Internship in OCZ Technology

VLDB 연구실

오기환

wurikiji@gmail.com

OCZ Technology

- At San Jose, CA, USA
- SSD product
- Worked at Indilinx firmware team
  - 2012. 1. 3 ~ 2012. 2. 3 (약 32일)
- 오전 9시 출근 오후 6시 퇴근
- 실제 컴파일 이전에 반드시 코드 리딩 진행

VLDB Lab. @ SKKU
http://vldb.skku.ac.kr

용어 정리

- 섹터 : 최소 I/O 단위
- 페이지 : 섹터들의 집합, SATA 호스트와 펌웨어 사이의 기본 통신 단위
- 블록 : 플래시 메모리에서 데이터를 지우는 최소 단위

Why sector mapping ?

- Indilinx 정현모 이사님의 Idea
  - 윈도우 및 리눅스의 실제 사용자 환경에서의 트레이스 결과 모든 I/O 가 페이지 크기에 맞춰서 요청이 일어나는게 아님
  - 페이지 크기보다 작은 사이즈의 I/O 요청이 생각보다 많이 일어난다.
- 작은 단위(Sector 단위) 의 I/O 요청을 최대한 빨리 처리할 수 있도록 한다

VLDB Lab. @ SKKU
http://vldb.skku.ac.kr

Sector mapping 개발 이슈

- 매핑 테이블의 크기
  - 64MB memory 제한
  - 한 섹터당 4Byte 매핑 정보
    - 4GB만 해도 32MB memory 필요
- POR(정상적 파워 오프)의 구현
- GC의 구현
  - 섹터별 valid 여부의 저장

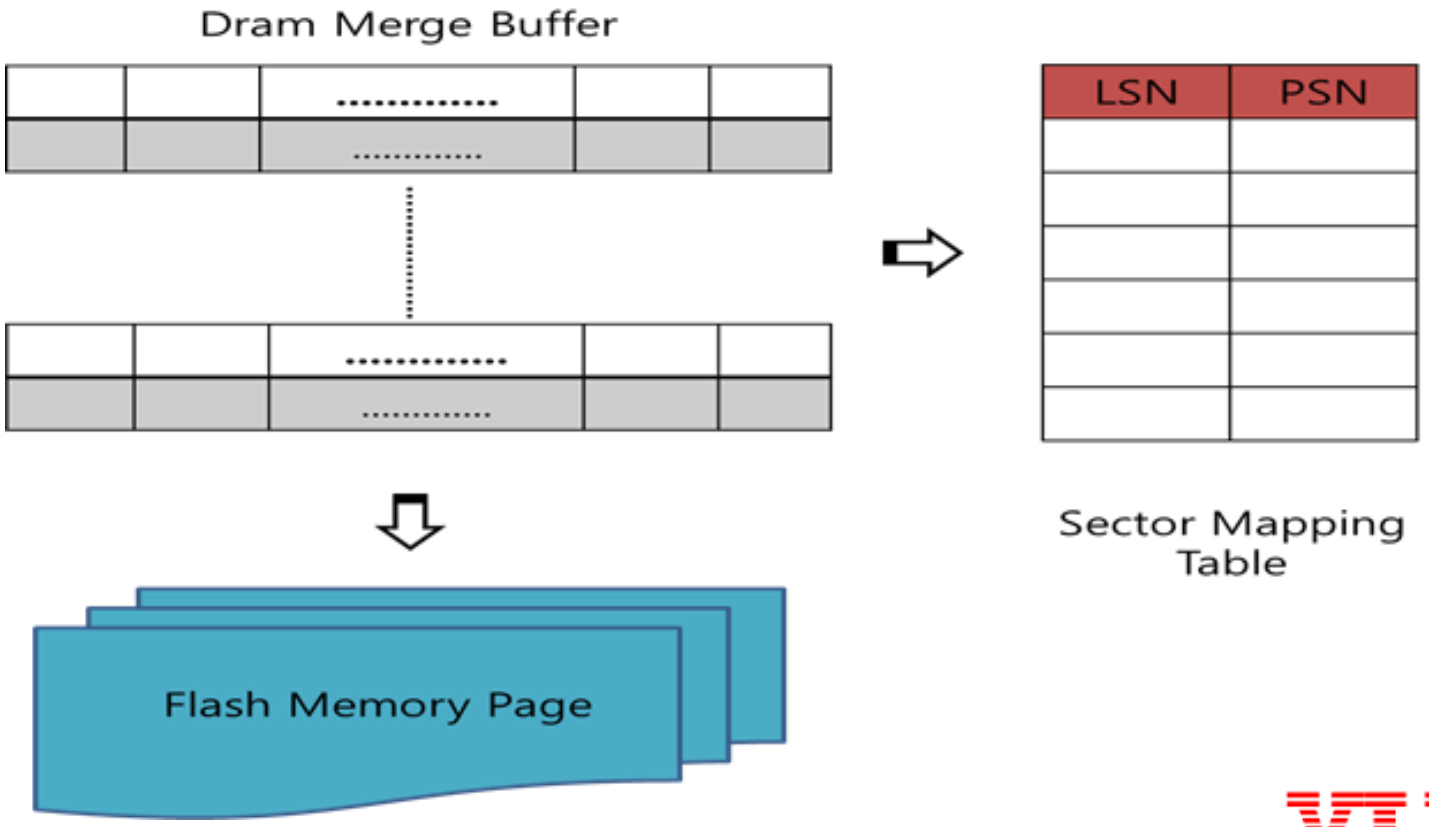
VLDB Lab. @ SKKU
http://vldb.skku.ac.kr

개발 과정

- 2GB SSD 기준의 기본 Sector mapping FTL의 구현
- 정상 종료 요청에 대한 POR 구현
- 2GB -> 32GB SSD 대상으로의 확장 구현
  - Mapping table 관리 기법의 변화
  - DRAM memory 및 nand Flash 관리 전략 수립
- 성능 테스트
- GC 구현

VLDB Lab. @ SKKU
http://vldb.skku.ac.kr

기본 Sector mapping FTL



VLDB Lab. @ SKKU
http://vldb.skku.ac.kr

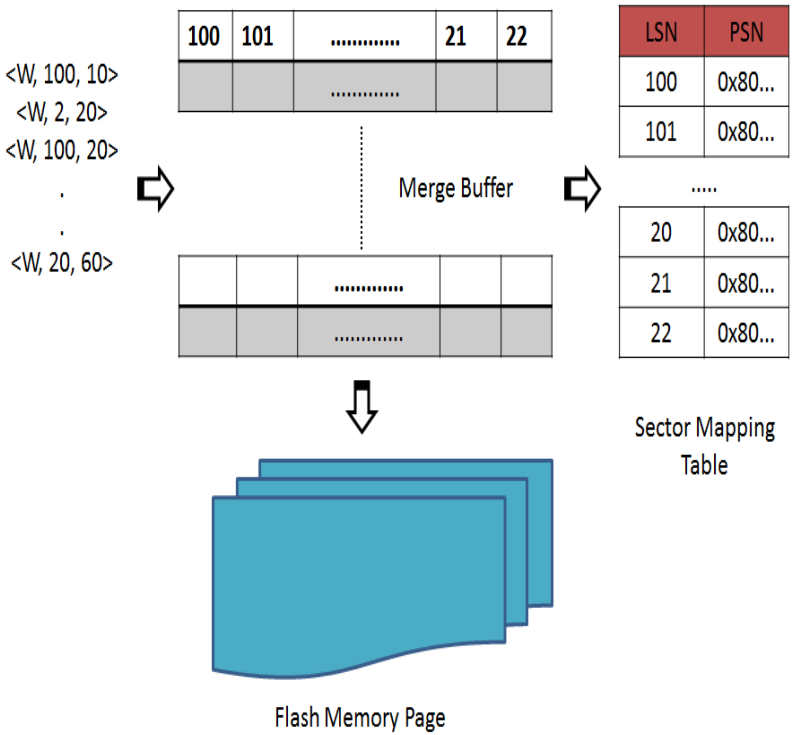
기본 Sector mapping FTL

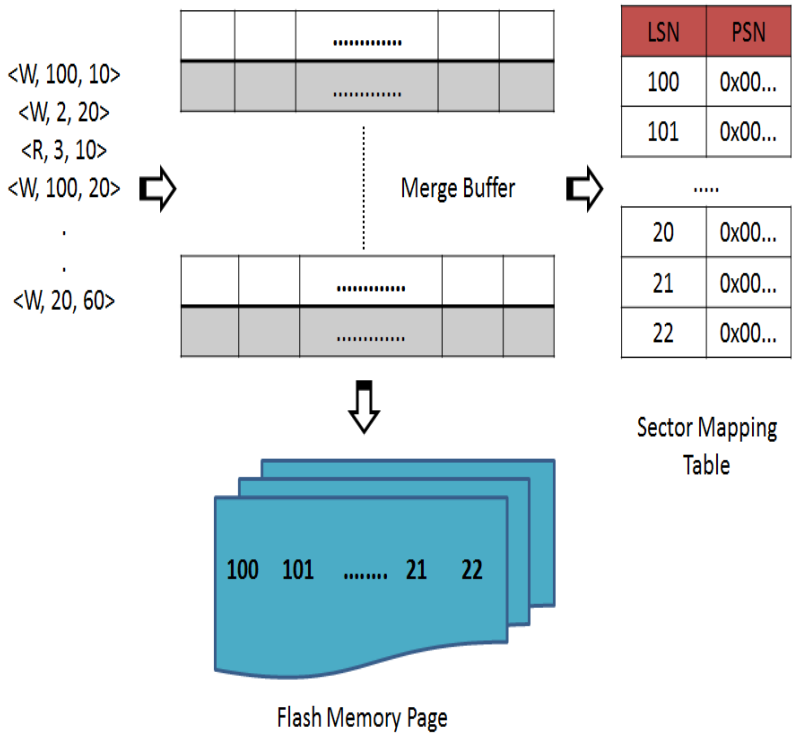
- 2GB SSD를 기준으로 시작
  - 16MB의 DRAM memory 공간 필요
- Sector들의 저장 방식의 지정
- Memory상의 데이터 관리
  - Nand로 Flash되지 않고 아직 memory에 존재하는 데이터의 Read 요청 처리

VLDB Lab. @ SKKU
http://vldb.skku.ac.kr

기본 Sector mapping FTL

- Write 요청 처리

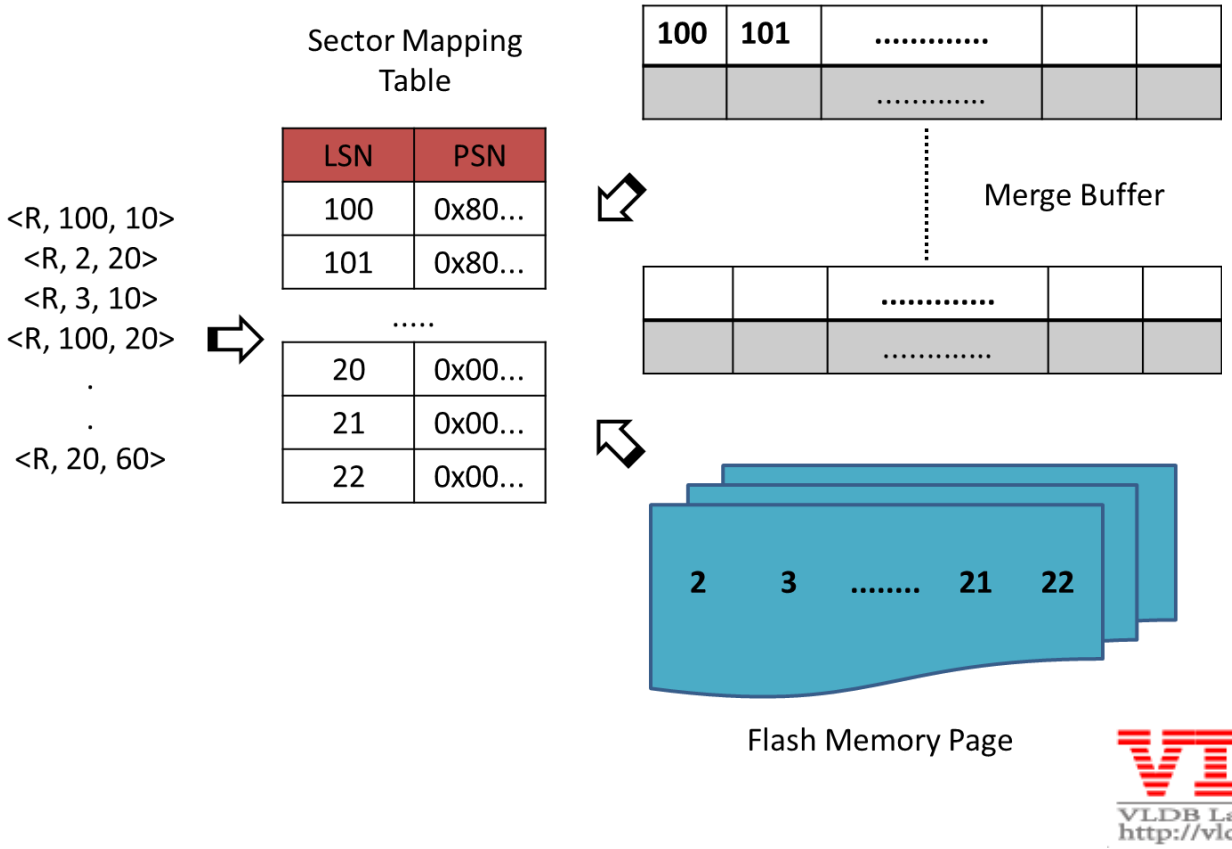




VLDB Lab. @ SKKU
http://vldb.skku.ac.kr

기본 Sector mapping FTL

- Read 요청 처리



VLDB Lab. @ SKKU
http://vldb.skku.ac.kr

확장 Sector mapping FTL

- 32GB, 64GB 대상으로의 확장
- 메모리 상에 mapping 정보를 모두 저장할 수 없음
  - Nand flash의 특정 위치를 mapping정보를 읽고 쓰는 공간으로 고정
  - Nand flash에서 mapping정보의 저장 위치를 가리키는 또다른 mapping 테이블 필요
  - 메모리 내에서 mapping 테이블에 대한 교체 정책 구현

VLDB Lab. @ SKKU
http://vldb.skku.ac.kr

확장 Sector mapping FTL

- 각 뱅크 별 mapping 테이블의 위치는 고정
  - ex) lsn 0~128 까지의 테이블은 bank 0의 3번째 블록에 위치하도록 고정
- Mapping 테이블을 저장할 메모리 공간이 부족할 경우에 대비한 교체 정책
  - FIFO 방식으로 교체
- 필요한 mapping 테이블은 요청이 들어올 때마다 nand flash에서 읽어옴

VLDB Lab. @ SKKU
http://vldb.skku.ac.kr

확장 Sector mapping FTL

- Nand flash의 사용 구조

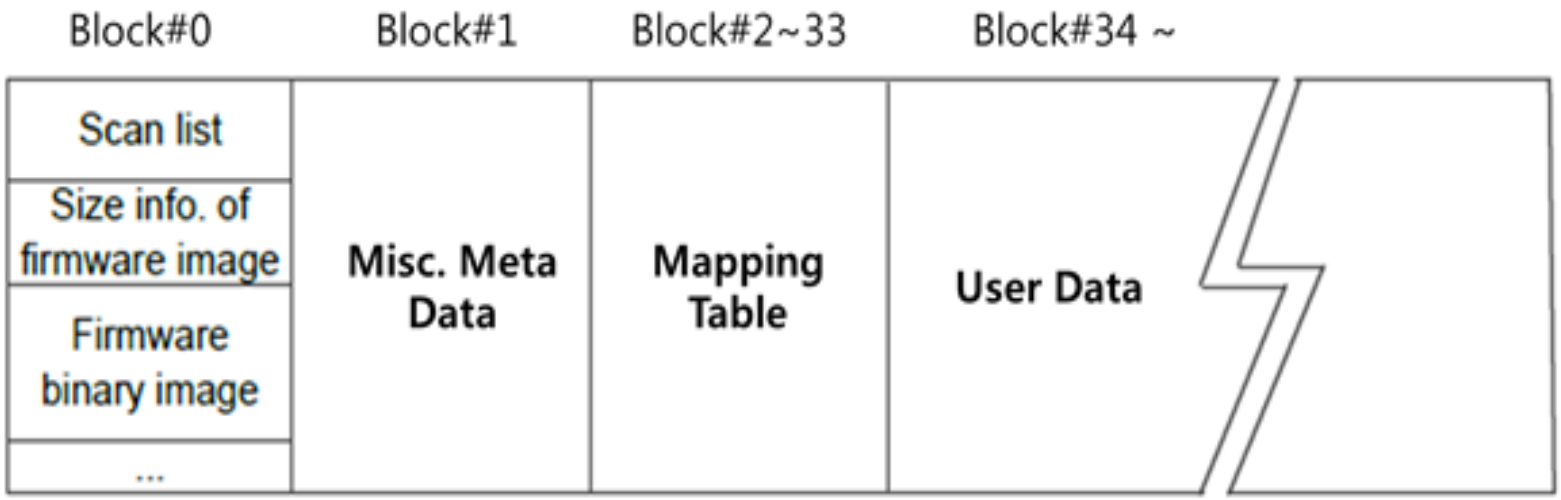


Figure 1. Bank 별 Flash Memory 사용 구조

VLDB Lab. @ SKKU
http://vldb.skku.ac.kr

확장 Sector mapping FTL

- #define SMT_ADDR (MERGE_BUFFER_ADDR + MERGE_BUFFER_BYTES)
- #define SMT_DRAM_BYTES (SECTORS_PER_BANK * sizeof(UINT32))
- #define SMT_BYTES (SECTORS_PER_BANK * sizeof(UINT32))
- #define SMT_PIECE_BYTES ((SMT_BYTES + NUM_BANKS_MAX -1)/ NUM_BANKS_MAX)
- #define SMT_INC_SIZE ((SMT_PIECE_BYTES + BYTES_PER_PAGE -1) / BYTES_PER_PAGE)
- #define SMT_LIMIT (PAGES_PER_VBLK / SMT_INC_SIZE)

VLDB Lab. @ SKKU
http://vldb.skku.ac.kr

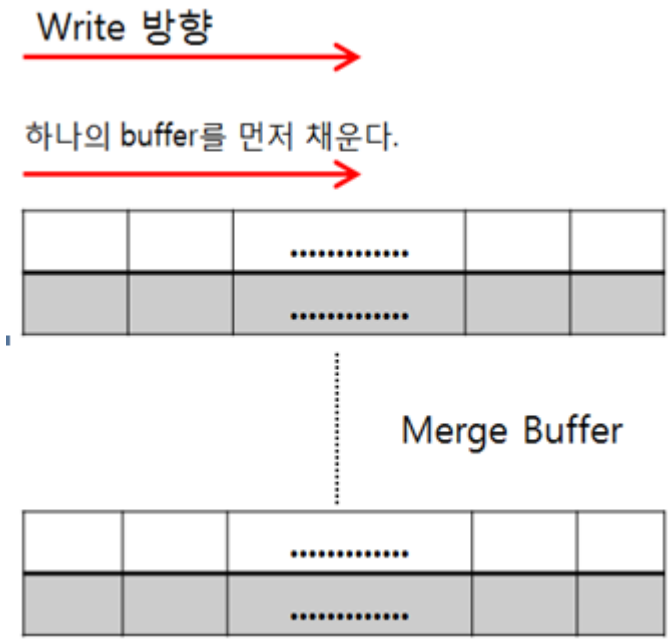
Write Buffering Technique

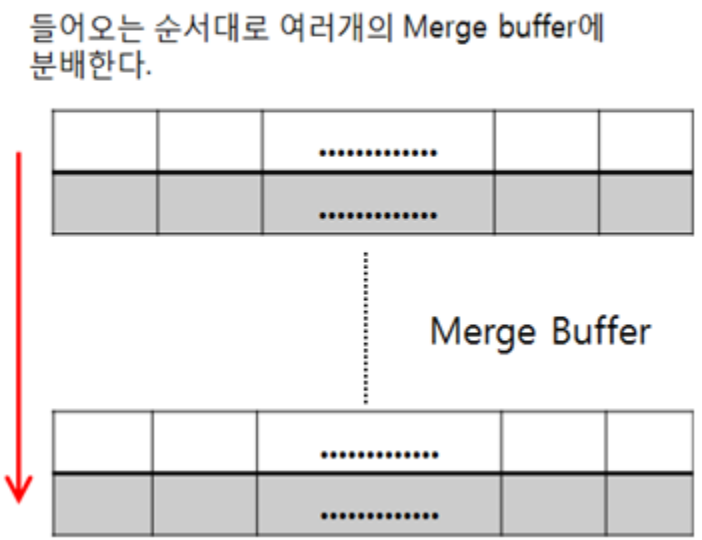
- Write 요청 처리시 섹터 단위로 곧바로 nand flash에 write하지 않고, buffer를 사용하여 page 크기만큼 모은 뒤에 write를 진행
- 크게 2가지 방법의 buffering 방식
  - 제일 첫 bank의 buffer부터 차례대로 꽉 채워 나가는 방법
  - 들어오는 순서대로 각 bank의 buffer로 병렬 분산 시키는 방법

VLDB Lab. @ SKKU
http://vldb.skku.ac.kr

Write Buffering Technique

- 여러가지 write buffering 기법





VLDB Lab. @ SKKU
http://vldb.skku.ac.kr

Write Buffering Technique

- #define MERGE_BUFFER_BYTES (((NUM_BANKS * BYTES_PER_PAGE + BYTES_PER_SECTOR - 1) / BYTES_PER_SECTOR)* BYTES_PER_SECTOR)

VLDB Lab. @ SKKU
http://vldb.skku.ac.kr

POR

- Firmware에서 주기적으로 호출되는 flush() 및 정상적인 종료 요청에 대비한 Meta data의 보관
- 메모리 내에 있는 mapping 테이블 및 아직 buffer에 남아있는 모든 data들을 nand flash로 내림
  - Buffer에 남아있는 data를 모두 내리고 mapping table 업데이트
  - mapping 테이블을 모두 내림
  - 기타 meta 데이터들을 내림
- Meta data를 위치시킬 곳은 배드 블록이 아니어야 함

VLDB Lab. @ SKKU
http://vldb.skku.ac.kr

구현 이슈

- CPU의 memory 직접 접근 vs 메모리 유틸리티 사용
- SATA와 FTL간의 요청 큐 limit 조절
  - bm_read_limit, bm_write_limit 의 직접 조절
- Buffer에서 nand flash로의 atomic write 보장
- Memory 접근 감소 vs 저장 공간 활용도 증가
- Mapping 테이블 정보의 atomic read/write 보장
- Mapping 테이블 관리 크기

VLDB Lab. @ SKKU
http://vldb.skku.ac.kr

성능 평가

- 측정 도구 : IOmeter, ATTO
- 측정 방법
  - IOmeter – Single User Test
    - Random Write transfer size : 512Bytes, 1KB, 2KB, 4KB
    - Sequential Write/Read transfer size : 128KB
  - ATTO testing environment
    - Queue depth : 10
    - Transfer size : 512Bytes, 1KB, 2KB, 4KB, 8KB, 16KB, 32KB

VLDB Lab. @ SKKU
http://vldb.skku.ac.kr

성능 평가

- 측정 FTL
  - 1-buffer Sector mapping
  - Dynamic Sector mapping
  - Static Sector mapping
  - Multi copy Sector mapping
  - Tutorial FTL
  - Greedy FTL

VLDB Lab. @ SKKU
http://vldb.skku.ac.kr

성능 평가

- 2GB 대상 FTL의 Random Write (IOPS)

IOPS	1_buffer	Dynamic	Static	Multi_copy	Tutorial_1	Greedy
512B	26930	27213	27868	28370	1400	1300
1KB	22075	22773	17485	24573	1340	1280
2KB	14415	13250	8858	19845	1270	1250
4KB	7377	6885	4693	10470	1200	1120

- 기존 FTL 대비 매우 빠른 Random Write 처리

VLDB Lab. @ SKKU
http://vldb.skku.ac.kr

성능 평가

- Sequential Read/Write 테스트

MB/S	1_buffer	Dynamic	Static	Multi_copy	Tutorial_1	Greedy
WRITE	31.5	29	33	62	90	85
READ	1.8	6.88	6.87	1.8	77	77

- Write buffering 기법 별로 서로 다른 결과

VLDB Lab. @ SKKU
http://vldb.skku.ac.kr

성능 평가

- 32GB 대상으로 확장된 FTL의 성능 평가
  - Mapping 정보의 nand read/write가 추가됨

IOPS	1_buffer	Dynamic	static	Multi_copy	Tutorial	Greedy
Single_512B	1293	1177	1261	540	1400	1300

- 매우 저하된 성능
  - Mapping 정보 load overhead

VLDB Lab. @ SKKU
http://vldb.skku.ac.kr

결과

- 2GB대상의 FTL 실험 결과를 보면 Mapping 테이블이 모두 DRAM Memory 상에 존재할 수 있는 경우에는 작은 단위의 I/o요청에서는 기존의 FTL보다 성능이 매우 향상된 것을 알수 있다
  - 512bytes 기준 약 100배 정도의 차이
- Sector mapping 사이에서도 write buffering의 방식에 따라서 성능 차이를 보이는 것을 알 수 있다.

VLDB Lab. @ SKKU
http://vldb.skku.ac.kr

결과

- Mapping 테이블이 DRAM에 모두 존재하지 않을 경우 매우 큰 성능 하락이 있음
  - Mapping 테이블을 NAND 에서 DRAM으로, DRAM에서 NAND로 옮기는 overhead
  - Random write 테스트시 miss rate가 90%이상

VLDB Lab. @ SKKU
http://vldb.skku.ac.kr

결론

- Map 테이블이 모두 DRAM에 상주할 수 있는 경우는 매우 뛰어난 Random I/O성능을 볼 수 있으나, 현실적으로 불가능
  - 128GB 용량을 위해 1GB의 메모리가 필요
- Mapping 테이블을 좀 더 효과적으로 관리할 기법이 필요

VLDB Lab. @ SKKU
http://vldb.skku.ac.kr