ISSN 2383-630X(Print) / ISSN 2383-6296(Online)
Journal of KIISE, Vol. 44, No. 2, pp. 208-212, 2017. 2
https://doi.org/10.5626/JOK.2017.44.2.208

MySQL InnoDB엔진의 Secondary Index Scan을 위한 Prefetch 기능 구현

(Implementation of a Prefetch method for Secondary Index Scan in MySQL InnoDB Engine)

황 다 솜 [†] (Dasom Hwang)
이 상 원 [††] (Sang-Won Lee)

요 약 플래시 SSD는 기존 하드디스크에 비해 높은 에너지 효율성, 외부 충격에 강한 내구성, 높은 입출력 처리량 등 여러 장점을 지니고 있다. 따라서 3D-NAND 및 V-NAND 등 단위 용량 당 비용을 획기적으로 개선하는 최신 기술의 등장과 맞물려서, 플래시 SSD가 많은 영역에서 하드디스크를 급격하게 대체하고 있다. 하지만, 주로 하드디스크를 가정하고 개발된 기존 데이터베이스 엔진은 플래시 SSD의 특성 (예를 들어, 내부 병렬성)을 제대로 활용하지 못하고 있다. 본 논문에서는, 더 빠른 질의 처리를 위해 플래시 SSD에 내재한 내부 병렬성을 활용하는 방법으로, MySQL InnoDB엔진에서 보조 인덱스(Secondary Index)를 이용한 스캔을 위해 비 동기적 입출력을 활용한 Prefetch 기능을 구현하였다. Prefetching을 사용한 스캔 기법은, 기존 InnoDB엔진의 보조 인덱스 스캔과 비교해서, 데이터 페이지 크기가 16KB일 경우, 약 3배 이상, 데이터 페이지 크기가 4KB일 경우, 약 4.2배 이상 성능 향상을 보인다.
키워드: 플래시 SSD, 인덱스 스캔, Prefetch, MySQL InnoDB엔진

Abstract Flash SSDs have many advantages over the existing hard disks such as energy efficiency, shock resistance, and high I/O throughput. For these reasons, in combination with the emergence of innovative technologies such as 3D-NAND and V-NAND for cheaper cost-per-byte, flash SSDs have been rapidly replacing hard disks in many areas. However, the existing database engines, which have been developed mainly assuming hard disks as the storage, could not fully exploit the characteristics of flash SSDs (e.g. internal parallelism). In this paper, in order to utilize the internal parallelism intrinsic to modern flash SSDs for faster query processing, we implemented a prefetching method using asynchronous input/output as a new functionality for secondary index scans in MySQL InnoDB engine. Compared to the original InnoDB engine, the proposed prefetching-based scan scheme shows three-fold higher performance in the case of 16KB-page sizes, and about 4.2-fold higher performance in the case of 4KB-page sizes.
Keywords: Flash SSD, index scan, prefetch, MySQL innnoDB engine

· 이 논문은 2016년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (R0126-16-1108, 비휘발성 메모리 기반 개방형 고성능 DBMS 개발)
· 이 논문은 2016년도 정부재원(미래창조과학부 여대학(원)생 공학연구팀제 지원사업)으로 미래창조과학부, 한국연구재단과 한국여성과학기술인지원센터의 지원을 받아 연구되었습니다.
· 이 논문은 2016 한국컴퓨터종합학술대회에서 '[여성연구자상 지원논문]MySQL InnoDB엔진의 Secondary Index Scan 성능 향상을 위한 Prefetch 기능 구현'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 성균관대학교 전자전기컴퓨터공학과
rhcqnssp32@skku.edu
†† 종신회원 : 성균관대학교 소프트웨어대학 교수
(Sungkyunkwan Univ.)
swlee@skku.edu
(Corresponding author임)

논문접수 : 2016년 8월 17일
(Received 17 August 2016)
논문수정 : 2016년 10월 17일
(Revised 17 October 2016)
심사완료 : 2016년 11월 4일
(Accepted 4 November 2016)

정보과학회논문지 제44권 제2호(2017. 2)

1. 서 론

플래시 SSD(Flash-based Solid-State Drive)는 높은 에너지 효율성, 외부 충격에 강한 내구성, 높은 입출력 처리량 등 여러 가지 장점을 가지고 있다. 또한 3D-Nand 및 V-NAND 등 단위 용량 당 비용을 획기적으로 개선하는 최신 기술의 등장과 맞물려서 플래시 SSD는 모바일 기기뿐만 아니라 데스크톱, 서버 등 여러 분야에서 하드디스크를 대체하는 저장장치로 자리매김하고 있다[1].

기존의 데이터베이스 시스템(Database Systems, DBMS)은 하드디스크를 저장장치로 하는 컴퓨팅 환경에서 개발되었기 때문에 하드디스크의 특성을 고려한 설계요소가 많이 남아있다. 하지만, 플래시 SSD는 하드디스크와 다른 특징을 가지고 있기 때문에 플래시 SSD의 특성을 이해하고 이러한 특성을 충분히 활용할 수 있는 새로운 DBMS 설계가 필요하다.

예를 들면, MySQL InnoDB엔진의 데이터 페이지의 기본 크기가 16KB로 하드디스크의 입출력 지연시간을 줄이기 위해 비교적 큰 데이터 페이지 크기를 채택하였지만 플래시 SSD의 최소 입출력 단위인, 4KB로 데이터 페이지 크기를 설정하는 것이 플래시 SSD 환경에서 DBMS 성능에 더 유리하다[2]. 또한, 플래시 SSD의 설계상의 특성으로 동시에 여러 입출력을 병렬적으로 처리할 수 있고 이를 Flash SSD의 내부 병렬성이라고 한다. 하지만 대부분의 DBMS의 인덱스 스캔은 동기적 입출력(Synchronous I/O)을 사용하기 때문에 단일 쿼리만 수행 될 경우, 한 페이지를 읽기 위해 한 칩만 활성되기 때문에 플래시 SSD의 내부 병렬성을 충분히 활용하지 못한다[3].

본 논문에서는 오픈소스 데이터베이스 시스템인 MySQL InnoDB엔진에서 보조 인덱스(Secondary Index)를 이용한 스캔을 위해 비 동기적 입출력을 활용한 Prefetch 기능을 구현하였다. Prefetch 기능 구현으로 보조 인덱스를 이용한 읽기의 성능향상을 기대할 수 있다. 실험을 통해 확인한 결과, Prefetch 단위가 32이상일 때, 데이터 페이지 크기가 16KB일 경우, 보조 인덱스를 이용한 읽기 성능은 기존의 InnoDB엔진의 성능보다 약 3배 향상되었다. 데이터 페이지 크기가 4KB일 경우, 기존의 InnoDB엔진의 보조 인덱스를 이용한 읽기 성능보다 4.2배 이상 향상되었다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구를 다루고 3장에서는 InnoDB엔진의 읽기 메커니즘과 InnoDB엔진에 구현된 버퍼 풀 Prefetch 기법을 소개한다. 4장에서는 본 논문에서 구현하는 보조 인덱스 스캔을 위한 Prefetch 기법 설명한다. 5장에서는 구현한 Prefetch 기법의 성능 평가 실험 및 실험 결과에 대해 분석하고 마지막으로, 6장에서 논문의 결론을 맺는다.

2. 관련연구

본 장에서는 플래시 SSD 저장장치 환경에서의 쿼리 프로세싱과 관련된 연구를 소개한다.

2.1 플래시 SSD와 정렬 인덱스 스캔

[3]은 플래시 SSD 저장장치 환경에서 상용 DBMS의 두 가지 주요 접근 방식인, 전체 테이블 스캔(Full table scan)과 인덱스 스캔의 성능을 비교한다. 또한 비 군집 인덱스에 대한 인덱스 스캔의 성능을 향상시키기 위해 정렬 인덱스 스캔을 활용하는 것을 제안한다. 정렬 인덱스 스캔은 인덱스 엔트리에 저장되어 있는 레코드 식별자를 우선 정렬한 후, 정렬된 순서로 데이터 페이지에 접근하는 방식으로 동작한다. 따라서 비 군집 인덱스에 대한 인덱스 스캔의 성능 저하 원인인 중 하나인, 제한된 버퍼 크기 때문에 같은 데이터 페이지를 반복적으로 읽어야하는 상황을 피할 수 있다. 따라서 정렬 인덱스를 사용하면 버퍼 크기와 상관없이 일정한 성능을 보장할 수 있다. 단일 사용자의 경우, 선택도가 최대 70%인 경우에도, 정렬 인덱스 스캔이 전체 테이블 읽기보다 성능이 뛰어남을 보였다. 또한, 동시 사용자의 수가 늘어날수록, SSD의 내부 병렬성 활용효과가 증가하면서 정렬 인덱스 스캔의 성능이 전체 테이블 스캔의 성능을 능가하는 선택도도 함께 증가하였다.

2.2 PostgreSQL에서의 인덱스 스캔 최적화 기법

[4]는 오픈소스 관계형 데이터베이스 시스템인 PostgreSQL에 정렬 인덱스 스캔과 비동기 입출력(Asynchronous I/O, AIO)을 활용한 병렬 동기적 입출력(Parallel synchronous I/O, P-sync I/O)[5]을 구현하여 비 군집 인덱스에 대한 인덱스 스캔의 성능을 향상하였다. 비 군집 인덱스를 접근할 때 발생하는 데이터에 대한 무작위 접근을 정렬 인덱스 스캔을 통해 순차적 접근 패턴으로 변환하고 P-sync I/O를 통해 한 번에 여러 입출력을 처리한다. 하지만, P-sync I/O는 이름에서도 유추 가능하듯이, 다수의 입출력을 동기적으로 처리하기 때문에 요청한 모든 입출력 처리가 끝날 때 까지 모든 작업들은 대기하여야 한다. 정렬 인덱스 스캔을 통해 필요한 데이터 페이지에 대한 접근은 최대 1번만 발생하고 P-sync I/O로 기존의 입출력 처리 방식에 보다 플래시 SSD의 내부 병렬성을 효과적으로 활용하여 저장장치의 대역폭을 증가시켰다. [4]는 비 군집 인덱스에 대한 스캔 성능을 기존의 PostgreSQL 대비 최대 약 3.8배 향상시켰다.

3. MySQL InnoDB엔진의 읽기 메커니즘

본 장에서는 MySQL InnoDB엔진에서의 기본 인덱

스와 보조 인덱스에 대한 읽기 동작 방식과 InnoDB엔진의 버퍼 풀 Prefetch 기법을 다룬다.

3.1 InnoDB엔진의 읽기 메커니즘[6]

Algorithm 1은 MySQL InnoDB엔진의 인덱스 스캔 알고리즘의 의사 코드이다. MySQL InnoDB엔진은 데이터를 기본 키에 대한 B-tree형태로 저장을 하여 관리하고, 테이블에 접근할 때, 기본적으로 기본 키를 이용한 인덱스 스캔을 사용한다. 쿼리에 따라 보조 인덱스를 이용한 인덱스 스캔을 할 경우에는 보조 인덱스를 탐색하여 보조 인덱스 레코드(보조 인덱스 키와 기본 키의 쌍)를 획득한 후, 해당 레코드에서 기본 키 값을 추출한다. 추출한 값으로 기본 인덱스를 탐색하여 최종적으로 필요한 데이터 레코드를 획득한다.

Algorithm 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 the primary index then
9:          Extract a primary key pk from r
10:         Open the cursor pcur at the primary index
11:         while !pcur points to a leaf page
12:             Move pcur at a proper position
                in a page
13:             Read a child page
14:         Read a record pr mathcing pk from the
            leaf page
15          return pr
16      else
17          return r
18  else
19      return r
```

기본 인덱스를 이용하여 데이터에 접근할 경우, 접근 패턴이 순차적이기 때문에 짧은 지연시간을 기대할 수 있다. 하지만 보조 인덱스를 이용할 경우, 보조 인덱스는 비 군집 인덱스이므로 보조 인덱스에서 레코드에서 추출한 기본 키 값들이 무작위적이고, 그러므로 실제 데이터 페이지를 읽기 위한 접근 패턴도 무작위적이다. 따라서 보조 인덱스를 이용하여 데이터에 접근할 때에는 짧은 지연시간을 기대하기 어렵다.

3.2 InnoDB엔진의 Read-ahead 기법[7]

Read-ahead기법은 InnoDB엔진이 입출력 처리 성능을 향상시키기 위해 구현된 버퍼 풀 Prefetch 기법으로 간단한 통계에 기반을 둔 예측을 통해 수행된다. 예측을 통해, 곧 접근 될 수도 있다고 판단된 데이터 페이지를 비동기 입출력을 이용하여 버퍼 풀에 미리 읽어온다. Read-ahead 기법은 2가지 기법, Linear Read-ahead 기법과 Random Read-ahead기법이 있다.

Linear Read-ahead기법은 DBMS의 접근 패턴 분석을 통해 다음 데이터 영역에 대한 접근을 고려한 기법이다. 버퍼 풀에 같은 Extent(64 data pages)에 속하는 데이터 페이지들이 “순차적”으로 접근 되었고, 접근된 페이지의 개수가 임계값이상 일 때, 다음 Extent 전체를 비동기 입출력을 통해 Read-ahead를 요청한다. 임계값은 56으로 설정되어있고, 서버 설정 변수, “innodb_read_ahead_threshold”로 조절 가능하다. 기본적으로 Linear Read-ahead기법은 활성 되어있다.

이에 반해, Random Read-ahead 기법은 기본적으로 비활성되어 있다. 서버 설정 변수, “innodb_random_ read_ahead”를 통해 활성화시킬 수 있다. Random Read-ahead는 데이터 페이지들의 접근된 순서는 고려하지 않고 버퍼 풀에 동일한 Extent에 속하는 13개의 연속된 페이지가 있다면, 해당 Extent의 나머지 데이터 페이지에 대하여 비동기 입출력을 통해 Read-ahead를 요청한다.

4. 보조 인덱스 스캔을 위한 Prefetch 기법

본 장에서는 본 논문에서 구현하는 기법인, 보조 인덱스 스캔을 위한 구현한 Prefetch 기능에 대해 설명한다. 앞서 설명하였듯이, MySQL InnoDB엔진에서 보조 인덱스를 이용한 인덱스 스캔을 할 경우, 이 때 발생하는 랜덤 읽기로 인해 시스템의 응답시간이 매우 길어진다. 이를 개선하고자, Libaio 라이브러리를 활용한 비동기 입출력을 통해 Prefetch를 구현하였다. Algorithm 2는 본 논문에서 구현한 Prefetch를 간략히 나타낸 의사 코드이다. Prefetch 구현은 Algorithm 1의 8과 9사이에 추가 되었고 플래그를 통해 활성화 또는 비활성화 시킬 수 있다. 앞 장에서 언급한 InnoDB엔진의 Read-ahead와는 다르게, 보조 인덱스를 우선 탐색하여 기본 키 값을 우선 추출한다. 수집된 기본 키 값으로 기본 인덱스의 level-1까지 탐색 한 후, level-1의 노드에서 다음으로 접근 할 자식 노드의 데이터 페이지 정보를 수집한다. 수집된 데이터 페이지 정보를 정렬한 후, 정렬된 데이터 페이지 정보를 이용하여 Prefetch를 수행한다. Prefetch 단위에 따라 수집할 데이터 페이지 개수가 달라진다. 예를 들어, Prefetch 단위가 8인 경우, 데이터 페이지도 8개 까지 수집한 후 Prefetch를 요청한다. 본 기법은 곧 접근될 데이터 페이지를 미리 읽기 때문에 InnoDB엔진의 Read-ahead 기법과는 다르게 버퍼 풀을 오염시킬 가능성이 없다. 또한, 다수의 읽기를 동시에 요청하기 때문에 SSD의 내부 병렬성을 활용하여 읽기

```
Algorithm 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 the primary index
5:  while !end of rec_list
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
```

대역폭을 향상시킬 수 있다.

본 논문에서 구현한 Prefetch 기법은 기존의 InnoDB 엔진에서 페이지 읽기가 발생하기 전에 추가되어 앞으로 필요한 데이터 페이지에 대한 미리 읽기 기능을 수행하는 것이다. 다수의 입출력 요청을 내린다는 점은 [4]와 같으나, [4]처럼 해당 입출력 요청들이 끝날 때까지 기다리지 않는다. 입출력 요청을 한 후, 처리 결과는 전적으로 시스템의 비동기 입출력 처리과정에 맡기고 기존의 InnoDB엔진과 같이, 현재 데이터베이스 시스템이 읽어야하는 레코드가 담긴 데이터 페이지에 대해서 동기적 읽기를 요청한다. 이 때, SSD의 빠른 입출력 성능 덕분에 대부분의 데이터 페이지는 버퍼 풀에서 찾을 수 있기 때문에 추가적인 입출력이 발생하지 않는다.

5. 성능 평가

본 장에서는 본 논문에서 구현한 보조 인덱스를 위한 Prefetch 기법의 성능을 평가하기 위해 수행한 실험을 소개하고 실험 결과를 분석한다.

5.1 실험 환경

표 1은 본 논문에서 실시한 실험 환경을 정리한 표이다. 운영체제는 Ubuntu 14.04, 프로세서는 Inter Core i5 (Ram 8GB), 저장장치는 Samsung SSD 850Pro(256GB)을 사용하였다. DBMS로 MySQL 5.6 버전을 사용하였고, 해당 버전에 Prefetch 기능을 구현하였다. 데이터베이스는 TPC-H 벤치마크를 사용하였고, 그 중 Orders 테이블에 대하여 Orders 테이블의 보조 인덱스를 이용한 범위 쿼리를 수행하였다. 그림 1은 본 논문에서 실시한 쿼리의 예를 나타낸 것이다. “Column_a”의 값이 “min”과 “MAX” 사이의 값을 가지는 레코드를 요청하는 쿼리이다. 데이터 페이지 크기에 따른 성능 비교를 위해, InnooDB의 기본 데이터 페이지 크기은 16KB와 플래시 SSD의 최소 입출력 단위인 4KB로 실험 데이터를 생성하였다. 또한, Prefetch 단위에 따른 성능 비교를 위해 Prefetch 단위를 8, 16, 32, 64, 128, 256로 다양하게 설정하였다. 실험은 Prefetch가 구현된 MySQL InnoDB엔진과 기존의 InnoDB엔진에서 각각 수행하였다.

표 1 실험 환경

Table 1 Experiment Environment

Operating System	Ubuntu 14.04
Processor	Intel Core i5, 3.40GHz
RAM	8GB
Storage	Samsung 850Pro(256GB)
DBMS	MySQL InnoDB 5.6
TPC-H Scale Factor	10
Page size	16KB, 4KB
Prefetch Factor	8, 16, 32, 64, 128, 256

```
SELECT * FROM table
WHERE colum_a BETWEEN min AND MAX ;
```

그림 1 예시 범위 쿼리

Fig. 1 An Example of a Range Query

5.2 실험 결과

그림 2와 그림 3은 기존의 InnoDB엔진과 Prefetch기능이 구현된 InnoDB엔진의 쿼리 수행 시간을 비교한 그래프이다. 그림 2는 데이터 페이지 크기가 16KB인 경우의 실험 결과 그래프이고 그림 3은 데이터 페이지 크기가 4KB인 경우의 실험 결과그래프다. 그래프의 x축은 논리적 선택도를 나타내고, y축은 쿼리 수행시간을 나타낸다. 그림에서 Original은 기존의 InnoDB엔진의 성능을 나타내고, Prefetch_N의 경우, Prefetch 단위가 N인 경우를 나타낸다. 예를 들어, Prefetch_8은 Prefetch 단위가 8인 경우이다. 그래프의 데이터는 Prefetch 단위가 8, 32, 128인 결과로 제한하였다. 데이터 페이지 크기가 16KB일 때, Prefetch가 구현된 InnoDB엔진의 경우, 기존의 InnoDB엔진의 보조 인덱스 스캔 성능보다 Prefetch 단위가 8일 때, 약 2.3배, Prefetch 단위가 32일 때, 약 3배 향상되었다. Prefetch 단위를 증가시키면 병렬적으로 처리할 수 있는 입출력 양이 증가한다. 이에 따라 저장장치의 내부 병렬성을 더욱 효과적으로 활용할 수 있게 되어 DBMS의 성능 향상으로 이어진다. 하지만, SSD의 내부 병렬성과 입출력 대기열 크기에 한계가 있기 때문에, Prefetch 단위를 계속 증가하더라도 성능 향상은 기존의 InnoDB엔진 대비 약 3.1배로 수렴한다. 데이터 페이지 크기가 4KB일 때, Prefetch가 구

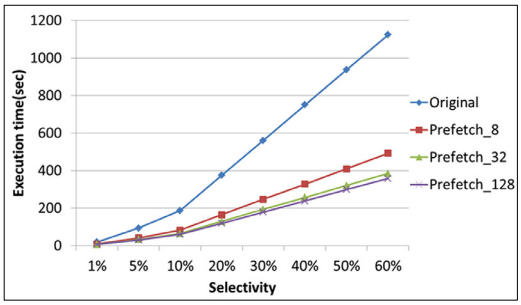


그림 2 MySQL InnoDB엔진 성능 비교 그래프(16KB)
Fig. 2 Performance Comparison: 16KB-page case

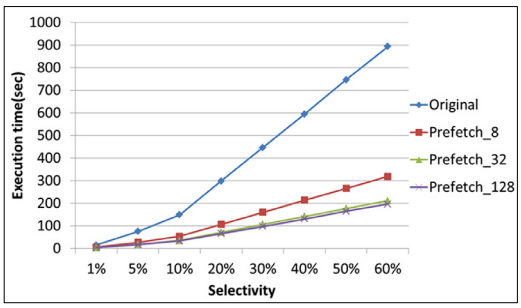


그림 3 MySQL InnoDB엔진 성능 비교 그래프(4KB)
Fig. 3 Performance Comparison: 4KB-page case

현된 InnoDB엔진의 경우, 기존의 InnoDB엔진의 보조 인덱스 스캔성능보다 Prefetch 단위가 8일 때, 약 2.8배, Prefetch 단위가 32일 때, 약 4.2배 이상 향상되었다. 16KB의 경우와 같은 이유로, Prefetch 단위를 증가시켜도 성능 향상 정도는 약 4.5배로 수렴한다.

6. 결 론

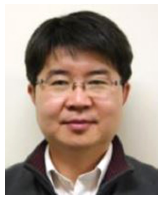
본 논문에서는 MySQL InnoDB엔진의 보조 인덱스 스캔의 성능 향상을 위해 Prefetch기능을 구현하였다. Prefetch 기능은 다수의 입출력을 동시에 요청하기 때문에 기존의 InnoDB엔진의 보조 인덱스 스캔 기법보다 플래시 SSD의 내부 병렬성을 효과적으로 활용한다. 따라서 저장장치의 입출력 대역폭을 증가시켜 MySQL InnoDB엔진의 성능을 향상 시켰다. Prefetch 단위가 32 이상일 때, 기존의 InnoDB엔진대비 약 3배 이상의 성능이 향상 되었다. 또한, 본 논문에서는 데이터 페이지 크기가 DBMS의 성능에 미치는 영향을 살펴보았다. 데이터 페이지의 크기가 플래시 SSD의 최소 입출력 단위인 4KB일 때, 기존의 InnoDB엔진 대비 약 4.2배 이상 성능이 향상되었다.

향후 연구로 Prefetch 기능 구현 최적화 및 성능 개선에 대한 연구를 계속 진행할 예정이다.

References

[1] Micheloni, Rino, Alessia Marelli, and Kam Eshghi, *Inside solid state drives (SSDs)*, 1st Ed., pp. 382, Springer Science & Business Media, 2012.
[2] Kang, Woon-Hak, Sang-Won Lee, Bongki Moon, Yang-Suk Kee, and Moonwook Oh, "Durable write cache in flash memory SSD for relational and NoSQL databases." *Proc. of the 2014 ACM SIGMOD international conference on Management of data*, pp. 529-540. ACM, 2014.
[3] Jae-Hyo Lee, Sang-Won Lee, Chanik Park, "Revisiting Sorted Index Scan with Flash SSDs," *EDB 2010*, Juju Korea, Aug. 2011.
[4] Hwang, Da-som, Woon-hak Kang, Gihwan Oh, and Sang-won Lee, "Flash-aware index scan in PostgreSQL," *Data Engineering Workshops (ICDEW), 2015 31st IEEE International Conference on*, pp. 161-166. IEEE, 2015.
[5] Roh, Hongchan, Sanghyun Park, Sungho Kim, Mincheol Shin, and Sang-Won Lee, "B+-tree index optimization by exploiting internal parallelism of flash-based solid state drives," *Proc. of the VLDB Endowment 5*, No. 4, pp. 286-297, 2011.
[6] MySQL Server ver. 5.6 Soruce code [Online] Available: http://dev.mysql.com/downloads/mysql
[7] MySQL Reference Manual(5.6) [Online] Available: https://dev.mysql.com/doc/refman/5.6/en/

황 다 솜
2013년 성균관대학교 컴퓨터공학과 학사 2015년 성균관대학교 전자전기컴퓨터공학과 석사. 2015년~현재 성균관대학교 전자전기컴퓨터공학과 박사과정. 관심분야는 플래시 메모리 DBMS, 데이터 분석



이 상 원
1991년 서울대학교 컴퓨터학과 학사. 1994년 서울대학교 컴퓨터학과 석사. 1999년 서울대학교 컴퓨터학과 박사. 1999년~2001년 한국 오라클. 2001년~2002년 이화여대 BK21 계약교수. 2002년~현재 성균관대학교 2015년~현재 성균관대학교 소프트웨어대학 교수. 관심분야는 플래시 메모리 DBMS