출되어 LOB의 연속된 부분을 차례로 검색한다.

사용자는 자신의 편의에 맞도록 클라이언트 시스템(예를 들어, 데스크탑 또는 노트북 PC)의 환경을 구성하고 사용할 뿐 아니라 검색하는 LOB은 관심 영역에 따라 전체 데이타베이스 중 비교적 국한된 부분에 속할 수 있다. 즉, 동일한 LOB의 검색을 반복하는 경우가 많다. 또한 이미지, 비디오 등과 같은 LOB은 일단 데이타 서비스가 시작되면 그 내용이 자주 변하지 않는 특성을 가지고 있다. 이와 같은 점들과 LOB의 용량을 고려할 때 사용자는 자주 접근하는 LOB을 자신의 PC 환경으로 캐쉬함으로써 더 효율적이고 신속한 LOB 검색을 수행할 수 있으며 그 효용성이 기존의 일반 정형 레코드의 캐쉬에 비해 훨씬 크다 하겠다.

(그림 1-b)에서 보듯이 기존 CLI 라이브러리가 확장되어 캐쉬에/로부터 LOB을 저장, 검색, 삭제하는 기능 등을 수행하는 LOB 캐쉬 관리자(LOB Cache Manager)가 지원된다면, 응용 프로그램은 (보통 망을 통해) 데이타베이스 서버로부터 LOB을 검색하는 대신 응용 프로그램이 수행되는 클라이언트 시스템의 캐쉬로부터 LOB을 검색할 수 있다.

본 논문에서는 윈도우 환경의 PC에서 널리 사용되고 있는 데이타베이스 API인 SQL CLI를 확장하여 LOB의 캐쉬를 수행할 수 있는 기법을 제안하고, 이를 위한 CLI의 확장을 제시한다. 그리고 제안한 확장 내용을 표준 CLI를 지원하는 실제 DBMS 상에서 구현하고 LOB 캐쉬의 성능을 평가한 결과를 기술한다.

클라이언트-서버 데이타베이스 환경에서 서버로부터 데이타를 효율적으로 검색하기 위한 방법으로 클라이언트 데이타 캐싱은 지금까지 많이 연구되어 왔다[6]. 이들 연구에서는 클라이언트에 캐쉬된 데이타의 일관성 유지 기법을 주로 다루고 캐쉬로 인한 성능 트레이드오프(tradeoff)를 연구하였다. 그러나 CLI 상에서 LOB 캐쉬를 지원하기 위한 연구는 아직까지 수행된 적이 없다. 위에서 열거한 CLI의 어떠한 표준 사양에서도 LOB 캐쉬를 지원하기 위한 기능이 명시되어 있지 않다.

본 논문의 구성은 다음과 같다. 2절에서는 CLI에서 LOB 캐쉬를 지원하기 위해 해결해야 할 기술적 문제점에 대하여 기술한다. 3절에서는 기존의 표준 CLI 사양을 LOB 캐쉬를 지원할 수 있도록 확장하기 위한 방안을 제안한다. 4절에서는 제안된 CLI 확장의 구현을 기술하고, 5절에서는 LOB 캐쉬/검색의 성능 평가 결과를 기술하며, 6절에서 결론을 맺는다.

2. SQL CLI에서 LOB 캐쉬의 지원

본 절에서는 LOB 캐쉬를 CLI 상에서 지원해 주기 위해 해결되어야 할 기술적인 사항에 대하여 다룬다. CLI 상에서 LOB 캐쉬를 지원하기 위해서는 다음과 같은 사항이 해결되어야 한다.

- 서버 측에 저장된 LOB과 캐쉬된 LOB 간의 일관성 유지
- LOB 캐쉬 교체 알고리즘
- LOB 캐쉬 정보의 효율적 관리
- LOB 캐쉬를 지원하기 위한 새로운 CLI 함수

이들에 대해 각각 설명하면 다음과 같다.

첫째, 캐쉬에 저장된 LOB은 서버 측에 저장된 LOB과 일치해야만 유효하다. 서버에 저장되어 있는 LOB은 갱신될 수 있기 때문에 클라이언트 측에 캐쉬된 LOB을 갱신 여부의 확인 없이 사용한다면, 최신의 것이 아닌 내용의 LOB을 접근하는 결과를 가져올 수 있다. 일관성 유지를 위한 방법은 크게 서버 중심 방법과 클라이언트 중심 방법으로 나눌 수 있다. 서버 중심 방법은 서버가 클라이언트들의 LOB 캐쉬에 대한 정보를 가지고 있어 서버에서 LOB이 갱신 될 때마다 해당 클라이언트들에 계속 알려주는 방법이다. 이 방법은 서버가 클라이언트의 모든 캐쉬 정보를 관리하므로 망 환경에서 수많은 클라이언트가 서버에 접속할 수 있다는 점을 감안할 때 서버에 많은 부하를 지우게 된다. 클라이언트 중심 방법은 클라이언트가 LOB을 검색하고자 할

때 캐쉬 적중(cache hit)의 경우 캐쉬에 저장되어 있는 LOB이 유효한 지의 여부를 서버에 접근하여 판단하는 것이다. 이 방법은 원하는 LOB이 캐쉬되어 있더라도 LOB을 검색하기 전 캐쉬 유효성 여부를 서버에 접근하여 확인하는 작업을 필요로 하지만 서버의 부하를 많이 덜게 된다.

둘째, 새로운 LOB을 캐쉬하고자 하는데 LOB 캐쉬 저장 공간(캐쉬 영역)이 가득차 있을 경우에는 일부 LOB을 교체해야 한다. 전통적인 DBMS에서 수행하는 페이지 단위의 버퍼 교체에 비해 LOB은 크기가 크고 다양하므로 LOB 교체 알고리즘은 이러한 점을 고려해야 한다. 예를 들어 하나의 큰 LOB을 캐쉬하기 위하여 작은 크기의 LOB을 여러 개 교체하거나 여러 개의 작은 크기 LOB을 캐쉬하기 위해 하나의 큰 LOB을 교체할 수 있다.

셋째, LOB 캐쉬 정보는 크게 LOB 캐쉬 영역에 관한 정보, 캐쉬의 효율적 검색을 위한 정보, 캐쉬된 각 LOB에 대한 정보, 캐쉬된 LOB의 통계 정보 등으로 나눌 수 있다. 이러한 정보가 LOB의 신속한 검색 및 교체를 위하여 효과적으로 유지될 필요가 있다.

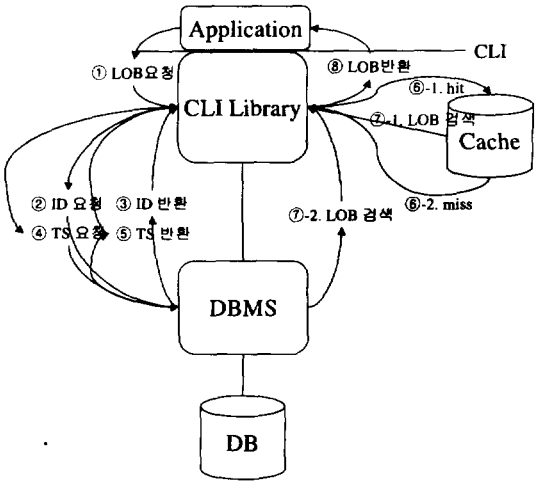
넷째, LOB 캐쉬 영역을 할당 또는 해제하고, 데이타베이스 서버로부터 LOB을 캐쉬하거나 캐쉬된 LOB을 캐쉬 영역에서 삭제하고, LOB 캐쉬 정보를 조작할 수 있는 새로운 CLI 함수가 제공되어야 한다.

3. SQL CLI의 LOB 캐쉬 확장 방안

본 절에서는 SQL CLI에 LOB 캐쉬를 지원하기 위한 확장 방안을 제안한다. 3.1절에서는 LOB 캐쉬 일관성 유지에 대해서 기술한다. 3.2절에서는 LOB 캐쉬 교체 정책에 대해 기술한다. 3.3절에서는 LOB 캐쉬 정보의 효율적 관리에 대해 기술하며, 3.4절에서는 LOB 캐쉬를 위한 새로운 CLI 함수에 대해 기술한다.

3.1 LOB 캐쉬 일관성 유지

일관성 유지는 CLI 상에 LOB 캐쉬를 지원하기 위해 필수로 충족되어야 하는 조건이다. 본 논문에서는 클라이언트 중심의 일관성 유지 방안을 채택하였다. 즉, 검색하고자 하는 LOB이 캐쉬에 있을 경우(캐쉬 적중) 그것의 유효성을 서버에 확인한 후, 유효할 경우에는 캐쉬로부터 LOB을 검색하고 그렇지 않을 경우에는 서버로부터 LOB을 검색 (및 캐쉬)하게 된다. 이를 위해 필요한 사항은 LOB의 식별자(ID)와 LOB이 서버에서 마지막으로 갱신된 시점을 나타내는 타임스탬프(TS)이다. LOB은 서버에서 생성될 때 고유의 ID를 부여받는다. 그리고 클라이언트의 캐쉬에 어떤 LOB이 존재하는지의 여부 점검은 이 ID를 통해서 수행한다. 따라서 LOB이 캐쉬될 때 LOB의 ID와 TS가 같이 캐쉬 영역에 저장된다.



(그림 2) 일관성을 유지하는 LOB 검색 및 캐쉬 과정

캐쉬 일관성을 유지하는 LOB의 캐쉬 및 검색 과정은 (그림 2)와 같이 수행된다. 응용 프로그램이 질의(예를 들어, Computer Encyclopedia 3rd edition 데이타베이스로부터 'Von Neumann Machine'에 대한 설명 전문을 검색)를 통해 CLI 라이브러리에 LOB을 요구하면(그림 2의 ①) CLI 라이브러리는 우선 서버의 데이타베이스로부터 해당 LOB의 ID와 TS를 검색한다(그림 2의 ②~⑤). 그리고 캐쉬 영역을 접근하여 이 ID로 해당 LOB이 캐쉬되어 있는지 검사한다. 만약 캐쉬되어 있다면 캐쉬에 기록되어 있는 그것의 TS와 서버로부터 검색한 TS를 비교하여 캐쉬되어 있는 LOB이 최신의 것인지를 확인한다. 그 결과 다음과 같은 세 가지의 경우가 생길 수 있다.

① 캐쉬 적중 & LOB의 내용은 최신의 것 (그림 2의 ⑥-1)
② 캐쉬 적중 & LOB의 내용은 최신의 것이 아님 (그림 2의 ⑥-1)
③ 캐쉬 미스(cache miss) (그림 2의 ⑥-2)

첫째 경우, 캐쉬에 검색하고자 하는 LOB이 있고, 해당 LOB의 내용이 최신의 것이므로 캐쉬로부터 LOB의 검색이 가능하다(그림 2의 ⑦-1). 따라서 (망을 통해) 서버에 접근하여 LOB을 검색하지 않아도 되므로 LOB 검색 성능의 향상을 가져 올 수 있다. 둘째 경우, 캐쉬에 검색하고자 하는 LOB이 있지만 해당 LOB의 내용이 최신의 것이 아니므로 캐쉬에 있는 LOB은 삭제되어야 한다. 그리고 (망을 거쳐) 서버로부터 LOB을 검색 및 새로이 캐쉬하게 된다(그림 2의 ⑦-2). 셋째 경우에는 둘째 경우와 마찬가지로 (망을 통해) 원하는 LOB을 검색 및 캐쉬하게 된다. 둘째와 셋째 경우에는 기존의 CLI 함수를 이용하여 서버로부터 바로 LOB을 검색하는 경우에 비해 LOB 검색 시간이 더 오래 걸린다. 이는 검색하고자 하는 LOB이 캐쉬에 있는 지의 여부 및 캐쉬되어 있을 경우에는 그 LOB이 최신의 것인지의 여

부를 검사하기 위해 먼저 서버에 접근하여 ID와 TS를 검색해오는 오버헤드 때문이다(이 오버헤드에 대해서는 5절 성능평가에서 측정.).

이상의 세 가지 경우에 대해 CLI 라이브러리는 응용 프로그램이 원하는 LOB을 검색하여 반환하게되며 (그림 2의 ⑧), LOB 캐쉬의 일관성을 유지하게 된다.

3.2 LOB 캐쉬 교체 알고리즘

전통적인 DBMS의 메모리 버퍼는 데이타베이스의 내용을 디스크 페이지 단위로 캐쉬한다. 이들 페이지에 대한 교체 알고리즘에 대해서는 LRU를 비롯하여 많은 연구가 있어 왔다[7]. LOB을 캐쉬하고 있을 경우에는 LOB의 용량이 제각각 다르므로 다양한 용량의 데이타가 저장되어 있는 상황 하에서의 교체 알고리즘이 필요하다. 이들로는 LRU에 기반을 둔 LRU-SIZE, LRU-MIN, LRU-Threshold 등이 제안되어 있다[8]. 본 논문의 CLI 확장을 위해서는 이들 중 어느 것이든 사용이 가능하다. 본 논문의 구현에서는 LRU-MIN을 사용하였다.

LRU-MIN이란 교체시 크기가 큰 LOB을 우선 버리는 정책을 취한다. 새로 캐쉬에 저장될 LOB의 크기보다 큰 LOB들을 대상으로 LRU 방식의 교체를 수행한다. 만약 이러한 교체 후에도 캐쉬의 용량이 부족한 경우에는 새로 캐쉬에 저장될 LOB의 크기를 2로 나누어 그 크기보다 더 큰 LOB들을 대상으로 다시 LRU 방식의 교체를 수행하게 된다. 이러한 교체 후에도 캐쉬의 용량이 부족하면 저장될 LOB의 크기를 다시 2로 나누어 그 크기보다 더 큰 LOB들을 대상으로 다시 LRU 방식의 교체를 수행한다. 이러한 과정이 필요한 캐쉬 용량이 확보될 때까지 반복되어 수행된다.

3.3 LOB 캐쉬 정보의 효율적 관리

2절에서 언급한 바 있는 LOB 캐쉬 정보는 크게 LOB 캐쉬 내 모든 LOB에 대한 전체 정보와 각 LOB 별 개별 정보로 나눌 수 있다. LOB 캐쉬의 전체 정보로는 LOB 캐쉬 영역 정보, LOB 통계 정보, 캐쉬 검색을 위한 정보 등이 있다. LOB별 개별 정보로는 LOB의 식별자, 타임스탬프, 마지막 참조 시간, 그리고 크기 등이 있다. <표 1>은 이들을 요약한 것이다. LOB 캐쉬 정보는 캐쉬 영역에 LOB들과 함께 저장되며, LOB 캐쉬를 효율적으로 지원하기 위해서는 LOB 캐쉬 정보를 실행 시간에 메모리 상에서 효율적으로 유지, 관리할 필요가 있다. LOB 캐쉬 정보의 신속한 검색 및 갱신을 위하여 요구되는 자료구조는 크게 다음과 같이 분류할 수 있다.

- LOB 캐쉬 핸들(handle)
- LOB 캐쉬의 전체 정보를 저장하는 클래스
- 캐쉬된 LOB의 각각에 대한 개별 정보를 저장하는 클래스
- 캐쉬된 LOB을 크기별로 유지하는 리스트

LOB 캐쉬 핸들이란 LOB 캐쉬 정보를 저장하는 자료구조를 가리키는 포인터이다. SQL CLI의 환경(environment) 핸들, 연결(connection) 핸들, 그리고 문장(statement) 핸들[2-4]과 유사한 역할을 수행하는 것이다. 즉, 응용 프로그램은 LOB 캐쉬 핸들을 이용하여 캐쉬에 관련된 CLI 함수를 호출할 수 있다.

<표 1> LOB 캐쉬 정보

분 류		내 용
전체정보	LOB 캐쉬 영역 정보	최대 용량
		가용 용량
		캐쉬 영역이 설치된 저장공간의 디렉토리 경로
	LOB 통계 정보	마지막 참조된 시간별 통계
		크기별 통계
		갱신 시간별 통계
	캐쉬 검색을 위한 정보	캐쉬된 LOB들에 대한 인덱스
개별정보	LOB별 정보	식별자
		최근 갱신 타임스탬프
		마지막 참조 시간
		크 기

LOB 캐쉬에 관한 전체 정보와 개별 정보는 각각 클래스로 정의되어 저장된다. 전체 정보와 개별 정보를 별도의 클래스로 분리하여 관리함으로 인해 얻는 이점은, LOB을 캐쉬에 추가·삭제할 때, 관련 정보의 추가·삭제를 편리하고 일관적으로 수행할 수 있다는 것이다. 예를 들어, 전체 정보는 LOB의 추가 및 삭제가 일어나더라도 정보 저장을 위하여 할당된 메모리가 해제되지 않는 데 반하여, 개별 정보는 LOB이 캐쉬에/에서 추가 또는 삭제될 때 정보 저장을 위해 할당된 메모리가 할당 또는 해제되는데, 이들 두 정보를 따로 관리함으로써 이러한 차이점을 효율적으로 지원할 수 있다.

한편, 본 논문의 구현에서 사용하는 교체 알고리즘인 LRU-MIN은 주로 크기가 큰 LOB을 교체의 대상으로 삼기 때문에 캐쉬된 LOB을 크기별로 유지함으로써 신속한 교체가 가능하도록 하는 것이 바람직하다.

3.4 LOB 캐쉬를 위한 새로운 CLI 함수

본 절에서는 LOB 캐쉬를 지원하기 위한 새로운 CLI 함수를 제안한다. LOB 캐쉬 지원을 위하여 CLI에 추가되어야 할 함수는 기능별로 크게 다음과 같이 분류할 수 있다.

- LOB 캐쉬 핸들의 할당과 해제
- LOB 캐쉬 영역의 할당과 해제
- LOB 검색 및 캐쉬로부터 LOB 삭제
- LOB 캐쉬 정보의 변경과 검색

첫째, LOB 캐쉬를 위하여 캐쉬 핸들을 할당하고 해제하는 함수가 필요하다. 할당 함수는 LOB 캐쉬 정보의 자료구조를 위한 메모리를 할당하고 초기화한다. 그리고 핸들에 이 자료구조를 가리키는 값을 할당한다. 해제 함수는 이 자료구조를 메모리로부터 해제하고 핸들의 값을 NULL로 한다.

둘째, 효율적인 LOB 캐쉬를 위하여 LOB 캐쉬 영역을 여러 개 유지할 수 있다. 예를 들어, 자주 참조하는 LOB은 용량을 크게 할당한 캐쉬 영역에, 자주 참조하지 않는 LOB은 용량을 작게 할당한 다른 캐쉬 영역에 저장함으로써 캐쉬 교체의 빈도를 조절할 수 있다. 캐쉬 영역 할당 함수는 캐쉬 영역이 설치된 저장 공간 (예 : 하드 디스크)에 하나의 디렉토리를 설정하고 핸들의 값과 캐쉬 정보를 그에 맞게 갱신하는 작업을 수행한다. LOB 캐쉬 영역 해제 함수는 해당 디렉토리를 삭제한다.

셋째, LOB을 검색하는 기능과 캐쉬로부터 LOB을 삭제하는 기능이 필요하다. X/Open에서 명시한 기존의 CLI 사양의 경우, LOB의 검색을 위하여 지원되는 함수는 GetCol이다[2]. LOB 캐쉬를 지원하기 위해서 GetCol 함수는, 캐쉬 유효성 여부를 검사하고 서버로부터가 아닌 캐쉬로부터 LOB을 검색하도록 확장할 필요가 있다. 또한, 캐쉬 영역은 한정되어 있기 때문에 캐쉬 영역의 효율적인 사용을 위하여 캐쉬된 LOB의 삭제가 가능해야 하는데, 예를 들어 특정 LOB 또는 오래 전에 참조된 LOB의 삭제를 수행하는 함수가 필요하다.

넷째, LOB 캐쉬를 더욱 효율적으로 사용하기 위하여 LOB 캐쉬에 대한 전체 정보 및 각 LOB별 개별 정보를 검색 또는 갱신할 수 있는 함수가 필요하다. 예를 들어, 통계 정보를 검색하여 특정 LOB 또는 오랫동안 참조되지 않은 LOB 등은 삭제할 수 있다. <표 2>는 LOB 캐쉬 정보별로 검색 또는 갱신이 가능한지의 여부를 나타낸 것이다. 'O'로 표시된 것은 가능함을, 'X'로 표시된 것은 불가능함을 의미한다.

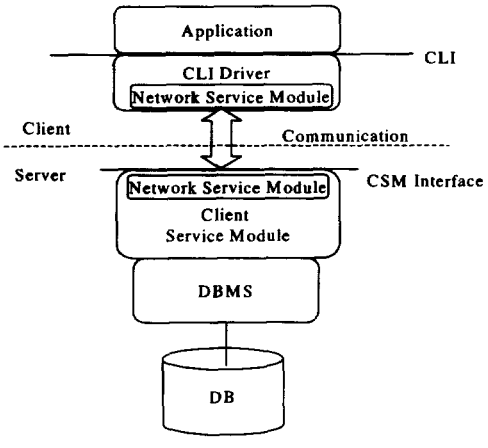
이상과 같이 새로이 제안된 CLI 함수들의 명칭을 정리하면 <표 2>와 같다. (이들 함수에 대한 완전한 명세와 설명(특히, 각 함수의 인자들 구성과 설명)은 [9]를 참조.)

〈표 2〉 LOB 캐쉬 정보의 검색 또는 갱신 가능 여부

LOB 캐쉬 정보	검색	갱신
캐쉬 영역의 이름	O	O
캐쉬 영역이 설치된 저장공간 상의 디렉토리 이름	O	X
캐쉬가 수용가능한 용량	O	O
현재 캐쉬에 저장된 LOB의 용량	O	X
사용가능한 캐쉬 용량	O	X
캐쉬된 LOB의 수	O	X
캐쉬된 LOB 중 최대 크기의 LOB	O	X
캐쉬된 최소 크기의 LOB	O	X

4. LOB 캐쉬를 지원하는 SQL CLI의 구현

본 논문에서 제안된 LOB 캐쉬 지원을 위한 CLI의 확장은 한국전자통신연구원에서 개발한 바다-II DBMS 상에서 구현하였다. 바다-II DBMS는 멀티미디어 데이타를 구성하는 LOB의 처리를 지원하도록 확장된 SQL CLI를 제공하고 있다[10]. 따라서 본 논문의 구현은 기존의 바다-II DBMS CLI 라이브러리를 LOB 캐쉬 기능을 추가하여 확장한 것이다. (본 구현에 있어 바다-IV와 같은 바다 DBMS의 최신 버전 대신 바다-II를 택한 것은, LOB을 지원하도록 확장된 CLI가 바다-II에서 제공되고 있기 때문이다.) 4.1절에서는 기존 바다-II DBMS CLI 라이브러리의 모듈 구조[10]를 기술하고, 4.2절에서는 그것의 클라이언트 및 서버측 확장에 대해 각각 기술한다. 4.3절에서는 본 논문의 구현 환경을 기술한다.



(그림 3) SQL CLI 라이브러리의 모듈 구조

4.1 SQL CLI 라이브러리 모듈 구조

망 환경의 클라이언트-서버 DBMS가 CLI를 응용 프로그래밍 인터페이스로 제공하기 위해서 CLI 라이브러리는 (그림 3)과 같이 세가지 모듈로 구성되어야 한다. 이들은 클라이언트 측 모듈, 서버 측 모듈, 그리고 이들 사이의 통신을 수행하는 모듈이다.

[10]에서는 클라이언트 측 모듈을 CLI 드라이버(CLI Driver), 서버측 모듈을 CSM(Client Service Module)이라 명명하였다. 그리고 이들 두 모듈간에 CLI 드라이버가 CSM의 기능을 호출하기 위한 인터페이스를 규정하였는데 이를 CSM 인터페이스라 명명하였다. 즉, CSM은 데이타 서버가 CLI 함수를 처리하기 위한 기능들을 CLI 드라이버에게 CSM 인터페이스를 통하여 제공한다. 한편, CLI 드라이버와 CSM은 각각 망을 통한 통신 기능을 제공해야 하는데 이를 담당하는 클라이언트 측과 서버 측의 모듈을 총칭하여 NSM(Network Service Module)이라 명명하였다.

CLI 드라이버는 응용 프로그램과 데이타 서버간의 데이타베이스 환경/연결 그리고 SQL 문장에 관한 상태 정보의 변경 및 관리와 상태 전이의 유효성 검사 등을 수행한다. CLI 드라이버는 CLI 응용 프로그램이 CLI 함수를 호출하

면 이를 받아서 데이타 서버의 기능을 요구하는 것일 경우에는 이에 대응되는 CSM 인터페이스 함수를 호출하고, 그렇지 않은 경우에는 CLI 드라이버 내에서 유지하고 있는 상태 정보를 이용하여 처리한다.

CSM은 CSM 인터페이스의 호출을 통해 CLI 드라이버로부터 요청받은 기능을 데이타 서버가 인식할 수 있는 형태로 변환하여 데이타 서버에게 요청한다. 그리고 데이타 서버의 수행 결과를 다시 CLI 드라이버에게 되돌려준다.

4.2 LOB 캐쉬 지원을 위한 SQL CLI 라이브러리의 확장

LOB 캐쉬를 지원하기 위한 CLI의 확장은 서버 측의 CSM 확장과 클라이언트 측의 CLI 드라이버 확장으로 구성된다. CSM의 확장은 곧 CSM 인터페이스의 확장을 의미하는데, 클라이언트의 응용 프로그램이 LOB 검색을 요청하였을 때 그것의 식별자(ID)와 타임스탬프(TS)를 서버 데이타베이스로부터 검색하여 전달하는 기능이 필요하다. 기존의 CSM에서는 LOB 검색 처리를 위해 우선 해당 LOB의 ID를 검색하여 전달하는 기능이 제공되고 있는데, 캐쉬되어 있는 LOB의 유효성 여부 판단을 위하여 TS 정보도 함께 반환하는 CSM 인터페이스 함수가 새로 제공되어야 한다.

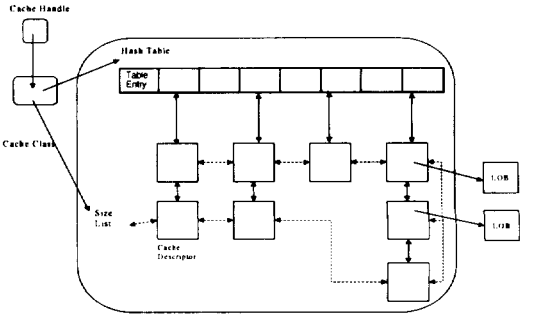
CLI 드라이버의 확장은 본 논문에서의 CLI 확장의 주된 내용을 구성하는 것으로서, 우선 기존 CLI드라이버에 캐쉬 관리자 모듈을 추가하고 이에 의해 관리되는 화일 공간을 할당하여 LOB 캐쉬 영역으로 사용할 수 있도록 하는 것이 필요하다. 또한, 캐쉬 영역 내에는 LOB 화일들과 LOB 캐쉬 정보가 저장되어야 하므로 이들을 위한 효율적인 자료구조가 필요하다. 그리고 이들 자료구조를 바탕으로 <표 3>에 열거된 LOB 캐쉬 지원을 위한 CLI 함수가 구현되어야 한다.

<표 3> LOB 캐쉬 지원을 위한 SQL CLI 함수*

CLI 함수	기 능
LOB 캐쉬 핸들의 할당과 해제	
AllocLOBCache()	캐쉬 핸들 할당
FreeLOBCache()	캐쉬 핸들 해제
LOB 캐쉬 영역의 할당과 해제	
AllocLOBCacheArea()	새로운 캐쉬 영역을 할당
FreeLOBCacheArea()	캐쉬 영역을 제거
LOB 검색과 캐쉬된 LOB의 삭제	
GetnCacheLOB()	LOB을 검색 및 캐쉬
PurgeLOB()	캐쉬에서 LOB을 삭제
캐쉬 정보의 변경과 검색	
GetLOBCacheInfo()	캐쉬 정보를 반환
SetLOBCacheInfo()	캐쉬 정보를 변경

* 각 함수의 명칭은 SQL CLI 표준 사양들에서처럼 모두 SQL로 시작

LOB 캐쉬 정보를 효율적으로 관리하기 위한 것으로 본 논문에서는 (그림 4)와 같이 해쉬 기반의 자료구조를 제안한다. 제안한 자료구조는 LOB 캐쉬 핸들(Cache Handle), 해쉬 테이블(Hash Table), 캐쉬 클래스(Cache Class), 캐쉬 기술자 클래스(Cache Descriptor Class), 크기 리스트 클래스(Size List Class)로 구성된다.



(그림 4) LOB 캐쉬를 지원하기 위한 자료구조

캐쉬 핸들은 캐쉬 정보를 저장하고 있는 자료구조에 대한 포인터로서 캐쉬 클래스를 가리킨다. 캐쉬 클래스는 LOB 캐쉬의 전체 정보 즉, 캐쉬의 총 용량, 가용 용량, 위치 등의 캐쉬 자체에 관한 정보를 저장한다. 그리고 해쉬 테이블에 대한 포인터, 크기 리스트에 대한 포인터를 저장한다. 캐쉬 클래스의 변수와 메소드는 (그림 5)와 같다.

```
class Cache
{
private:
    int     HT_size;
                //해쉬 테이블의 크기를 나타낸다.
    CacheTable  *_cacheTable;
                // 캐쉬 핸들을 가리킨다.
public:
    long    cache_max_size;
                // 캐쉬가 수용할 수 있는 최대 용량
    long    cache_current_size;
                // 현재의 캐쉬 용량
    char    *cache_directory_name;
                // 캐쉬의 디렉토리 이름
    CacheDescriptor  **HT_table;
                // CacheDescriptor를 가리키는 해쉬 테이블
    char    *current_directory;
                // 현재 프로세스가 실행되는 디렉토리
    char    *last_referenced_file;
                // 마지막 참조된 시간을 나타내는 화일
    SizeList    *_size_list;
    void    insert_descriptor(CacheDescriptor *p_descriptor);
                // CacheDescriptor를 하나 추가
    bool    delete_descriptor(CacheDescriptor *p_descriptor);
                // CacheDescriptor를 하나 삭제
    void    link_size_list(CacheDescriptor *p_descriptor);
                //SizeList에 추가
    void    remove_link_size(CacheDescriptor *p_descriptor);
                // SizeList에서 삭제
    void    alloc_timestamp(long p_lob_root_pid, char *p_last_referenced);
                // 새 타임스탬프 할당

    int     hash(long p_lob_root_pid);
                // 해쉬함수
    int     is_valid(long p_lob_root_pid, char *p_timestamp);
                // 유효성 검사
    void    construct_cache();
                // 캐쉬 정보 구축
    int     clear_cache();
                // 모든 캐쉬와 CacheDescriptor 제거
```

```
    int    LRU_MIN(long p_filesize);
                // LRU-MIN에 의한 캐쉬 대치
    int    remove_all();
                // 모든 캐쉬의 화일을 실제로 제거
    int    remove_a_lob(CacheDescriptor *p_desc);
                // 캐쉬에 저장된 LOB을 삭제
    int    remove_max_size(long p_size);
                // 주어진 크기보다 큰 화일을 제거
    int    remove_min_size(long p_size);
                // 주어진 크기보다 작은 화일을 제거
    long   remove_old_reference_time(char *p_referenced_time);
                // 오래전 참조된 LOB 제거

    Cache(char *p_aliasname="DEFAULT", int p_len_aliasname=6,int
    table_size=20, CacheTable *table=NULL); // 생성자~Cache(); //
    파괴자
};
```

(그림 5) 캐쉬 클래스의 선언부

캐쉬 기술자는 캐쉬되어 있는 LOB에 대한 개별 정보, 즉 LOB의 식별자, 타임스탬프, 마지막 참조 시간, 크기 등의 정보를 유지하고 아울러 실제 LOB이 저장된 곳을 가리키는 포인터를 저장한다. 캐쉬 기술자 클래스의 변수와 메소드는 (그림 6)과 같다.

```
class CacheDescriptor
{
public:
    long   lob_root_pid;
                // LOB 화일을 구분할 식별자
    char   *time_stamp;
                // LOB에 대한 데이타가 서버의 테이블에 입력된 시간
    char   *last_referenced_time;
                //  LOB이 마지막 참조된 시간
long   file_size;
                // LOB 화일의 크기
    char   *filename;
                // LOB 화일에 대한 포인터
    CacheDescriptor   *prev_pointer;
                // 앞 CacheDescriptor에 대한 포인터
    CacheDescriptor   *next_pointer;
                // 후 CacheDescriptor에 대한 포인터
    CacheDescriptor   *prev_size_pointer;
                //앞 SizeList에 대한 포인터
    CacheDescriptor   *next_size_pointer;
                //후 SizeList에 대한 포인터

    CacheDescriptor(long p_lob_root_pid=0, long p_file_size=0,
    char *p_time_stamp="", char *p_filename="",
         char *p_last_referenced_time="19000000");
                // CacheDescriptor 클래스의 생성자
    ~CacheDescriptor();
                // CacheDescriptor 클래스의 파괴자
};
```

(그림 6) 캐쉬 기술자 클래스의 선언부

크기 리스트는 캐쉬 기술자를 해당 LOB의 크기 순으로 유지하는 양방향 리스트이다. 이는 LRU-MIN 알고리즘에 의해 LOB의 크기를 기준으로 캐쉬 교체가 수행될 때 유용하게 사용될 수 있다. 크기 리스트 클래스의 변수와 메소드는 (그림 7)과 같다. (그림 5)~(그림 7)의 세가지 크래스에 대한 더욱 자세한 설명은 [9]를 참조.

```
class SizeList
{
public:
    int              num_descriptor;  // SizeList의 개수
    CacheDescriptor  *max_pointer;
                                  // 가장 크기가 큰 LOB의
                                    CacheDescriptor를 가리키는 포인터
    CacheDescriptor  *min_pointer;
                                  // 가장 크기가 작은 LOB의
                                    CacheDescriptor를 가리키는 포인터

    SizeList();   // SizeList의 생성자
    ~SizeList();  // SizeList의 파괴자
};
```

(그림 7) 크기 리스트 클래스의 선언부

해쉬 테이블은 캐쉬 기술자들의 리스트에 대한 앵커(anchor)들의 배열로서 LOB 식별자를 인자로 하는 해쉬 함수에 의해 버켓(즉, 해쉬 테이블의 한 엔트리)을 지정하면 그 버켓에 연결된 캐쉬 기술자들의 리스트를 탐색하여 해당 LOB이 캐쉬되어 있는지를 판단하게 된다.

<표 3>의 CLI 함수 중 가장 중요한 함수인 GetnCache LOB의 구현을 설명하면 다음과 같다[9]. 이 함수는 LOB을 서버 혹은 캐쉬로부터 검색한다. 만약 서버로부터 검색하였다면 검색 후 캐쉬한다. 이를 위해 기존의 X/Open CLI 사양에 정의된 함수인 GetCol의 기능에, LOB이 캐쉬되어 있는 지 확인하는 부분, LOB이 캐쉬에 있으면 캐쉬로부터 검색하는 부분, 캐쉬에 LOB이 없으면 서버로부터 LOB을 검색한 후 캐쉬에 저장하는 부분이 추가되었다. 기존의 GetCol 함수는 메모리가 지정되지 않은 컬럼의 LOB을 검색하는데 GetnCacheLOB도 같은 내용의 기능을 수행하도록 하되, 중간에 유효성을 검사하고 캐쉬로부터 검색하는 루틴만 추가함으로써, 기존의 CLI 응용 프로그램에서 GetCol 함수 호출을 GetnCacheLOB으로 교체하는 등의 일부 수정만을 요하도록 구현하였다. 따라서 GetnCacheLOB 함수의 인자 구성은 GetCol과의 용이한 호환을 고려하여 LOB 캐쉬 핸들, 구문 핸들, 검색할 LOB 컬럼 번호, 응용 프로그램의 데이타 타입, 검색된 LOB이 저장될 저장 장소 및 크기 등으로 하였다.

GetnCacheLOB 함수가 호출되면 CLI 드라이버는 먼저 망을 통해 서버에 접속하여 해당 LOB의 식별자(ID)와 타임스탬프(TS)를 동시에 검색한다. 이때 서버측의 확장된 CSM 인터페이스가 호출된다. 검색된 ID로 해당 LOB이 캐쉬에 있는지 확인하기 위해 (그림 4)의 해쉬 테이블에서 해당 버켓을 찾아 그에 연결된 캐쉬 기술자들을 탐색한다. 해당 LOB이 캐쉬되어 있다면 해당 캐쉬 기술자에 기록된 타임스탬프와 서버로부터 검색된 TS를 비교하여 캐쉬된 LOB의 유효성을 검사한다. 만약 유효하다면 캐쉬 기술자에 기록된 LOB 화일 포인터를 통해 LOB을 캐쉬로부터 직접 검색한다. LOB이 유효하지 않거나 캐쉬 미스가 발생한 경우에는 기존의 GetCol처럼 망을 통해 서버로부터 LOB을

검색하게 된다. 이 경우에는, 캐쉬 영역에 LOB 화일을 생성하여 검색된 LOB을 복사하고, 이를 위한 캐쉬 기술자를 할당하여 해당 버켓의 리스트에 삽입한 후 캐쉬 영역의 LOB 화일에 대한 포인터를 기록한다.

4.3 구현 환경

본 논문에서 확장된 CLI 라이브러리는 클라이언트 측에서는 Windows NT PC에서 MS Visual C++ 6.0으로 실행시간 라이브러리로 구현하였고, 서버 측에서는 동적 SQL을 사용하는 ESQL/C 프로그램[11]으로 구현하였다. 바다-II DBMS 서버는 Solaris 2.4 환경에서 구동되고, PC 클라이언트와 Ethernet으로 연결되어 있다.

5. 성능평가

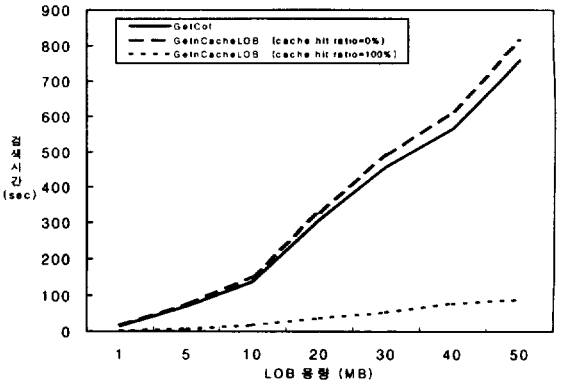
본 절에서는 본 논문에서 구현한 LOB 캐쉬를 위해 확장된 CLI에서 지원하는 LOB 검색 (및 캐쉬) 함수인 GetnCacheLOB의 성능을 기존의 X/Open CLI 표준 함수인 GetCol의 성능과 비교하여 측정한 결과를 기술한다. 성능 평가 사항은 다음과 같다.

- GetCol과 GetnCacheLOB 함수의 LOB의 크기에 따른 최악의 성능 및 최고의 성능 비교
- GetCol과 GetnCacheLOB 함수의 캐쉬 적중률에 따른 성능 비교

(그림 8)은 X/Open의 CLI 사양에서 LOB을 검색하는 함수인 GetCol과 이 함수에 캐쉬 기능을 부가하여 확장한 GetnCacheLOB의 LOB의 크기에 따른 최고의 성능과 최악의 성능을 비교한 것이다. 최고의 성능이란 GetnCacheLOB의 경우에 캐쉬 적중률이 100%일 때를 의미하고, 최악의 성능이란 캐쉬 적중률이 0%일 때를 의미한다. 여기서 캐쉬 적중률은 검색하고자 하는 LOB이 캐쉬 영역에 존재할 뿐만 아니라 유효한 상태여서 서버로부터의 검색이 필요하지 않은 경우의 비율을 의미한다. 즉, 캐쉬 영역에 존재하더라도 유효하지 않아 서버로부터 다시 검색해오는 경우는 캐쉬 미스로 간주하여 캐쉬 적중률에 포함시키지 않았다.

LOB의 크기로는 1M, 5M, 10M, 20M, 30M, 40M, 50M 바이트 용량의 LOB들을 대상으로 검색 시간을 측정하였다. 검색 시간의 단위는 초(sec)이다. 각 LOB의 검색 시간은 20회 검색 시간의 평균값이다. 다음과 같은 세가지 경우의 검색 시간을 측정하였다.

- GetCol을 사용하여 LOB 검색
- 캐쉬 적중률 0%일 때, GetnCacheLOB을 사용하여 LOB 검색
- 캐쉬 적중률 100%일 때, GetnCacheLOB을 사용하여 LOB 검색



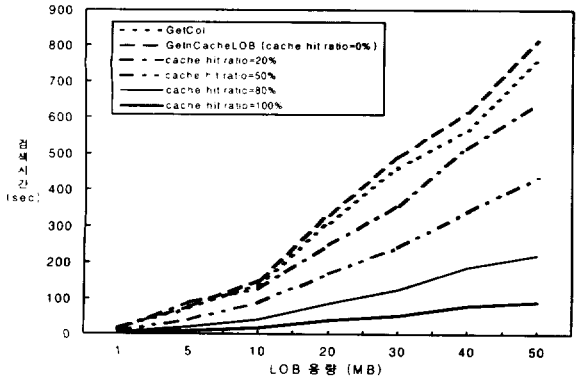
(그림 8) GetnCacheLOB의 성능 평가

캐쉬 적중률이 0%일 경우에는 LOB 캐쉬 지원 시 최악의 성능을 내는 경우로 기존의 GetCol을 이용한 LOB 검색보다 좋지 않은 성능을 나타낸다. 그 이유는 유효성 검사를 위하여 LOB의 식별자와 타임스탬프를 검색하는 시간과 검색된 LOB을 캐쉬에 저장하기 위하여 추가 비용이 들기 때문이다. 반면에 캐쉬 적중률이 100%인 경우에는 최고의 성능을 내는 경우로 GetCol에 비해 상당한 성능 향상이 있음을 알 수 있다. 이것은 캐쉬 적중률이 100%일 경우 모든 LOB은 캐쉬로부터 검색 가능하기 때문이다. 예를 들어, 50M 바이트 LOB의 경우 캐쉬 미스 시 GetnCacheLOB은 GetCol에 비해 약 7%의 성능 저하를 가져왔다. 반면에 캐쉬 적중 시 GetnCacheLOB은 GetCol에 비해 약 88%의 성능 향상을 가져왔다. 이상으로부터 LOB을 캐쉬함으로써 얻을 수 있는 성능 향상이 그 오버헤드를 감안하더라도 아주 크다는 것을 알 수 있다.

(그림 9)는 캐쉬 적중률에 따른 GetnCacheLOB과 GetCol의 성능 비교를 나타낸 것으로 LOB의 크기 단위로 검색 시간을 측정한 것이다. GetnCacheLOB을 사용하는 경우, 캐쉬 적중률이 0%, 20%, 50%, 80%, 100%일 경우로 나누어 성능을 측정하였다. (위에서 언급한 것처럼, 예를 들어 캐쉬 적중률이 80%란 것은, 검색하고자 하는 LOB이 유효한 상태로 캐쉬 영역에 존재하여 서버로부터의 검색이 필요하지 않은 경우가 80%라는 것을 의미한다.)

0%와 100%의 경우는 앞의 실험(그림 8)에서 얻은 결과와 같다. 캐쉬 적중률이 높아질수록 검색 시간은 줄어들었다. 캐쉬 적중률이 낮을수록 검색 시간은 증가하지만 기존의 GetCol의 검색 시간 만큼 커지지는 않았다. 50M 바이트 LOB 검색의 경우 캐쉬 적중률이 20%인 경우에 GetnCacheLOB은 기존의 GetCol에 비해 약 21%의 성능의 향상을 가져왔다. 이 성능 향상은 최악의 경우(즉, 캐쉬 적중률 = 0%)에 오버헤드로 인한 7%의 성능 저하보다 크다. 이와 같은 결과는 검색하고자 원하는 LOB의 20%만이 서버로부터가 아닌 캐쉬로부터 검색 가능하다 할지라도 캐쉬를 사용하여 얻을 수 있는 성능 향상이 캐쉬 지원을 위한 오버

헤드로 인한 성능 저하보다 훨씬 크다는 것을 나타낸다.



(그림 9) 갱신 비율에 따른 GetnCacheLOB의 성능 평가

6. 결 론

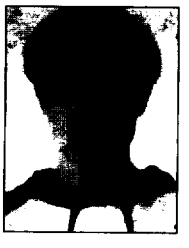
망을 통한 클라이언트/서버 컴퓨팅 환경에서 멀티미디어 데이타를 구성하는 LOB의 효율적 검색 기법이 요청되고 있다. 사용자가 자주 검색하는 LOB은 응용에 따라 전체 데이타베이스의 일부분에 국한되어 동일한 LOB의 검색이 반복되며, 오디오, 이미지와 같이 멀티미디어 데이타를 구성하는 LOB은 그 내용이 잘 변경되지 않는다. 본 논문에서는 이러한 특성을 고려하여 윈도우 환경의 PC에서 널리 쓰이고 있는 SQL CLI 상에서 LOB의 캐쉬를 제안하였다. 이를 위하여 캐쉬 일관성 유지 기법, 캐쉬 교체 기법, LOB 캐쉬 정보 관리를 위한 자료구조, 기존의 표준 CLI에 추가되어야 할 새로운 CLI 함수를 제안하였다. 그리고 제안한 CLI 확장 내용을 CLI를 지원하는 바다-II DBMS에 적용하여 설계, 구현하고 기존의 CLI 함수를 이용한 LOB 검색의 성능과 LOB 캐쉬 확장 함수를 이용한 LOB 검색의 성능을 비교 평가하였다.

성능 평가 결과, 캐쉬 적중률이 그다지 높지 않은 경우에도 LOB 캐쉬를 지원하기 위한 오버헤드로 인한 성능 저하에 비해 서버로부터가 아닌 캐쉬로부터의 LOB 검색이 가져다 주는 성능 향상이 훨씬 크다는 것을 확인할 수 있었다.

향후 연구과제로는, 첫째, 본 논문의 구현에서는 LOB 교체 기법으로 LRU-MIN을 사용하였는데 이와 같은 기존의 교체 기법에 사용자의 LOB에 대한 선호도를 반영하는 연구가 필요하다. 즉, 사용자가 선호하는 특정 LOB의 교체를 최대한 지연시킬 수 있도록 교체 기법을 확장하는 것이 바람직하다. 이를 위한 구현 방안은 LOB 캐쉬 정보의 하나로서 캐쉬된 각 LOB별 선호도 값을 유지하고 이를 교체 알고리즘이 참조하도록 하는 것이다. 사용자의 선호도는 변경될 수 있으므로 이 선호도 값을 조작할 수 있도록 해당 CLI 함수도 확장되어야 한다. 둘째, 본 논문에서 제시한 CLI를 통한 LOB의 캐쉬 기능은 관계 데이타 모델을 기반으로 한 질의 캐싱인데 이 기능과, ObjectStore와 같은 기존 상용 객체지향 DBMS들에서 제공하고 있는 페이지 캐싱 기반의 클라이언트 데이타 캐싱을 통한 LOB 캐쉬 기능과의 비교 연구도 필요하다.

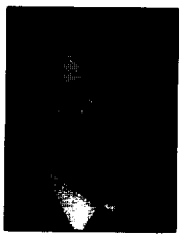
참 고 문 헌

[1] M. Venkatrao and M. Pizzo, "SQL/CLI - A New Binding Style For SQL," SIGMOD Record, Vol.24, No.4, pp.72-77, Dec. 1995.
[2] X/Open Company Ltd., "Data Management : SQL Call Level Interface(CLI)," X/Open Snapshot, Sep. 1992.
[3] J. Melton (Ed.), "ISO-ANSI (Working Draft) SQL Call Level Interface (CLI)," ISO DBL MUN-005/ ANSI X3H2-93-360, Aug. 1993.
[4] Microsoft Corp., "Microsoft ODBC 3.0 Software Development Kit and Programmer's Reference," Microsoft Press, 1997.
[5] G. Hamilton et al., "JDBC Database Access With Java : A Tutorial and Annotated Reference," Addison Wesley, 1997.
[6] M. Franklin et al., "Transactional Client-Server Cache Consistency : Alternative and Performance," ACM TODS, Vol.22, No.3, pp.315-363, Sep. 1997.
[7] W. Effelsberg and T. Haerder, "Principles of Database Buffer Management," ACM TODS, Vol.9, No. 4, pp.560-595, Dec. 1984.
[8] M. Abrams, et al., "Caching Proxies : Limitations and Potentials," In Proc. of the Fourth Int'l Conf. on the WWW, Boston, USA, December 1995.
[9] 이종민, "개인 컴퓨팅 환경에서 효율적인 LOB 검색을 위한 SQL CLI의 확장", 석사학위논문, 중앙대학교 컴퓨터공학과, 2000. 2.
[10] 강은지, 이재성, 김지현, 장한울, 강현철, 전성택, "클라이언트-서버 DBMS를 위한 멀티미디어 확장 SQL CLI의 개발", 한국정보과학회 논문지(C), 3권 4호, pp.343-352, 1997.
[11] 이재성, "망 환경에서 멀티미디어 데이타 처리를 위한 동적 SQL의 확장", 석사학위논문, 중앙대학교 컴퓨터공학과, 1996. 8.



이 종 민

e-mail : lons@bada.voin.co.kr
1998년 중앙대학교 컴퓨터공학과 졸업 (공학사)
2000년 중앙대학교 대학원 컴퓨터공학과 (공학석사)
현재 보인기술주식회사 연구원 재직중
관심분야 : XML, Web Mail, ASP & JSP 등



강 현 철

e-mail : hckang@cau.ac.kr
1983년 서울대학교 컴퓨터공학과 졸업 (공학사)
1985년 U. of Maryland at College Park, Computer Science(M.S.)
1987년 U. of Maryland at College Park, Computer Science(Ph.D.)
1988년~현재 중앙대학교 컴퓨터공학과 교수
관심분야 : 이동 데이타베이스, 웹 데이타베이스, DBMS 저장 시스템 등