MySQL Insert Buffer 기반 SSD 쓰기 최적화

이황교° , 이상원
성균관대학교 컴퓨터공학과
gt369kr@skku.edu, swlee@skku.edu

SSD Write Optimization using MySQL Insert Buffer

Hwanggyo Lee° , Sang-won Lee
Dept of Computer Engineering, Sungkyunkwan University

요 약

MySQL의 InnoDB 엔진에서 사용하는 Insert Buffer는 비군집 보조 인덱스로 인해 발생하는 랜덤한 저장 장치 입출력을 회피, 절감하여 데이터베이스 시스템의 성능을 향상시키는 데에 중요한 역할을 하고 있다. 하지만, 기존의 Insert Buffer의 동작 방식은 저장 장치를 하드디스크로 사용할 때를 가정하고 구현되어 있어, SSD 상에서 사용할 경우 그 동작 방식이 SSD에는 다소 맞지 않는 경향이 있다. 본 논문에서는 Insert Buffer를 SSD에 맞게 수정하여 SSD에서 발생하는 쓰기 연산을 최적화하는 아이디어를 제시한다.

1. 서 론

MySQL의 InnoDB 엔진은 B+ 트리(Tree) 인덱스(Index) 기반으로 동작한다. 테이블(Table)에 레코드(Record)들이 새로 삽입 또는 갱신되는 경우, 해당 테이블이 가지고 있는 하나 이상의 보조 인덱스(Secondary Index)에도 변경된 정보가 갱신되어야 한다. 보조 인덱스는 비군집성(Non-clustered)을 지니기 때문에, 갱신 과정에서 랜덤(Random)한 입출력을 유발할 수 있고, 이는 데이터베이스 시스템의 성능을 크게 저하시킬 수 있는 문제이다. InnoDB에서는 Insert Buffer라는 자료 구조를 통해 보조 인덱스가 유발하는 랜덤 입출력을 통제하고, 성능을 향상시킨다.

Insert Buffer가 구현될 당시에는, 하드디스크가 주요 저장매체였기 때문에, Insert Buffer의 동작 역시 하드디스크의 입출력 방식에 맞게 구현되어있다. 최근 널리 사용되는 저장매체인 SSD(Solid State Drive)는 하드디스크와는 상이한 동작 방식을 가지기 때문에, 기존의 Insert Buffer 동작 방식은 SSD와는 다소 맞지 않는 부분이 있다. 본 논문은 Insert Buffer의 동작을 SSD에 더 잘 맞게끔 최적화하는 아이디어를 제시한다.

본 논문의 2장에서는 MySQL InnoDB에서 사용하는 Insert Buffer의 동작 원리에 대해 다룬다. 3장에서는 하드디스크와 구분되는 SSD의 특징을 다루고, 이것이 Insert Buffer의 동작과 연결했을 때 어떤 문제점을 불러올 수 있는지 서술한다. 4장에서는 3장의 내용을 바탕으로 SSD의 성능을 최적화하기 위한 Insert Buffer의 변경 아이디어를 간단히 설명한다.

2. Insert Buffer의 동작

2.1. 엔트리의 저장

테이블에 순차적으로 다수의 레코드를 삽입하는 경우, 주 인덱스(Primary Index)는 레코드가 그 자체로 인덱스의 엔트리(Entry)가 되어, 엔트리의 순서와 레코드의 순서가 일치하는 군집성(Clustered)을 가지게 된다. 따라서 엔트리들이 동일 페이지(Page) 또는 인접한 페이지에 저장되므로 페이지 입출력도 그에 맞게 순차적으로 발생한다. 하지만, 비군집성을 가지는 보조 인덱스는 엔트리들의 순서가 레코드의 순서와 동일하다는 보장이 없으므로, 엔트리들이 임의의 여러 페이지에 흩어져있을 수 있고, 만약 필요한 페이지가 메모리에 존재하지 않아 저장 장치에 접근해야 하는 경우, 랜덤 입출력이 발생하게 된다. Insert Buffer는 InnoDB 전체를 관리하는 시스템 테이블스페이스(System Tablespace)에 저장되는 별도의 인덱스로, 위처럼 보조 인덱스로 인한 랜덤 입출력이 발생하는 경우, 페이지 탐색을 위한 저장 장치 접근을 막고, 대신 해당 엔트리들을 테이블스페이스 별, 페이지 별로 정렬하여 대신 저장해두는 역할을 한다[1].

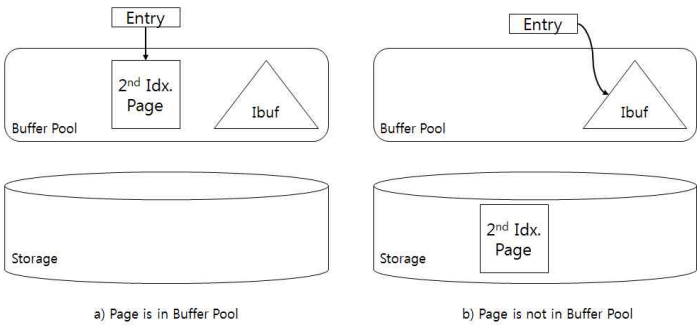

그림 1 Insert Buffer의 동작

2.2. 병합

Insert Buffer에 저장된 인덱스 엔트리들은 최종적으로는 실제 인덱스에 반영, 저장되어야 할 내용이다. 다른 읽기 작업 등으로 인해 보조 인덱스의 페이지가 메모리로 읽혀져 오면, Insert Buffer를 탐색하여 해당 페이지에 대한 엔트리들을 인덱스에 반영하는 작업을 하는데, 이를 병합(Merge) 작업이라고 한다[1]. Insert Buffer가 최대

크기에 도달하여 전체 내용을 병합하는 경우도 있으나, 대부분의 병합 작업은 읽혀져 온 페이지에 대해서 그 즉시 발생한다. Insert Buffer 내의 엔트리들은 그 양에 관계없이 모두 병합되며, 병합된 엔트리는 곧바로 Insert Buffer에서 삭제된다.

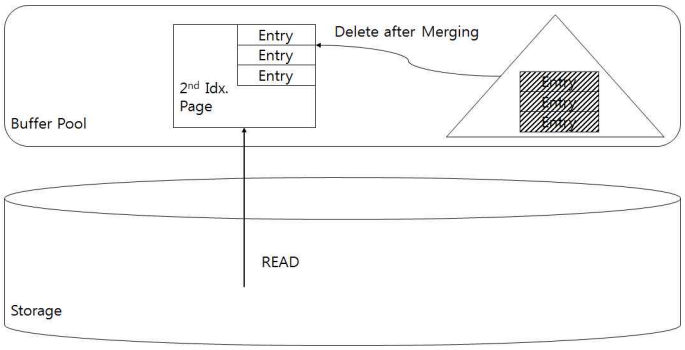

그림 2 Insert Buffer의 병합

3. SSD와 Insert Buffer

3.1. SSD의 특성

SSD는 다수의 플래시 메모리(Flash memory) 칩 및 이를 제어하는 프로세서, 펌웨어 등으로 구성된 저장 장치이다. 하드디스크와 비교했을 때, SSD만이 가지는 대표적인 특성은 다음과 같다[2].

- 읽기와 쓰기의 성능이 비대칭이다.

하드디스크는 읽기와 쓰기의 성능이 거의 동일한 반면, SSD는 읽기 연산이 쓰기에 비해 몇 배 이상 빠르게 동작한다. 또, 하드디스크는 읽기와 쓰기 모두 순차적으로 발생하는 경우가 랜덤한 경우에 비해 더 빠른 반면, SSD는 읽기 연산에서는 순차적 접근과 랜덤 사이의 차이가 거의 없다. 쓰기 연산은 SSD 역시 랜덤한 경우가 더 느리게 나타난다.

- 덮어쓰기가 불가능하다.

이미 데이터가 써져 있는 영역에 바로 새로운 데이터를 덮어씌울 수 있는 하드디스크와 달리, SSD는 바로 덮어쓰는 것이 불가능하다. 그래서 SSD에서는 삭제 연산을 별도로 지원하지만, 삭제 연산은 읽기나 쓰기에 비해 수십~수백 배 느린 연산으로 자주 발생하게 되는 경우 성능이 크게 저하된다.

- 기록 가능 횟수가 제한적이다.

반영구적으로 데이터를 반복 기록할 수 있는 하드디스크와 달리, SSD는 제한된 횟수 이상으로 쓰기/삭제가 반복되면 해당 영역은 수명이 다해 사용할 수 없게 된다.

- 기본 연산 단위가 상대적으로 크다[3].

섹터(Sector) 단위로 데이터 접근이 가능한 하드디스크와 달리, SSD는 페이지(Page) 단위로만 읽기/쓰기 연산을 수행할 수 있다. 삭제 연산의 경우는 다수 페이지의 집합인 블록(Block) 단위로만 수행된다. 덮어쓰기가 불가능한 특성과 이어보면, 페이지 내의 극히 일부분만 수정이 가해져도 전체 페이지를 무효화(Invalidation)해야 하고, 해당 페이지를 즉각 삭제 가능한 것이 아니기 때문에 실제 데이터는 논리적 주소와 다른 물리 페이지에 기록되는 경우가 많아진다. 따라서 이를 관리할 별도의 매핑 테이블(Mapping table)이 필요하게 된다.

3.2. SSD에서 보조 인덱스 및 Insert Buffer의 동작

Insert Buffer는 2장에서 언급했듯이, 갱신하려는 페이지가 메모리에 없을 경우, 즉 쓰기를 수행하려는 페이지를 저장 장치에서 읽어 와야 하는 경우에 사용된다. 부가적으로 쓰기 연산을 줄여주는 효과도 따라오긴 하지만, 주된 목적은 랜덤한 읽기 연산을 회피하는 것이다. 3장 1절에서 언급했듯이, SSD의 경우는 읽기 연산이 쓰기보다 더 빠르며, 랜덤한 읽기 연산의 성능 역시 매우 빠르기 때문에 SSD에서 Insert Buffer를 최적화하기 위해서는 읽기보다는 쓰기 연산, 특히 랜덤한 쓰기의 부하를 감소시키는 것에 집중할 필요가 있다.

InnoDB의 보조 인덱스 엔트리는 테이블의 PK(Primary key)와 인덱싱할 컬럼(Column)의 쌍으로 이루어져 있다. 따라서 보조 인덱스 엔트리는 수~수십 바이트(Byte) 내외의, 기본 16KB 크기를 가지는 InnoDB의 페이지에 비해 매우 작은 크기를 지닌다. 따라서 매우 적은 양의 엔트리만 갱신된 보조 인덱스 페이지가 SSD에 다시 기록되는 경우, 덮어쓰기가 불가능한 특성상 몇 십 바이트에 지나지 않은 변경 사항 때문에 16KB 페이지 전체가 무효화되어야 한다. 대다수의 상용 SSD는 4KB 페이지를 사용하고 있으므로, 실제 SSD 상에서는 4개의 페이지가 무효화될 수 있는데, 이 중에는 물리적으로는 아무 변경사항이 없는 페이지도 포함되어 굉장히 비효율적으로 동작하게 된다. 무효화된 페이지가 많아지는 것은 SSD 내에 삭제해야할 페이지가 많아진다는 뜻인데, 앞서 언급했듯이 SSD의 삭제 연산은 읽기나 쓰기에 비해 현저히 느리게 동작하므로, 위와 같은 현상은 단순히 페이지 공간 활용의 비효율뿐만 아니라 실제 성능에도 악영향을 미치게 된다. 설령 인덱스 페이지에 바로 적용된 것이 아니라 Insert Buffer에 저장된 경우라 해도, Insert Buffer 역시 동일한 16KB 페이지를 사용하는 자료 구조이므로 같은 현상이 발생할 수 있다.

4. 최적화 아이디어

3장에서, SSD의 입출력 성능에서의 주요 부분은 읽기 연산보다는 쓰기와 이에 뒤따라오는 삭제 연산을 제어하는 데에 있다는 점, 특히 극소량의 변화로 인해 페이지 전체가 무효화되어 발생하는 비효율을 줄이는 것에 있다는 것을 확인했다. 간단히 생각해서, 소량의 변화들을 최대한 축적시켜, 일정 수준 이상 많이 모였을 때 한 번에 적용시킬 수 있다면, 위의 문제점은 상당히 해결될 것이다. 이 사항을 기반으로 아래처럼 Insert Buffer를 최적화하는 아이디어를 제시한다.

- 모든 엔트리 변경 사항을 Insert Buffer에만 저장한다.

기존의 방식에서는 버퍼 내 인덱스 페이지의 존재 유무에 따라 선택적으로 기록했기 때문에, 인덱스 페이지가 버퍼 내에 존재하는 경우에는 소량, 극단적으로 단 하나의 엔트리만 갱신된 경우에도 페이지 전체를 무효화시킬 수 있었다. 무조건 적으로 Insert Buffer에 저장하게 하는 것은 앞서 말한 대로 변경 사항을 최대한 많이 축적하기 위함이고, 전체 데이터베이스에 대한 내역을 Insert Buffer라는 제한된 영역에 저장하므로 Insert Buffer의 페이지들은 자체적으로 대량의 변경 사항을 지닐 수밖에 없어질 것이다.

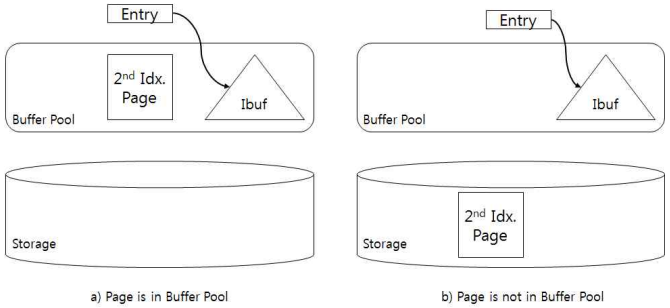

그림 3 Insert Buffer 저장 방식의 변경

- 대신, 병합 작업을 선택적으로 진행한다. 기존의 병합 방식은 페이지가 버퍼에 읽어들어져 올 때, Insert Buffer를 탐색, 해당 페이지에 저장되어야 할 모든 사항을 바로 적용시키는 방식이어서, Insert Buffer에 저장된 변경 사항이 매우 적은 경우에는 앞서 말한 비효율적인 페이지 무효화 작업을 일시적으로 뒤로 미룬 것에 지나지 않게 된다. 그러므로 병합 작업으로 원본 페이지의 최신판을 만들 때 Insert Buffer 내부에 저장된 엔트리의 양도 파악하여, 일정 수준 이상 쌓여 변경 사항이 많은 경우에는 기존처럼 만들어진 최신의 페이지를 저장 장치에 반영한 뒤 Insert Buffer 내부에 저장된 것은 삭제하고, 그렇지 않은 경우에는 최신 페이지의 복사본만 버퍼 상에서 사용한 뒤, 이후 저장장치에 반영하는 작업은 무시하게끔 구현한다.

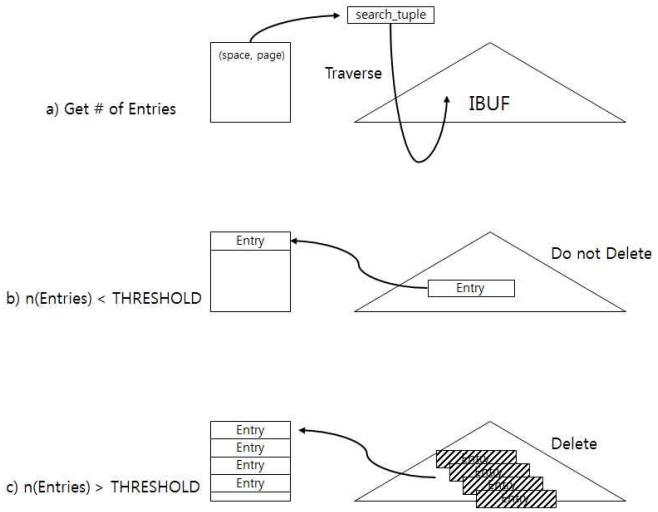

그림 4 Insert Buffer 병합 방식의 변경

제시한 방법과 같이 변경하였을 때, 기대하는 효과는 다음과 같다.

- 쓰기 요청 수의 감소

병합 과정에서, 병합되는 엔트리의 양에 따라 최신 페이지를 저장장치에 쓰지 않을 수 있으므로, 이에 따른 쓰기 감소 효과를 기대한다.

- Insert Buffer 접근 수의 증가

선택적 사용이 아닌, 무조건적 접근을 하도록 구현하므로 상대적으로 Insert Buffer에 삽입, 병합 등의 과정에서 접근하는 횟수가 늘어날 것이며, 이로 인한 읽기/쓰기의 증가도 발생할 수 있다.

- 전체 성능 지표 향상

Insert Buffer는 전체 데이터베이스에 비해 극히 제한된 크기를 가지므로, 올바른 구현이 되었다는 가정 하에는, Insert Buffer에 요청되는 입출력의 증가량 보다 전체 데이터베이스에서 감소하는 쓰기 요청 수가 더 클 것이다. 따라서 전체 입출력 요청이 감소하는 효과가 있을 것이고, 이에 따라 시스템 성능도 향상될 것이다.

5. 결론

MySQL InnoDB 엔진에서 사용하는 Insert Buffer는 보조 인덱스 동작 과정에서 발생하는 비효율성을 절감 또는 제거하여 데이터베이스 시스템 전체 성능을 향상시키는 데에 중요한 역할을 하는 자료 구조이다. 다만, 당시 하드디스크 기준으로 구현된 사항들이 현재의 SSD 상에서 효율적이지 못한 부분이 있어 이를 개선하면 지금보다 더 좋은 성능을 이끌어낼 잠재성을 지니고 있다. SSD가 데이터베이스 시장에서 중요도가 계속 높아지는 만큼, SSD에 최적화된 데이터베이스 시스템은 앞으로 경쟁력을 지니게 될 것이고, 본 논문의 연구 역시 그러한 이유에서 의미를 지니게 될 것이다.

사사

본 연구는 미래창조과학부 및 정보통신기술진흥센터의 SW컴퓨팅산업원천기술개발사업(SW스타랩)의 연구결과로 수행되었음“ (IITP-2015-0-00314)

참고 문헌

[1] The InnoDB Change Buffer, http://mysqlserverteam.com/the-innodb-change-buffer/
[2] Sang-won Lee, Bongki Moon, “Design of Flash-Based DBMS: An In-Page Logging Approach”, ACM SIGMOD, 2007
[3] Woon-Hak Kang, Sang-Won Lee, Bongki Moon, Yang-Suk Kee, Moonwook Oh, “Durable Write Cache in Flash Memory SSD for Relational and NoSQL Databases“, ACM SIGMOD 2014