클라이언트 캐싱 데이터 관리 시스템을 위한 캐쉬 일관성 알고리즘

김 치 연*

*목포해양대학교

A Cache Consistency Algorithm for Client Caching Data Management Systems

Chi-yeon Kim*

*Mokpo National Maritime University

E-mail : gegujang2@mmu.ac.kr

요 약

클라이언트의 캐쉬된 데이터 관리는 클라이언트 응용의 정확성을 보장하기 위해 필요하다. 캐쉬 일관성 알고리즘은 탐지 기반과 회피 기반의 두 부류가 있다. 탐지 기반의 캐쉬 일관성 방법은 일단 비일관된 데이터의 접근을 허용하고 완료하기 전 캐쉬된 데이터의 유효성을 검사한다. 반면, 회피 기반의 알고리즘 하에서 트랜잭션은 비일관된 데이터를 접근할 기회를 전혀 갖지 않는다. 이 논문에서는 회피 기반의 버전을 이용한 새로운 캐쉬 일관성 알고리즘을 제안한다. 제안하는 방법은 서버와 클라이언트에 이중 버전을 유지하여 단일 버전만 사용한 방법에 비하여 콜백으로 인한 메시지 교환을 제거하고 트랜잭션의 철회율을 감소시킬 수 있다. 또한 갱신 전달을 위하여 무효화와 전파를 혼합하여 사용함으로써 캐쉬 실패를 최소화하였다.

ABSTRACT

Cached data management of clients is required to guarantee the correctness of client's applications. There are two categories of cache consistency algorithms : detection-based and avoidance-based cache consistency algorithm. Detection-based schemes allow stale data access and then check the validity of any cached data before they can be allowed to commit. In contrast, under avoidance-based algorithms, transactions never have the opportunity to access stale data. In this paper, we propose a new avoidance-based cache consistency algorithm make use of version. The proposed method maintains the two versions at clients and servers, so it has no callback message and it can be reduced abort ratio of transactions compare with the single-versioned algorithms. In addition to, the proposed method can be decreased cache miss using by mixed invalidation and propagation for remote update action.

키워드

cache consistency, avoidance-based, version, invalidation, propagation

Ⅰ. 서 론

대부분의 데이터베이스 관리 시스템에 존재하는 데이터들은 다양한 형태로 분산되어 존재한다. 데이터는 서버들 사이에서 분산되어 저장되기도 하지만 이 논문에서 관심 있는 분산의 형태는 서버와 클라이언트 사이의 분산이다.

클라이언트는 캐쉬를 사용하여 서버와 데이터를 공유하고, 응답성을 향상시킬 수 있다[1]. 특히, 객체지향 데이터베이스 관리 시스템에서는 분

산 객체의 일관성을 위해 캐쉬 일관성 알고리즘이 중요하다[2].

캐쉬 일관성 알고리즘은 클라이언트에서 수행되는 트랜잭션 동시성 제어 알고리즘이라고도 할 수 있다. 지금까지 제안되어 온 캐쉬 일관성 알고리즘은 낙관적 방법과 비관적 방법으로 분류할 수 있다[2][3]. 낙관적 방법은 트랜잭션을 먼저 수행한 후 데이터의 유효성을 검사하는 방법으로 탐지 기반으로도 알려져 있으며, 비관적 방법은 트랜잭션 수행 전에 먼저 유효성을 검사하는 방법으로 회피 기반으로도 불린다. 낙관적 방법은 비관적 방법에 비하여 알고리즘이 비교적 단순하고 서버보다는 서버의 부담이 큰 반면 비관적 방법은 알고리즘이 복잡하고 클라이언트의 부담이 크며, 낙관적 방법보다는 많은 알고리즘들이 제안되어져 왔다.

클라이언트의 캐쉬 유형은 inter-transaction 캐쉬과 intra-transaction 캐쉬으로 분류할 수 있는데 전자는 트랜잭션의 종료와 상관없이 클라이언트가 삭제할 때까지 데이터를 캐쉬하며 수행되는 어플리케이션이 항상 데이터베이스의 일관성 있는 뷰를 보기 위하여 캐쉬 일관성 프로토콜을 필요로 한다. 후자는 하나의 트랜잭션을 수행하는 동안만 데이터를 캐쉬하며 구현이 쉽고 클라이언트 프로세스가 고유의 버퍼 풀을 관리하고 얻은 잠금을 추적하는 능력을 요구한다[transactional]. 그림 1은 클라이언트에서 캐쉬를 사용하는 클라이언트-서버 환경의 시스템 모델이다.

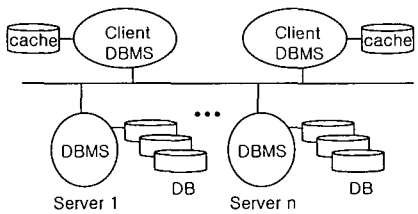

그림 1. 클라이언트-서버 환경 모델

이 논문에서는 회피 기반의 캐쉬 일관성 알고리즘을 기술한다. 제안하는 알고리즘이 기존의 알고리즘과 차별되는 부분은 클라이언트의 캐쉬와 서버에 이중 버전을 유지한다는 점과, 원격 클라이언트의 갱신을 반영하기 위해 일반적으로 사용하는 무효화(invalidation) 뿐 아니라 전파(propagation)를 동적으로 사용한다는 점이다. 캐쉬 일관성 알고리즘을 설계할 때 고려해야 할 중요한 부분은 성능인데 메시지 교환과 트랜잭션의 철회율 측면의 성능을 고려해야 한다. 캐쉬에 이중 버전을 유지하면 직렬화 그래프(SG : Serialization Graph)에 순환을 발생시키지 않으면서 too-late 연산을 수행시킬 수 있어, 트랜잭션 철회율을 낮출 수 있다. 또한 무효화와 전파를 동적으로 사용하게 되면 캐쉬에서 자주 접근하는 데이터의 무조건적인 삭제를 막을 수 있어 캐쉬 실패가 적어지고 그로 인한 서버와 클라이언트 사이의 메시지 교환이 감소할 수 있다.

이 논문의 구성은 다음과 같다. Ⅱ장에서는 캐쉬 일관성 알고리즘을 제안한 기존의 연구들에 대하여 살펴보고, Ⅲ장에서는 새롭게 제안하는 캐쉬 일관성 알고리즘에 대하여 기술한다. 마지막으로 Ⅳ장에서는 결론과 향후 연구방향을 기술한다.

Ⅱ. 관련연구

클라이언트-서버 환경에서 캐쉬 일관성을 다룬 알고리즘들을 분류해보면 표 1과 같다.

캐쉬 일관성 알고리즘은 클라이언트가 타당하지 않은 데이터의 접근을 방지하는 방법에 따라 탐지 기반과 회피 기반으로 분류할 수 있고, 클라이언트가 갱신 의도를 서버에 처음 알렸을 때 서버와의 상호작용 방식에 따라 분류하면 동기적, 비동기적, 그리고 지연된 방법으로 분류할 수 있다. 동기적 방법은 서버의 응답을 기다린 후 기록 연산을 수행하는 방법이며 비동기적 방법에서는 서버의 응답을 기다리지 않고 바로 연산을 수행한다. 지연된 방법은 트랜잭션의 수행을 모두 끝내고 완료 직전에 갱신 요구를 하는 방법이다. 이 외에도 캐쉬 일관성 알고리즘을 설계할 때 고려할 수 있는 항목으로 기록 잠금의 보유 시간과 갱신된 결과를 클라이언트에 알리는 방법 등이 있다. 기록 잠금의 보유 시간은 한 트랜잭션이 종료될 때까지만 유지되거나 클라이언트가 명확히 데이터를 삭제(무효화)할 때까지 계속 유지하는 방법을 고려할 수 있다. 마지막으로 갱신 결과의 전달 방법은 무효화나 전파가 주로 사용된다. 이 절에서는 이들 중 대표적 알고리즘들과 2V-CBL 방법[4]에 대하여 살펴본다.

표 1. 캐쉬 일관성 알고리즘의 분류

구분	Synchronous	Asynchronous	Deferred
Avoidance-based	ACBL[5]	AACC[2]	O2PL[6]
Detection-based	C2PL[6]	NWL[7]	AOCC[8]

• ACBL : 이 방법은 inter-transaction 캐쉬을 사용하며 잠금(lock)을 사용하여 트랜잭션의 동시성을 제어한다. 이 방법은 회피 기반 알고리즘이므로 일관되지 않은 데이터를 접근하지 않으나, 교착상태의 발생 가능성이 있다.

• AACC : 비동기적 방법으로 ACBL 방법을 변형시킨 알고리즘이다. 서버와 클라이언트에서 각기 다른 레벨의 잠금을 관리하고 잠금의 충돌이 발생할 때 교착상태를 탐지하는 알고리즘을 수행한다. 캐쉬에 없는 데이터를 처음으로 캐쉬할 때 일단 전용 판독 모드로 잠금을 얻고 기록을

위한 기록 잠금을 나중에 callback을 이용하여 잠금 상승을 함으로써 불필요한 callback 메시지가 발생한다.

• 2V-CBL : CBL방법의 한 변형으로 두 개의 버전을 사용한 회피 기반의 2V-CBL 알고리즘이다. 이 방법에서는 트랜잭션의 잠금을 세분화하여 동시성을 높임으로써 특히 읽기 전용 트랜잭션의 크기가 작을 때 AACC 방법보다 성능이 나음을 증명하였다.

III. 제안하는 알고리즘

이 장에서는 회피 기반의 새로운 캐쉬 일관성 알고리즘(TACC : Two-version Avoidance-based Cache Consistency Algorithm)에 대하여 기술한다.

3.1 클라이언트 알고리즘

클라이언트에서 수행해야 할 첫 번째 기능은 캐쉬에 없는 데이터에 대한 요구이다. 클라이언트는 접근하려는 데이터가 캐쉬에 존재하지 않으면 접근하기 위한 잠금 모드와 더불어 데이터 식별자를 서버에 보낸다. 데이터 요구에 대하여 서버는 충돌 여부를 검사하여 적절한 버전의 데이터를 전달하거나 수행이 불가능하다고 판단되는 경우는 트랜잭션을 블록시킨다.

접근하려는 데이터가 캐쉬에 있고 수행하려는 연산이 판독인 경우는 캐쉬에 있는 버전을 접근하여 항상 수행이 가능하고, 기록 연산인 경우는 서버에 요구했을 때 이미 설정되어 있는 잠금 상태에 따라 즉시 수여되거나 블록될 수 있다.

트랜잭션이 완료되거나 철회되면 수행한 트랜잭션에 대한 로그를 서버에 보내 서버에 유지되는 각종 자료구조를 갱신하도록 한다.

서버로부터 원격 클라이언트가 갱신한 데이터가 전달되면 클라이언트는 무효화와 전파를 동적으로 선택한다. 클라이언트가 현재 접근하지 않는 데이터 중, 자주 사용되는 데이터인 경우는 전파를, 자주 사용되는 데이터가 아닌 경우는 무효화를 적용하여 가능한 한 캐쉬 실패가 적게 발생하도록 한다.

3.2 서버 알고리즘

클라이언트의 이중 버전을 지원하기 위해 서버 또한 데이터에 대한 이중 버전을 유지한다. 새로운 버전을 기록 연산이 성공적으로 수행될 때마다 만들어지며 버전의 특성상 기존의 판독 연산에 이어지는 기록 연산은 충돌 유형에서 제외된다[9]. 버전은 일종의 완전히 다른 데이터 항목으로 간주되기 때문이다. 버전 자체의 생성과 삭제는 일반적인 버전 관리 방법과 동일하다.

서버는 먼저, 클라이언트의 데이터 요구에 대하여 응답한다. 클라이언트가 캐싱을 위해 요구한 데이터에 잠금이 전혀 설정되어 있지 않다면 요구된 데이터의 최신버전을 클라이언트에게 전달하며, 잠금 테이블과 트랜잭션_캐쉬_리스트를 갱신한다. 트랜잭션_캐쉬_리스트는 각 데이터 항목에 대하여 유지되며 데이터를 캐쉬하고 있는 클라이언트의 식별자를 유지한다. 잠금 테이블에는 서버에 유지되는 각 데이터에 대하여 설정된 잠금을 유지한다.

다른 클라이언트에 의해 기록 잠금이 설정된 데이터에 대한 판독 잠금을 요구하는 경우, 신버전 대신 구버전을 제공한다. 이미 기록 연산이 수행중이었다면 직렬화 순서에서 앞선 트랜잭션이 존재함을 의미하기 때문에 최신버전 대신 구버전을 제공하는 것이다.

다른 클라이언트에 의해 기록 잠금이 설정된 데이터에 대한 기록 잠금을 요구하는 경우, 트랜잭션은 직렬화 순서를 유지하기 위해 블록된다. 블록된 트랜잭션은 대기 그래프에 유지되며 선행하는 트랜잭션이 완료된 후에 수행가능하다.

서버에 이중 버전을 유지하게 되면, 단일 버전을 유지한 알고리즘에서 기록 연산을 수행하기 위해 데이터를 캐쉬하고 있는 모든 클라이언트들에게 무효화 여부를 묻기 위해 보내졌던 콜백 메시지가 사라지게 된다. 요구된 기록 잠금은 클라이언트와 상관없이 서버 잠금 테이블의 상태에 따라 즉시 수여되거나 블록된다.

완료한 트랜잭션의 로그가 클라이언트로부터 도착하면 잠금 테이블 및 캐쉬된 클라이언트의 리스트, 대기 그래프에서 해당 엔트리를 삭제하고, 블록된 트랜잭션 중 활성화시킬 트랜잭션이 있는지 찾는다. 완료된 트랜잭션 안에 갱신된 데이터가 있으면 그 데이터를 캐쉬하고 있는 모든 클라이언트들에게 알려 최신 버전을 캐쉬에 반영하도록 한다.

예를 들어, 다음과 같은 시나리오에 대하여 단일 버전 알고리즘에서는 하나의 트랜잭션이 철회되나 제안하는 알고리즘에서는 모든 트랜잭션이 완료될 수 있다.

(시나리오) TACC 알고리즘의 예

하나의 서버와 세 개의 클라이언트 C1, C2, C3를 가정한다. 각 클라이언트에서는 다음과 같은 트랜잭션 T1, T2, T3가 각각 수행되며 히스토리 H와 같이 수행된다고 가정한다. 서버에는 데이터 x, y, z가 저장되어 있으며, 처음 수행을 시작할 때 모든 클라이언트의 캐쉬에는 어떤 항목도 저장되지 않았다고 가정한다.

T1 : r1(x) r1(y) w1(y)
T2 : w2(x) r2(y)
T3 : w3(x) w3(y) w3(z)
H : r1(x)w3(x)w3(y)w2(x)r1(y)w1(z)r2(y)

r1(x)는 서버에서 최신 버전을 요구하여 수행되고, w3(x)는 새로운 버전의 생성이므로 잠금 테이블을 갱신한 후 수행된다. w3(y)는 기존에 설정된 잠금이 하나도 없으므로 수행가능하고 w2(x)는 w3(x) 때문에 블록된다. r1(y)가 수행되는 시점에서 단일 버전 방법에서는 교착 상태가 발생하여 T1이나 T3중 하나의 트랜잭션이 철회되어야 하나, 제안하는 알고리즘에서는 r1(y)가 구버전을 접근하는 것으로 수행가능하다. 그래서 결과적으로 세 개의 트랜잭션이 모두 완료가능하다. □

IV. 결론 및 향후 연구방향

이 논문에서는 이중 버전을 이용한 회피 기반의 캐쉬 일관성 알고리즘을 제안하였다. 캐쉬 일관성 알고리즘은 데이터가 서버와 클라이언트에 분산되어 있는 환경에서 매우 중요한 기술이다.

지금까지 제안된 캐쉬 일관성 알고리즘은 탐지 기반과 회피 기반으로 분류할 수 있다. 탐지 기반은 알고리즘이 간단한데 비하여 낙관적 가정에 근거하여 충돌이 드문 환경이 아니면 좋은 성능을 발휘하기 어렵다. 그래서 일반적으로 회피 기반에 대한 연구가 많이 이루어져 왔다.

회피 기반의 캐쉬 일관성 알고리즘을 설계할 때 고려할 점은 기록 의도를 서버에 알린 후 상호작용하는 방법 및 기록 잠금의 보유 시간, 그리고 갱신 결과의 전달 방법 등이다. 이 논문에서는 회피 기반의 비동기적 방법, 클라이언트가 무효화할 때까지 기록 잠금을 보유하고, 갱신 결과의 전달 방법은 무효화와 전파를 혼합한 방법을 사용하였으며, 클라이언트 트랜잭션의 철회율을 낮추기 위해 이중 버전을 적용하였다.

지금까지 제안된 회피 기반의 알고리즘 중 가장 좋은 성능을 인정받고 있는 알고리즘은 AACC 방법이다. 이 논문에서 제안한 알고리즘과 AACC 알고리즘을 메시지 교환 측면과 철회율 측면에서 비교하면 다음과 같다.

메시지 교환 측면에서 우선 두드러지는 부분은 서버에서 기록 잠금을 수여하기 위해 데이터를 캐쉬하고 있는 모든 클라이언트에게 보내던 콜백이 제거되었다는 점이다. 물론 콜백을 다른 메시지에 피기백(piggyback)해서 보낸다고는 하나, 이 역시 하나의 메시지로 인정되므로 콜백이 없어지면 그 만큼의 메시지 교환은 감소한다고 할 수 있다.

철회율의 측면에서 살펴보면 III장의 시나리오에서 보는 것과 같이 직렬성에 위배되지 않으면서 접근가능한 버전을 찾을 수 있어 트랜잭션의 철회율이 낮아진다. 또한 갱신 결과를 원격 클라이언트들에게 전달하기 위해 무효화와 전파를 혼합하여 사용하였는데, 클라이언트에서 자주 접근하는 데이터에 전파를 적용하게 되면 무조건 무효화하는 것 보다 캐쉬 실패 확률이 낮아져 캐쉬 실패로 인한 메시지 교환도 감소시킬 수 있다.

제안하는 방법이 갖는 오버헤드는 서버와 클라이언트에 이중 버전을 저장하기 위해 필요한 공간에 대한 부담이다. 이러한 부담이 전체적인 성능에 미치는 정확한 영향은 모의실험을 통해 보여져야 할 것으로 생각된다.

참고문헌

[1] M. Franklin, M. Livny, and M. Carey, "Client-Server Caching Revisited", Distributed Object Management, Morgan Kautmann, 1994.

[2] M. T. Ozsu, K. Voruganti, and R. C. Unrau, "An Asynchronous Avoidance-based Cache Consistency Algorithm for Client Caching DBMSs," Proceedings of the 24th VLDB Conference, pp. 440-451, 1998

[3] M. J. Franklin, M. J. Carey, and M. Livny, "Transactional Client-Server Cache Consistency : Alternatives and Performance," ACM Transactions on Database Systems, Vol. 22, Num. 3, pp 315-363, 1997.

[4] 강홈근, 민준기, 전석주, 정진완, "클라이언트-서버 DBMS 환경에서 콜백 잠금 기반 다중 버전의 활용," 정보과학회 논문지, Vol. 31, Num. 5, pp. 457-467, Oct. 2004.

[5] Carey, M., A. Adya, B. Liskiv, and M. Zaharioudakis. "Fine Grained Sharing in a Page Server OODBMS", In Prod. ACM SIGMOD International Conference on Management of Data. pp. 359-370, 1994.

[6] Carey, M., M. Franklin, and M. Zaharioudakis, and E. Shekita, "Data Caching Tradeoff in Client-Server DBMS Architectures", In Proc. ACM SIGMOD International Conference on Management of Data, pp. 357-366, 1991.

[7] Wang, Y. and L. Rowe, "Cache Consistency and Concurrency Control in a Client/Server DBMS Architecture, In Proc. ACM SIGMOD International Conference on Management of Data, pp. 367-376, 1991.

[8] A. Adya, R. Gruber, B. Liskov, and U. Maheshwari, "Efficient Optimistic Concurrency Control Using Loosely Synchronized Clocks," ACM SIGMOD Conference, pp. 23-34, June, 1995.

[9] P. A. Bernstein, V. Hadzilacos, and N. Goodman, "Concurrency Control and Recovery in Database Systems," Addison-wesley, 1987.