MySQL InnoDB엔진의 Secondary Index Scan 성능 향상을 위한 Prefetch 기능 구현

황다솜○ 이상원
성균관대학교
{rhcqnssp32, swlee}@skku.edu

An Implementation of a Prefetch Algorithm for a Secondary Index in MySQL InnoDB Engine

Da-som Hwang○ Sang-Won Lee
Sungkyunkwan University

요 약

플래시 SSD(Flash-based Solid-State Drive)는 높은 에너지 효율성, 외부 충격에 강한 내구성, 높은 I/O처리량뿐만 아니라 단위 용량 당 가격 하락으로 모바일 기기뿐만 아니라 데스크톱, 서버 등 여러 분야에서 하드디스크를 대체하는 저장장치로 자리매김하고 있다. 하지만 Flash SSD 저장장치 환경에서도, MySQL InnoDB엔진에서 secondary 인덱스를 이용한 스캔을 할 때, CPU의 I/O wait 비율이 사용되고 있는 CPU의 80%이상을 차지한다는 점을 실험을 통해 확인하였다. 본 논문에서는 MySQL InnoDB엔진의 secondary 인덱스를 이용한 스캔의 성능을 향상시키기 위해 Prefetch기능을 구현하여 I/O 지연시간을 감소시켜 읽기 성능을 약 3.1배 증가시켰다.

1. 서 론

플래시 SSD(Flash-based Solid-State Drive)는 높은 에너지 효율성, 외부 충격에 강한 내구성, 높은 I/O처리량 등 여러 장점을 가지고 있다. 또한 단위 용량 당 가격하락으로 플래시 SSD는 모바일 기기뿐만 아니라 데스크톱, 서버 등 여러 분야에서 하드디스크를 대체하는 저장장치로 자리매김하고 있다[1].

MySQL InnoDB엔진은 모든 테이블을 B-tree인덱스 형태로 저장하고 테이블에 접근할 때에도 primary key를 이용한다. 따라서 대부분의 스캔은 순차적 I/O를 발생시키고 저장장치 종류에 상관없이 짧은 지연시간을 보장할 수 있다.

이에 반해, secondary 인덱스를 이용한 인덱스 스캔 시, 하드디스크 저장장치 환경에서는 플래시 SSD 저장장치 환경에서보다 짧은 지연시간을 보장 할 수 없다. 이는 secondary 인덱스가 비 군집 인덱스(non-clustered index)이고 secondary 인덱스를 이용한 스캔은 랜덤 I/O를 유발하기 때문이다.

하지만 플래시 SSD환경에서 secondary 인덱스를 이용한 스캔 실험을 수행한 결과, 실험을 진행하는 동안의 총 CPU활용률 25% 중 I/O wait의 비율이 약 22%로 사용되고 있는 CPU의 80%가 I/O처리를 기다리고 있음을 확인 하였다(표 1).

본 논문에서는 오픈소스 데이터베이스 시스템인 MySQL InnoDB 엔진에서 secondary 인덱스를 이용한 스캔을 위해 Prefetch 기능을 구현하였다. Prefetch 기능 구현으로 플래시 SSD의 응답속도를 감소시켜 secondary 인덱스를 이용한 읽기의 성능향상을 기대할 수 있다. 실험을 통해 확인한 결과, Prefetch 단위가 128일 때, I/O wait의 비율이 약 1%로 감소하였고(표 1) secondary 인덱스를 이용한 읽기 성능은 약 3.1배 향상되었다 (그림 2).

본 논문의 구성은 다음과 같다. 2장에서 관련 연구를 소개하고 3장에서 Prefetch 구현에 대해 자세히 설명한다. 4장에서 실험 및 실험 결과에 대해 분석하고 마지막으로 5장에서 논문의 결론을 맺는다.

표 1 CPU 사용률

CPU / InnoDB	User	System	IO-wait	Idle
Clean	1.52	1.97	21.73	74.78
Prefetch-128	9.09	8.56	1.03	81.31

2. 관련 연구

2.1 PostgreSQL에서의 인덱스 스캔 최적화 기법[2]

[2]는 오픈소스 관계형 데이터베이스 시스템인

```
알고리즘 1: InnoDB Page Read
1:  Open  a cursor pcur at an index
2:  READ:
3:      while pcur points to a leaf page
4:          Move pcur at a proper position in a page
5:          Read a child page
6:  Read a record r from the leaf page
7:  if r != a primary index record then
8:      if need to read primary index then
9:          Extract a primary key pk from r
10:         goto READ
11:     else
12:         return r
13: else
14:     return r
```

PostgreSQL에 정렬 인덱스 스캔과 Parallel synchronous I/O를 구현하여 비 군집 인덱스에 대한 스캔의 성능을 향상시켰다. 정렬 인덱스 스캔으로 비 군집 인덱스를 접근 할 때 발생하는 무작위 접근을 순차적 접근으로 변환하고 Parallel synchronous I/O를 통해 한 번에 여러 I/O를 요청한다. 정렬을 통해 필요한 데이터 페이지에 대한 접근은 최대 1번으로 줄이고 Parallel synchronous I/O로 저장장치의 내부 병렬성을 활용하여 저장장치의 사용률을 증가시켰다. [2]는 비 군집 인덱스에 대한 스캔 성능을 기존의 PostgreSQL 대비 최대 약 3.8배 향상시켰다.

2.2 MySQL InnoDB의 읽기 메커니즘 [3]

알고리즘 1은 MySQL InnoDB 인덱스 스캔 알고리즘의 pseudo 코드이다. MySQL InnoDB는 테이블을 primary 키에 대한 B-tree형태로 저장을 하고 기본적으로 primary 키를 이용한 인덱스 스캔을 사용한다.

쿼리에 따라 secondary 인덱스를 이용한 인덱스 스캔을 할 경우에는 secondary 인덱스를 탐색하여 secondary 인덱스 레코드를 획득한 후, 해당 레코드에서 primary 킷 값을 추출한다. 추출한 값으로 primary 인덱스를 탐색하여 데이터 레코드를 획득한다.

Primary 인덱스를 이용하여 데이터에 접근할 경우, 접근 패턴이 순차적이기 때문에 짧은 지연시간을 기대할 수 있다. 하지만 secondary 인덱스를 이용할 경우, secondary 인덱스 레코드에서 추출한 primary 킷값들이 무작위적이고, 그러므로 실제 데이터 페이지를 읽기 위한 접근 패턴도 무작위적이다. 따라서 secondary 인덱스를 이용하여 데이터에 접근할 때에는 짧은 지연시간을 기대하기 어렵다.

3. Prefetch 구현

본 논문에서는 MySQL InnoDB엔진에서 secondary 인덱스를 이용한 인덱스 스캔을 할 경우, 이 때 발생하는 랜덤 읽기로 인해 시스템의 응답시간이 매우 길어진다. 이를 개선하고자, Libaio 라이브러리를 활용한 Asynchro--nous I/O를 통해 Prefetch를 구현하였다. 알고리즘 2는 본 논문에서 구현한 Prefetch를 간략히 나타낸 Pseudo 코드이다. Prefetch 구현은 알고리즘 1의 8과 9사이에 추가 되었고 플래그를 통해 활성화 또는 비활성화 시킬 수 있다.

```
알고리즘 2: Prefetch
1:  while Prefetch threshold <
        the number of records cnt in a list rec_list
2:      Store r in rec_list
3:      Extract primary keys pks from rec_list
        and store them into pk_list
4:  Open a cursor pfcur at an index
5:  while !end of rec_list Level-1-Read:
6:      Level-1-Read:
7:          Move pfcur at a proper position in a
            page p
8:          if p is a level-1-page then
9:              Store child page number pg_num
                into a list page_no_list
10:         else
11:             Read a child page
12:             goto Level-1-Read
13: Sort and remove duplications in page_no_list
14: Do prefetch with page_no_list
```

표 2 실험 환경

운영체제	Ubuntu 14.04
프로세서	Intel Core i5, 3.40GHz
메모리(RAM)	8GB
저장장치	Samsung 850Pro(256GB)
데이터베이스	MySQL InnoDB 5.6
TPC-H Scale Factor	10

```
SELECT * FROM table FORCE INDEX (idx)
WHERE colum_a BETWEEN min AND MAX ;
```

그림 1 예시 Range 쿼리

4. 성능 평가

4.1 실험 환경

표 2은 본 논문에서 실시한 실험에 대한 실험 환경을 정리한 표이다. 성능 평가를 위해 TPC-H 벤치마크의 orders 테이블을 이용하였다. 실험 쿼리로는 secondary 인덱스를 이용하는 range 쿼리를 사용하였다. 그림 1은 예시 쿼리를 나타낸 것이다. *FORCE INDEX* 명령어는 테이블에 접근 할 때, 해당 인덱스을 이용하라는 MySQL에서 제공하는 힌트명령어다. 성능평가는 Prefetch가 구현된 MySQL InnoDB 엔진과 기존의 엔진에 대하여 실험을 수행하였다.

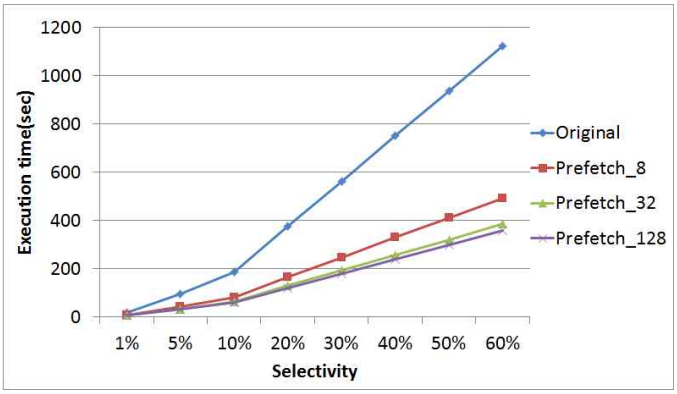

그림 2 기존의 InnoDB 엔진과 Prefetch가 구현된 InnoDB 성능 비교 그래프

4.2 실험 결과

본 논문에서는 *min*값과 *MAX*값 변경을 통해 쿼리 선택도에 변화를 주며 실험하였다. 그림 2는 기존의 InnoDB엔진과 Prefetch가 구현된 InnoDB엔진에서 쿼리를 수행한 실행 시간을 나타낸 그림이다. 그래프의 x축은 logical selectivity를 나타내고 y축은 쿼리 수행 시간을 나타낸다. Prefetch가 구현된 InnoDB엔진의 secondary index 스캔 성능은 기존의 InnoDB엔진의 secondary index 스캔의 성능보다 Prefetch 단위가 8일 때, 약 2.3배, 32 이상일 때, 약 3~3.1배 향상되었다. Prefetch 단위를 증가시키면 저장장치의 내부 병렬성도 증가하여 성능이 향상되었다. Prefetch 32일 때, 내부 병렬을 최대로 활용하여 Prefetch가 더 증가하더라도 성능 향상은 기존의 InnoDB 엔진 대비 약 3배로 수렴한다.

표 1는 쿼리를 수행하는 동안의 평균 CPU 사용률을 정리한 표이다. 기존의 InnoDB 엔진의 경우, 쿼리를 수행하는 동안 총 CPU 사용률은 약 25%이고, 그 중 85%이상인, 약 22%는 I/O wait를 하고 있다. 반면에 Prefetch가 구현된 InnoDB 엔진의 경우, Prefetch 단위가 128인 경우, I/O wait의 비율은 약 1%로 전체 CPU사용률의 약 5.5%이다. 실험을 통해, Prefetch가 구현된 InnoDB엔진을 사용할 경우, I/O wait 비율이 급격하게 감소한 것을 확인하였다.

본 논문에서 구현한 Prefetch는 secondary 인덱스 스캔의 랜덤 접근을 정렬을 통해 부분적으로 순차적 접근으로 변경하고, 정렬된 순서로 asynchronous I/O를 통해 필요한 데이터 페이지에 대한 읽기 요청을 미리 한다. 따라서 실제로 데이터가 필요한 시점에 해당 페이지에 대한 읽기 요청을 하였을 때, 이미 해당 데이터 페이지는 데이터베이스의 버퍼에 있기 때문에 I/O 지연시간을 줄일 수 있다.

5. 결론 및 향후 연구

본 논문에서는 MySQL InnoDB엔진의 secondary 인덱스 스캔을 위한 Prefetch 기능을 구현하였다. Prefetch기능으로 저장장치의 내부 병렬성을 활용하여 I/O 지연시간을 감소시켰다. Prefetch 단위가 32 이상일 경우, 최소 3배 이상의 성능이 향상되는 것을 실험을 통해 확인하였다.

하지만 성능 향상이 기대와는 달리 3~4배 사이로 수렴하는 경향이 관찰되었으므로 이를 개선하기 위한 연구를 후속 연구로 진행 할 예정이다.

사사

1) 이 논문은 2016년도 정부재원(미래창조과학부 여대학(원)생 공학연구팀제 지원사업)으로 미래창조과학부, 한국연구재단과 한국여성과학기술인지원센터의 지원을 받아 연구되었습니다.

2) 이 논문은 2016년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (R0126-16-1108, 비휘발성 메모리 기반 개방형 고성능 DBMS 개발)

3) 이 논문은 2015년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (10041244, 스마트TV 2.0 소프트웨어 플랫폼)

참고문헌

[1] Micheloni, Rino, Alessia Marelli, and Kam Eshghi. Inside solid state drives (SSDs). Vol. 37. Springer Science & Business Media, 2012.

[2] Da-som Hwang, Woon-hak Kang, Gihwan Oh, and Sang-won Lee. “Flash-aware index scan in PostgreSQL.“ In Data Engineering Workshops (ICDEW), 2015 31st IEEE International Conference, pp. 161-166. IEEE, 2015.

[3] MySQL-5.6.24 source code, http://dev.mysql.com/downloads/mysql/