

플래시 SSD를 활용한 비동기 복수 I/O 인덱스 스캔¹⁾²⁾

박지영, 강운학, 이상원
성균관대학교 전자전기컴퓨터공학과
e-mail:bjy8027@skku.edu

Asynchronous plural I/O index scan using flash SSD

Ji-Young Park, Woon-Hak Kang, Sang-Won Lee
Department of electrical and Computer Engineering, SungKyunKwan University

요 약

인덱스는 데이터 검색을 빠르게 하기 위하여 사용되며, 많은 데이터를 저장하는 대용량 데이터베이스 시스템은 B+-tree 인덱스를 주로 사용한다. B-tree 인덱스를 사용하여 범위 검색을 수행하는 경우 레코드 각각에 대하여 I/O를 요청함으로써 프로세스가 자주 대기(waiting) 상태가 되어 많은 오버헤드가 발생하였다. 이러한 문제를 해결하고자 본 논문에서 비동기 복수 I/O 인덱스 스캔방법을 제안한다. 비동기 복수 I/O 인덱스 스캔이 최고 6.5배 빠른 성능을 보였다.

1. 서론

대용량 데이터베이스시스템에서 데이터를 찾는데 걸리는 시간의 대부분이 I/O시간이다. 인덱스는 이러한 데이터 검색을 빠르게 하기 위하여 사용되며, 많은 데이터를 저장하는 대용량 데이터베이스시스템은 B+-tree 인덱스를 주로 사용한다.

B+-tree는 균형 트리로서 단말 노드(leaf node)에는 키 값과 그 키 값에 해당하는 레코드 아이디를 가리키고 있는 포인터 값이 저장되어 있는 형태이다. 단일 검색에서 값을 찾는데 최소한의 I/O를 통해 빠른 속도로 결과를 반환 하지만 범위 검색의 경우는 다르다. 예를 들어 Postgresql의 범위 검색을 살펴보면 찾고자 하는 키 값의 범위를 알고 있으면서도 하나의 레코드 당 한 번의 읽기를 요청하는 방법을 사용하여 느린 성능을 보이고 있다. 프로세스는 읽기를 수행 할 때마다 시스템 호출을 일으키는데 이때 프로세스는 디스크에서 데이터를 읽어오는 동안 대기(waite)상태에 빠지게 되어 오버헤드가 발생하게 된다. 데이터를 빠르게 읽어 오고자 인덱스를 사용하지만 이러한 오버헤드로 검색 성능이 떨어지는 것을 볼 수 있다. 따라서 본 논문에서는 한번의 I/O를 요청하는 것이 아

니라 범위에 해당하는 레코드를 한 번에 요청함으로써 프로세스가 대기상태에 빠지게 되어 발생하는 오버헤드를 줄이는 방법을 제안한다. 하지만 많은 데이터를 한 번에 읽어오는데 시간 또한 오래 소요 되는 문제점이 발생한다. 이것은 디스크에 반드시 연속해서 저장되어 있다고 보장할 수 없기 때문에 임의 읽기(random read)가 발생하기 때문이다. 하드 디스크에서 임의 읽기를 수행할 경우 디스크 암과 헤더를 움직이는데 순차 읽기보다 더 많이 움직이게 되어 시간이 많이 소요된다. 대기 상태에 빠져 있는 오버헤드를 줄인다고 해서 검색시간을 크게 단축시키는 효과는 누릴 수 없게 된다. 그리하여 임의 읽기에 효과적인 플래시 SSD(Solid State Drive)를 기반에서의 범위 검색 시 효과적인 비동기 복수 I/O방법을 제안한다.

플래시 SSD의 경우 기계적인 요소가 없어 접근 속도가 매우 빠른 장점을 가지고 있으며, 내부적으로 병렬화를 포함하고 있어 복수 I/O를 요청 시 채널 수 만큼 한 번에 처리 할 수 있는 장점을 가질 수 있다. 이러한 장점을 이용한 B+-tree인덱스 최적화 기법이 제안된 적이 있었다.[2] 이 논문은 본 논문에서 제안하는 방법과 달리 내부 노드에서의 복수 I/O를 이용한 기법이다. 우리는 단말 노드를 통해 읽기가 주가 되는 OLAP(OnLine Analytical Processing)환경에서 효율적인 효과를 볼 수 있을 것으로 예상한다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 관련 연구로 Postgresql의 B+-tree에 대하여 설명하고, 내부 노드에 대한 B+-tree 인덱스 최적화기법에 대하여 설명한다. 3장에서는 본 논문에서 제안하는 비동기 복수 I/O 인덱스 스캔에 대하여 설명한다. 4장에서는 구현과 실험, 실

- 1) 본 연구는 지식경제부 및 한국산업기술평가관리원의 산업융합원천기술개발사업(정보통신)의 일환으로 수행하였음. [10041244, 스마트TV 2.0 소프트웨어 플랫폼]
- 2) “본 연구는 지식경제부 및 정보통신산업진흥원의 대학 IT연구센터 육성지원사업의 연구결과로 수행되었음” (NIPA-2012-(H0301-12-3001))

험 결과를 설명하고 5장에서 결론 및 향후 연구로 논문을 맺는다.

2. 관련 연구

이 장에서는 Postgresql에서 사용하는 B+-tree 구조에 대하여 설명하고, 플래시 SSD의 내부 병렬화를 사용하여 B+-tree 인덱스 최적화한 논문을 설명한다.

먼저 Postgresql에서 적용하는 B+-tree 인덱스는 B-link-tree를 변형하여 사용한다. B-link-tree 구조를 알아본 후, Postgresql의 B+-tree를 알아본다.

2.1 B-link-tree 구조[1]

B-link-tree는 루트(root)에서 단말 노드(leaf node)까지 모든 경로의 길이가 같은 균형 트리이다. 루트를 제외한 모든 노드는 최소한 $k+1$ 개의 자식을 가지고 있어야 하며, 최대 $2k+1$ 개를 가질 수 있다. 그림 1과 같이 B-link-tree 노드는 키 값과 포인터를 포함하며, 노드 안 키 값은 오름차순으로 정렬되어 있다.

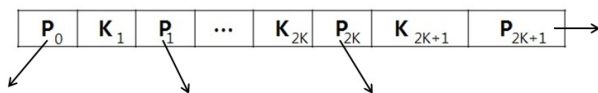


그림 1 B-link-tree 노드

모든 노드의 마지막 검색키 값은 큰 키 값(high key)로 이 노드에서 나타낼 수 있는 가장 큰 키 값을 나타낸다. 그리고 마지막 포인터는 같은 높이(level)의 오른쪽 형제 노드를 가리키고 있다. 이 포인터를 링크 포인터(link pointer)라 부르며, 검색키 값이 큰 키를 넘는 경우에는 이 링크 포인터를 통해 빠르게 옆 노드로 이동할 수 있는 장점을 가지고 있다. 또한 순차 검색에도 효율적이다.

단말 노드의 가장 첫 번째 포인터는 단말 노드라는 표시를 두고, P_i 는 K_i 에 해당하는 파일 레코드를 가리키고 있다.

비단말 노드(non-leaf node)의 포인터는 단말 노드와는 달리 트리의 자식 노드를 가리키고 있으며, P_i 는 $K_i < v \leq K_{i+1}$ 사이의 값을 포함하는 자식 노드를 가리킨다.

2.2 Postgresql의 B+-tree

Postgresql에서는 앞서 설명한 B-link-tree를 조금 변형하여 사용하고 있다.

첫 째, 단말 노드에서는 키 값이 유일해야만 하지만 Postgresql에서는 중복키를 허용한다. 따라서 부모 페이지에서 포인터를 따라 내려간 서브트리(subtree)는 $K_i \leq v \leq K_{i+1}$ 범위의 키 값을 가지고 있다.

둘 째, B-link-tree에서는 읽기 잠금을 요청하지 않고, 각 프로세스들은 공유되지 않는 메모리에 페이지를 복사

하여 읽기를 각각 수행하였다. 하지만 Postgresql은 메모리 버퍼가 공유되기 때문에 수정이 아닌 데이터를 읽을 때도 읽기 잠금을 수행한다.

읽기 잠금은 공유락(SharedLock)으로, 읽는 중에 다른 프로세스가 수정하는 것을 막기 위해서이다. 이것으로 정확한 수행을 할 수 있다.

다른 프로세스에 의해 올라와있던 페이지를 수정하고자 할 때 메모리 버퍼를 공유하고 있으므로 다시 페이지를 읽어오지 않고, 그 페이지에 쓰기 잠금을 하고 데이터를 수정하여 사용할 수 있는 장점이 있다. 쓰기 잠금은 배타적락(ExclusiveLock)으로 한 프로세스만이 잠금을 수행할 수 있으며, 배타적락을 잡고 있으면 다른 프로세스들은 잡고 있는 프로세스가 잠금을 풀기 전까지는 접근을 할 수 없다. 쓰기 잠금과는 달리 읽기 잠금은 잠금을 잡고 있는 프로세스 외에도 다른 프로세스가 접근하는 것은 가능하다.

2.3 다중 경로 탐색 알고리즘[2]

플래시 SSD(Solid State Drive)의 내부 병렬화를 통하여 B+-tree인덱스를 최적화하는 방법이 제안된 적이 있었다. 이 논문에서는 다중 경로 탐색(Multi Path Search) 알고리즘을 제안한다. 이것은 각 레벨에 존재하는 필요한 여러 노드들을 한 번의 병렬동기I/O(Parallel Synchronous I/O)를 통하여 검색과 갱신을 효율적으로 하는 방법을 제안하였다. 이것으로 단말 노드까지 빠르게 검색할 수 있다. 여기서 말하는 병렬동기I/O는 I/O요청을 배열로 하는 것 이외에는 기존의 동기I/O(syncI/O)와 같다. 본 논문에서는 이 논문을 참조하여 비동기 복수 I/O 인덱스 스캔방법을 제안한다.

3. 비동기 복수 I/O 인덱스 스캔

2장에서 언급하였듯이, B+-tree 인덱스의 노드는 키 값과 키 값의 포인터를 포함하고 있으며 단말 노드의 포인터는 해당 키 값과 관련된 파일 레코드의 위치를 나타내고 있다. 인덱스 스캔을 수행하는 경우, 검색하고자하는 키 값을 루트에서 해당 키 값이 존재하는 단말 노드까지 찾아 내려온 뒤 해당 키 값의 포인터가 가리키는 곳의 페이지를 읽어 오라는 명령을 시스템에 호출하게 된다. 단일 검색의 경우에는 문제가 없지만, 범위 검색의 경우 처음 시작부터 끝까지 알고 있음에도 불구하고 하나의 레코드에 한번의 I/O를 요청하는 방법을 사용한다. 이것은 프로세스의 상태가 대기 상태에 레코드 수만큼 빠져있는 오버헤드가 발생하게 된다. 이러한 문제점을 해결하고자 본 논문에서는 비동기 복수 I/O 인덱스 스캔을 제안한다.

비동기 복수 I/O 인덱스 스캔은 범위 검색 시 해당 범위에 포함하고 있는 여러 파일 레코드의 위치를 저장하여 한번의 I/O를 통해 읽어 들이는 것으로, 프로세스가 대기 상태에 빠져 있는 오버헤드를 크게 줄일 수 있다. 또한 많

은 데이터를 하드 디스크에서 찾아 읽어오는데 걸리는 시간을 단축하고자 플래시 SSD를 사용하여 우리가 제안하는 비동기 복수 I/O 인덱스 스캔으로 검색 시간을 크게 단축시키는 효과를 누릴 수 있었다.

4. 구현과 실험

4.1 구현

B+-tree 인덱스 범위 검색 시, 루트(root)에서 키 값이 존재하는 단말 노드까지 찾아간다. 단말 노드에서 키 값이 존재하는 정확한 위치를 찾는다. 그 위치에서부터 찾는 범위에 만족하는지 검사하여 만족한다면 디스크 블록 번호와 인덱스 오프셋 번호(offset number)를 저장한다. 해당 범위를 벗어나거나 단말 노드의 끝에 도달하면 검사를 멈추고 저장한 디스크 블록 번호를 보고 하나씩 읽어오는 과정을 거친다. 여기서 50개의 복수 I/O를 요청하기 위하여 50개의 디스크 블록 번호만을 따로 다시 저장하고, 50개의 버퍼를 저장할 공간을 마련한다.

50개의 블록이 메모리에 존재하는지 검사를 한 뒤 존재하는 경우에는 그대로 그 버퍼 아이디를 저장하고, 존재하지 않는 경우에는 존재하지 않는 수를 세고 비동기 I/O 데이터 구조체인 'iocb'에 내용을 저장한다. 이후 존재하지 않는 수만큼 'io_submit()'이라는 리눅스 비동기 I/O를 통해 읽어 들인다.

50개의 모든 블록을 메모리에 읽어온 뒤, 하나의 레코드에서 필요한 데이터를 저장하는 작업을 수행한다. 50개의 블록을 모두 올려놓았으므로 50개의 레코드에 대해서는 디스크 I/O를 유발하지 않는다. 50개의 레코드를 다 처리한 후에는 다음 50개의 블록을 읽어오는 작업을 반복 수행 한다.

4.2 실험 환경

비동기 복수 I/O의 성능을 측정하기 위한 실험 환경은 <표2>과 같다. PostgreSQL-8.4.6 버전에서 실험하였으며, 비동기 복수 I/O를 하기 위해서 direct I/O를 수행하여야 한다. 따라서 기존 소스 또한 direct I/O로 수정하여 인덱스 스캔 성능을 측정 하였다.

항목	설명
CPU	AMD Opteron(tm) Processor 6128, 2GHz [8 core * 2]
RAM	DDR3 40GB
SSD	Samsung 470 Series 128GB
데이터베이스 크기	1.12GB
데이터베이스 메모리 크기	320MB

표 2 성능평가 환경

4.3 결과

실험 결과 아래 그림 2와 같은 결과를 보였다.

선택도가 20%에서는 기존 소스에 direct I/O를 수행하는 것보다 비동기 복수 I/O를 수행하는 것이 6.5배 빠른 것을 볼 수 있었다. 나머지 선택도에서도 최소 5.7배 더 빠른 것을 확인 할 수 있었다.

이것은 플래시 SSD에서의 내부 병렬화 특징으로 복수 I/O요청을 병렬적으로 수행되기 때문에 하나씩 요청하는 것에 비해 빠른 성능을 보일 수 있었다.

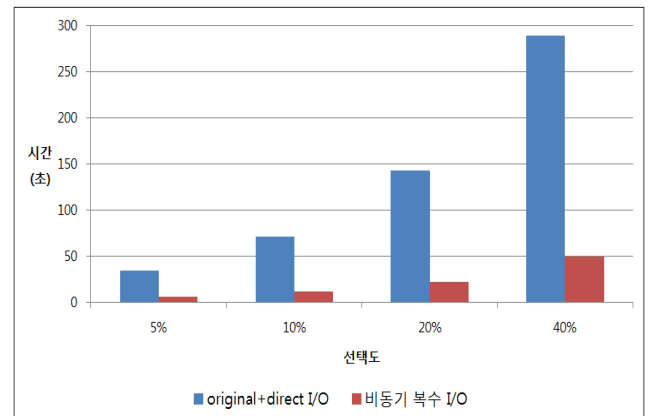


그림 2 실험 결과

5. 결론 및 향후 연구

본 논문에서는 B-link-tree에 대하여 살펴보고, 플래시 SSD에서의 범위 검색 시 효율적으로 동작하도록 하는 방법을 제안 하였다.

인덱스에서 순차적으로 저장되어 있다고 디스크에서도 연속해서 저장되어 있지 않기 때문에 임의 읽기에 대하여 빠른 성능을 보이는 플래시 SSD를 사용함으로써 검색 시간을 크게 단축시킬 수 있었으며, 또한 플래시 SSD의 내부 병렬화를 가지고 있어 복수 I/O를 요청함으로써 Original에 비해 더 빠른 성능을 보일 수 있었다.

향후 연구로 더 다양한 복수 I/O를 요청하여 측정해볼 것입니다.

참고문헌

- [1] Philip L. Lehman and S. Bing Yao, "Efficient Locking for Concurrent Operations on B-Tree", Proc. ACM, 1981
- [2] Hongchan R, Sanghyun P, Sungho K, Mincheol S and SangWon L, "B+-tree Index Optimization by Exploiting Internal Parallelism of Flash-based Solid State Drives", Proc. VLDB, 2012
- [3] Christoffer Hall and Philippe Bonneet, "Getting Priorities Straight : Improving Linux Support for Database I/O", Proc. VLDB, 2005
- [4] Lawrence To, "Asynchronous Reads", Oracle Technical Report