시스템 성능 향상을 위한 Primary key 기반 T-Cache 설계 및 구현

강형만, 이언배
*한국방송통신대학교 평생대학원 정보과학과
e-mail:khm6280@gmail.com

Implementation of T-Cache engine based on Primary key for enhancing System Performance

Hyung-Man Kang, Un-Bae Lee
*Dept of Information Science, Korea National Open University Graduate School

요 약

인터넷 및 스마트폰 등 모바일 시장의 급성장으로 다양한 채널이 발달하여 금융거래가 급속하게 증가함으로써 시스템 자원이 부족하고 또한, 급변하는 금융시장에서 경쟁력을 잃지 않기 위하여 국내의 금융권 시스템들은 차세대를 진행하면서 경쟁적으로 프레임웍을 도입하여 프로젝트를 진행하였거나 또는 진행하고 있다. 프레임웍은 요청한 거래를 검증하고, 처리하여 결과를 반환할 수 있도록 여러 가지 편의성을 제공하지만, 동일 테이블 데이터를 매 거래마다 데이터베이스를 조회함으로써 데이터베이스 서버의 부하가 증가하고 거래 처리가 지연되는 문제점이 있다.

본 논문에서는 프레임웍 기반의 매 거래마다 동일 데이터를 데이터베이스로부터 질의함으로써 발생하는 거래처리 지연을 극복하고 보다 빠른 응답 처리를 위하여 1) 대부분 조회를 처리하는 테이블에 대해서 테이블 단위로 Primary key을 이용하여 공유메모리에 저장하고, 많은 응용프로그램 간에 공유하는 방식으로 거래를 처리함으로써 디스크 I/O나 네트워크 I/O, DBMS 자체 프로세싱을 크게 감소하여 전체적으로 시스템의 성능을 향상시키며 2) 공유메모리에 저장하고 있는 데이터와 데이터베이스 테이블에 저장된 데이터간의 동기화를 지원하는 Primary key 기반 T-Cache(Table Cache) 알고리즘을 제안한다.

1. 서론

그동안 금융 거래 처리는 영업점 단말 위주의 거래 처리에서 인터넷 및 스마트폰 등 모바일 시장의 급성장으로 금융거래의 다양한 채널이 발달하여 금융 거래가 급속하게 증가하면서 데이터베이스 처리량도 기하급수적으로 증가하고 있다[1]. 이처럼 급변하는 금융시장에서 선도 금융그룹으로 도약할 수 있는 기반을 만들고자 국내의 금융시스템들은 차세대를 진행하면서 경쟁적으로 프레임웍을 도입하여 프로젝트를 진행하였거나 또는 진행하고 있다. 프레임웍은 업무 개발자가 순수하게 응용프로그램 개발에만 전념할 수 있도록 거래 처리에 필요한 전반적인 시스템 환경을 제공하는 솔루션이다. 프레임웍 환경에서의 거래 처리 흐름은 그림1과 같이 대표서비스 개념을 도입하여, 요청된 거래는 1차적으로 대표서비스에서 검증 및 분석 후에 업무 처리 응용프로그램을 호출하여 거래를 처리하고 결과를 반환하는 구조이다[2]. 그러나 프레임웍은 거래 처리를 위한 전반적인 환경을 제공하는 반면 요청 거래를 검증하고, 분기하기 위하여 거래마다 데이터베이스의 특정 테이블을 조회함으로써 병목현상을 초래하여 거래처리가 지연되고 시스템 오버헤드를 가중시키는 문제점이 있다.

특히 금융시스템에서 거래 처리 속도는 고객 만족과 직결되기 때문에 얼마나 빠르게 처리할 수 있는가에 따라서 금융시스템의 경쟁력이 되고 있다. 이처럼 빠른 거래 처리 시스템을 만들기 위한 여러 가지 요소가 있겠지만, 데이터베이스 관점에서 보면 데이터를 어떻게 모델링하고, 시스템 간 데이터 정합성을 유지하면서 데이터베이스 질의를 최적화하며, 매 거래마다 반드시 필요한 데이터를 어디에 저장하는가에 달려있다.

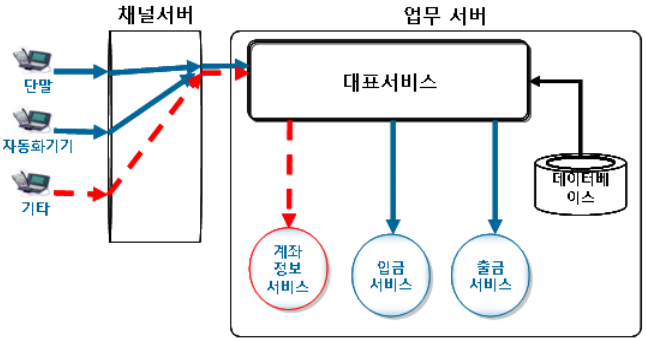

그림1. 대표 서비스 구조

그러나 다양한 종류의 거래가 발달함에 따라 응용프로그램에서 처리할 데이터베이스 질의 건수가 증가하고 있으며, 요청한 거래 처리를 위하여 단순하게 하나의 응용프로그램만으로 처리하지 못하고 다수의 응용프로그램이 연동하면서 복잡도가 증가하고 있다. 시스템 구성 측면도 업무 처리 서버와 데이터베이스 서버가 분리되어 있어 데이터베이스 질의 건수가 증가하면 할수록 네트워크 I/O도 급속하게 증가하고 있다.

거래 처리를 위하여 사용하는 데이터의 활용도 측면에서 보면, 한번 읽은 데이터를 재사용하지 못하고 연동하는 응용프로그램 간에 동일 데이터를 개별적으로 데이터베이스에서 질의함으로써 데이터베이스 서버에 부담을 가중시킨다. 데이터를 재사용할 수 있도록 캐쉬하는 방법에는 여러 가지가 있는데 첫째, 데이터베이스 자체적으로 한번 읽은 데이터를 데이터베이스 자체 메모리에 저장해 두었다가 재사용하여 처리 속도를 개선하는 방법[3]과 둘째, 성능에 가장 영향을 미치는 테이블에 대해서 메모리 DB를 사용하여 메모리에 저장함으로써 처리 성능을 향상시키는 방법 셋째, 유사질의 매칭기반 알고리즘을 사용하여 SQL 질의 결과를 저장해 두었다가 동일 질의이면 저장해둔 데이터를 재사용하는 방법[5] 등이 있는데 위의 방법들은 모두 금융시스템의 OLTP 환경에는 적합하지 않다.

본 논문에서는 OLTP 환경에서 매 거래마다 조회용으로 많이 사용하는 테이블에 대해서 업무 서버 시스템의 공유메모리에 저장해 두었다가, 이후에 공유메모리에 저장된 데이터를 재사용하여 전체적으로 시스템의 성능을 향상시킬 수 있는 T-Cache(Table Cache) 알고리즘(이후 캐쉬 알고리즘이라 함)을 제안한다.

본 논문의 구성은 다음과 같다. 2장 관련 연구에서는 사용제품인 데이터베이스 관리 시스템의 데이터 캐쉬 기능과 유사 질의 매칭 기반 데이터베이스 캐쉬 엔진을 설명하고, 3장에서 T-Cache의 개념과 알고리즘과 성능 실험 결과를 분석하고, 4장에서는 논문을 요약하고 결론을 맺는다.

2. 관련연구

데이터를 캐쉬하는 목적은 시스템에 따라서 매우 다양하다. 첫 번째로 데이터베이스 엔진 자체에서 데이터를 캐쉬하는 목적은 물리적인 I/O 없이 많은 프로세스 간에 데이터를 공유함으로써 목표 시스템의 처리 성능을 향상시키는 방법과, 두 번째로 유사 질의 매칭처럼 응용프로그램 단에서 특정 알고리즘으로 질의 결과 데이터를 캐쉬하여 응용프로그램 간에 데이터를 공유하도록 함으로써 처리속도를 향상시킬 수 있는 방법이 있다.

먼저 오라클 데이터베이스에서 데이터를 캐쉬하는 방법은 그림2와 같은 구조의 공유메모리(System Global Area)를 이용한다. 공유메모리는 공유 풀(Shared Pool), 데이터베이스 버퍼 캐쉬(Database buffer cache), 리두 로그 버퍼(Redo log buffer)로 구성된다[3]. 공유 풀은 SQL 구분 분석 결과와 사용자에 대한 정보가 있으며, 데이터베이스 버퍼 캐쉬는 SQL 구문 분석 결과인 데이터베이스의 데이터를 저장하고 있다. 공유 풀의 저장 대상인 SQL은 데이터의 삽입, 삭제, 수정, 조회하는 모든 경우를 분석 대상으로 한다. 사용자가 SQL을 실행하면 먼저 공유 풀에서 이전에 질의한 질의와 동일한 질의가 존재하는지 찾아본 후, 동일 질의인 경우 이전에 파싱하고 논리적·물리적 실행 계획을 수립한 것을 곧바로 실행할 수 있다. 또한 데이터베이스 버퍼 캐쉬에 결과 데이터가 존재하면 해당 결과를 곧바로 응용프로그램에 전달한다[4]. 만약 동일한 질의가 아닌 경우에는 처음부터 SQL 질의 문을 파싱하여 실행 계획을 수립하고 실행한다. 이처럼 데이터베이스 버퍼 캐쉬에 저장된 데이터라 할지라도 매번 파싱하고 실행 계획을 수립하기 때문에 시스템에 대한 오버헤드가 발생한다. 또한 SQL 질의가 오라클 데이터베이스 서버에 전달되어야 하고, 데이터베이스 버퍼 캐쉬에 저장된 데이터를 네트워크를 통하여 전달해야 하기 때문에 데이터베이스 자체 처리 및 통신 네트워크에 부담을 주게 된다.

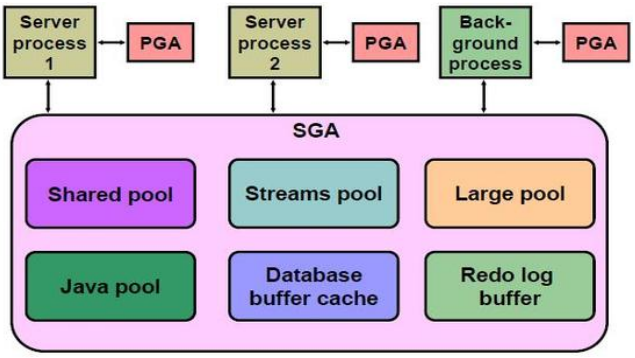

그림2. 오라클 데이터베이스 메모리 구조

유사 질의 매칭 기반 데이터베이스 캐쉬는 오라클 데이터베이스에서 나타난 문제점을 해결하기 위하여 완전 일치 매칭과 유사 질의 매칭 방법을 사용한다. 완전 일치 매칭은 두 질의 문장의 문자가 정확히 일치할 경우를 말하고, 유사 질의 매칭은 질의 결과로 생성되는 결과가 같은 경우에 동일한 질의로 판단한다[5]. 이와 같이 유사 질의 매칭은 완전 일치 매칭을 포함하여 더 넓은 의미의 매칭이다. 그러나 유사 질의 매칭 알고리즘은 캐쉬에 저장된 데이터의 일부가 변경될 경우 데이터베이스 데이터와 동기화가 이루어지지 않는 문제점이 있으며, 또한 “SELECT A, B, C FROM table”와 “SELECT B, C, A FROM table” 는 동일한 질의로 인식되어 응용프로그램에 데이터 전달 시 응용프로그램과 칼럼의 불일치가 발생하는 문제점이 있다. 마지막으로 OLTP 환경에서 업무 처리는 대부분 동일한 질의라 하더라도 조건 문장이 다르기 때문에 유사 질의 매칭 기반 데이터베이스 캐쉬에 저장된 데이터를 대부분 재사용할 수 없다.

지금까지 데이터베이스 관련 연구는 데이터베이스 엔진

자체에 대한 튜닝 방법과 유사 질의 매칭 기반 데이터베이스 캐쉬처럼 동일 질의가 많은 대량의 데이터를 조회하는 부분에 연구가 많았고, OLTP 환경의 업무 처리에서 Primary key를 기반으로 효과적으로 데이터를 저장하고 이를 재사용 할 수 있는 부분에 대해서는 연구가 미진하다. 본 논문에서는 OLTP 환경의 업무 처리에서 데이터를 효과적으로 저장하여 재사용함으로써 전체적으로 시스템의 성능을 향상시킬 수 있는 방법을 제시한다.

3. T-Cache 설계 및 구현

3.1. T-Cache 알고리즘

T-Cache 알고리즘은 OLTP 환경의 거래 처리에서 특정 테이블을 매번 데이터베이스에서 조회함으로써 거래 처리가 지연되는 문제점을 해결하기 위해 그림3과 같이 많은 응용프로그램들 간에 테이블 단위로 데이터를 공유하는 알고리즘이다.

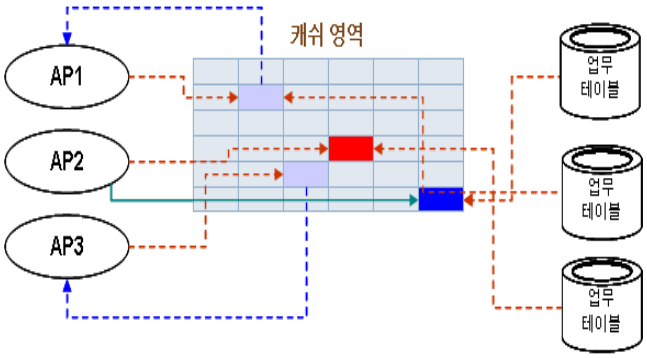

그림3. T-Cache 알고리즘 아키텍처

캐쉬 알고리즘은 데이터를 필요로 하는 응용프로그램에서 처음 한번만 데이터베이스로부터 조회하여 Primary key 기반으로 해싱 알고리즘을 사용하여 공유메모리에 저장하고, 이후부터 해당 데이터를 사용하는 응용프로그램은 저장되어 있는 데이터를 공유하는 방식이다. 즉, 공유메모리에 데이터를 저장하고 조회하는 순서는 캐쉬 대상 테이블의 데이터를 응용프로그램에서 먼저 Primary key를 이용하여 공유메모리에서 참조하고, 공유메모리에 데이터가 없으면, 데이터베이스로부터 참조하여 공유메모리에 저장한다. 이후 다른 응용프로그램에서 동일 Primary key에 해당하는 데이터를 필요로 하는 경우, 공유메모리로부터 데이터를 조회함으로써 거래 처리를 빠르게 할 수 있는 알고리즘이다. 그림4는 응용프로그램에서 캐쉬 알고리즘을 이용하여 데이터를 조회하는 순서도이다.

3.2. T-Cache 메모리 구조

캐쉬 알고리즘에서 저장 공간으로 사용하는 공유메모리의 구조는 그림5와 같다. 가장 상위에 캐쉬 대상 테이블의 정보를 저장하는 마스터 영역이 있고, 그 아래에 Primary key에 해당하는 데이터를 빠르게 저장하고 찾을 수 있도록 해싱 알고리즘으로 선택하는 해싱 엔트리 영역이 있다. 특히, 해싱 엔트리 영역에는 엔트리마다 Lock을 설정할 수 있어 데이터 영역에 저장되어 있는 데이터를 변경할 때 데이터 영역 전체에 Lock을 설정하지 않고 해당 엔트리에만 설정하게 하여 변경이 발생해도 거래 처리 지연을 최소화 할 수 있는 구조이다.

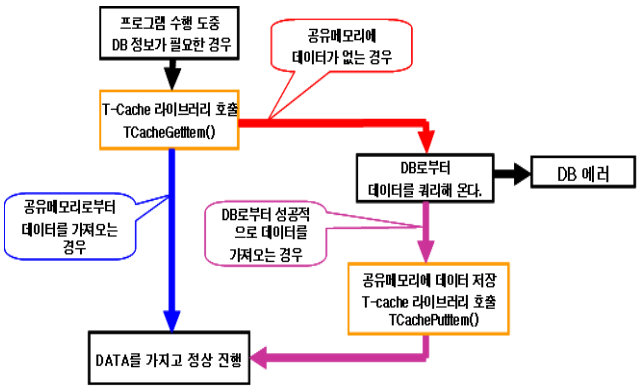

그림4. T-Cache 데이터 조회 순서도

마지막으로 데이터를 저장하는 데이터 엔트리 영역으로 Free List를 두어 데이터를 저장할 때 Free List에서 데이터 엔트리를 가져다가 데이터를 저장하고 해싱 엔트리에 연결하고, 삭제하면 해싱 엔트리로부터 Free List로 옮겨 재사용할 수 있다. 서로 다른 Primary key가 충돌이 발생하면 Linked List 방식으로 데이터 엔트리를 관리한다.

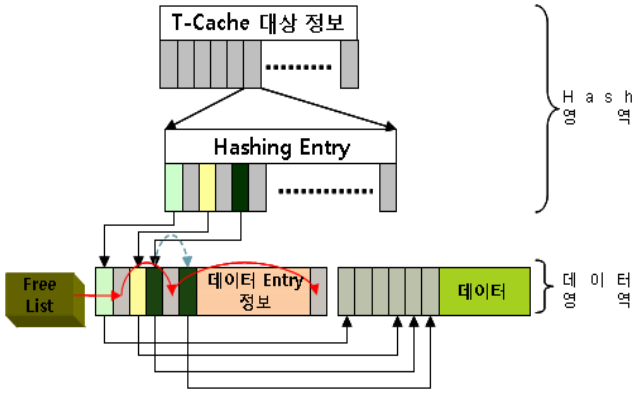

그림5. T-Cache 메모리 구조

위와 같은 구조의 공유메모리는 용량이 한정되어 있기 때문에 캐쉬 대상 테이블의 데이터를 모두 공유메모리에 저장할 수 없다. 특히 금융 거래에서 가장 사용 빈도가 높은 거래는 전체 거래의 5% 내외이기 때문에 LRU 알고리즘을 사용하면 사용 빈도가 높은 거래는 항상 공유메모리에서 찾을 수 있어 적은 용량으로도 상당한 효과를 볼 수 있다.

3.3. 데이터 동기화 방법

캐쉬 알고리즘은 캐쉬에 저장된 데이터와 데이터베이스에 저장된 데이터 간에 Primary key 단위로 동기화를 지원한다. 응용프로그램에서 Primary key에 해당하는 데이터를 변경할 때, 먼저 데이터베이스의 데이터를 변경하고, 다음으로 공유메모리에 저장된 데이터를 삭제한다. 트랜잭션은 서비스가 종료되는 시점에 커밋한다. 이후에 동일 데이터를 필요로 하는 응용프로그램에서 삭제된 데이터를 공유메모리에서 찾을 수 없기 때문에 데이터베이스에서 질의하게 하는 방식으로 동기화를 지원한다. 그러나 공유메모리에서 데이터를 삭제하고 데이터베이스에 커밋할 때까지 시간차가 발생하여, 이 사이에 다른 응용프로그램이 데이터베이스로부터 이전 데이터를 조회하여 캐쉬에 저장할 수 있기 때문에 캐쉬에는 이전 데이터가 저장되고, 데이터베이스에는 최신 데이터가 저장되어 데이터 불일치 문제가 발생할 수 있다. 그래서 캐쉬 알고리즘에서는 캐쉬에 저장된 데이터를 삭제한 이후부터 일정시간 동안 해당 데이터를 데이터베이스에서 직접 조회하고, 지정된 시간이 경과한 이후부터 캐쉬에 저장된 데이터를 조회하게 하는 방식으로 데이터베이스와 동기화를 처리한다.

3.4. 실험 및 성능평가

본 논문의 실험 환경은 업무 서버와 데이터베이스 서버를 별도의 노드로 분리하고, 업무 서버에서 매 거래마다 데이터베이스를 조회하는 프로세스와 캐쉬 알고리즘을 적용한 프로세스를 동일한 수만큼 실행한 상태에서 사용자 수를 증가하면서 거래를 요청하였을 때 처리 시간을 비교하였다.

캐쉬 알고리즘을 적용한 프로세스와 적용하지 않은 프로세스의 처리시간을 비교하면 그림 6과 같다. 캐쉬 알고리즘을 적용한 프로세스는 클라이언트가 증가하더라도 응답시간이 거의 증가하지 않고 안정적으로 처리하는 반면, 캐쉬 알고리즘을 적용하지 않은 프로세스는 클라이언트가 증가함에 따라서 응답 시간이 기하급수적으로 증가하는 것을 볼 수 있다. 또한 적용전과 적용후의 차이가 많이 발생하는 이유는 캐쉬 알고리즘에서 적중률이 높거나 또는 데이터베이스 자체 캐쉬의 적중률이 떨어지는 경우에 더 많은 차이가 발생하는 것으로 나타났다.

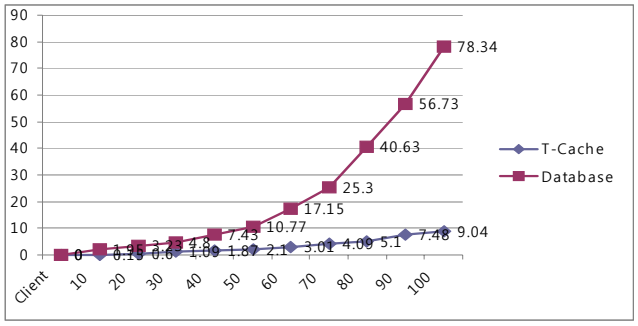

그림6. T-Cache 알고리즘 성능 비교

이처럼 캐쉬 알고리즘을 적용하지 않은 프로세스는 매 거래마다 데이터베이스를 조회함으로써 응답시간이 증가할뿐더러 시스템의 사용량 및 네트워크 I/O가 증가하여 전체적으로 시스템의 성능이 저하되는 것을 볼 수 있다.

4. 결론

본 논문에서는 금융시스템 OLTP 환경의 거래 처리에서 시스템의 성능을 향상시키기 위해 Primary key 기반 T-Cache 알고리즘을 제안하였다. 데이터베이스 자체에서 캐쉬하고 있는 데이터를 재사용하기 위해서는 질의 문을 데이터베이스에 전달하고 결과를 받아야 하기 때문에 데이터베이스 자체 처리 및 네트워크 부하가 발생하지만 본 논문에서 제안한 T-Cache 알고리즘은 응용프로그램이 실행되는 서버에서 직접 처리하기 때문에 데이터베이스 및 네트워크 부하가 발생하지 않는다. 또한, 멀티 데이터를 캐쉬하는 유사질의 매칭기반 캐쉬 알고리즘은 대부분 하나의 로우를 처리하는 OLTP 거래 처리에는 적합하지 않다. 이와 같이 본 논문에서 제안하는 T-Cache 알고리즘은 모든 거래에서 반복적으로 데이터베이스에 질의하던 것을 공유메모리에서 빠르게 처리함으로 디스크 I/O나 네트워크 I/O, 데이터베이스 자체 프로세싱을 감소하여 전체 시스템의 성능을 크게 향상시킬 수 있는 알고리즘이다.

본 논문의 실험 결과처럼 모든 거래에서 데이터베이스를 조회하는 것보다 T-Cache 알고리즘을 적용한 것이 시스템 성능을 크게 향상되는 것으로 나타났다. 그러나 본 논문에서 구현한 T-Cache는 단일 노드에서만 데이터베이스와 동기화를 지원하고, 멀티 노드 간에 동기화는 지원하지 않는다. 향후 연구로 멀티노드 환경에서 특정 노드의 데이터 변경이나 삭제 시 다른 노드에서도 자동으로 데이터베이스와 동기화를 지원할 수 있는 알고리즘을 개발하고자 한다.

참고문헌

[1] http://cholmin0318.blog.me/70050127142

[2] OLTP Framework : http://www.curocom.com/english/bancs/bancs_architec.asp

[3] 오라클(Oracle) 데이터베이스서버 http://www.oracleclub.com/download.action?fileId=3091 :Oracle10g_Architecture.pdf

[4] Oracle Database Buffer Cache: http://blog.naver.com/kiyoun82?Redirect=Log&logNo=110068863372

[5] 한윤희, “유사 질의 매칭 기반 데이터베이스 캐쉬 엔진 설계 및 구현”, 한국산업기술대 지식기반기술·에너지대학원, 200802