플래시SSD에 적합한 외부 합병정렬
(External Mergesort for FlashSSDs)

이 준 희 † 노 홍 찬 †† 정 원 묵 †† 박 상 현 †††
(Joonhee Lee) (Hongchan Roh) (Wonmook Jung) (Sanghyun Park)

요 약 합병정렬은 가장 널리 알려진 외부정렬 알고리즘 중 하나로, 정렬하고자 하는 데이터가 가용 메모리보다 더 클 때 사용된다. 합병정렬에 필요한 I/O 시간을 줄여 전체적인 효율을 개선한 기존의 연구들이 많았으나 이러한 시도들은 합병정렬이 하드디스크에서 작동한다는 가정 하에서 이루어졌다. 플래시SSD는 차세대 저장매체로 대두되고 있으며 하드디스크를 대체할 수 있을 것으로 기대된다. 플래시SSD는 기계적으로 움직이는 부분이 없기 때문에 하드디스크보다 훨씬 빠른 접근시간을 갖는다. 또한 내부 병렬성을 활용하여 하드디스크보다 훨씬 높은 I/O 대역폭을 발휘할 수 있다. 본 논문은 플래시SSD에 적합한 합병정렬인 플래시 합병정렬을 제안한다. 플래시 합병정렬은 합병에 필요한 데이터 블록의 순서인 블록 소모 순서를 런 생성 단계에서 미리 계산하고, 합병 단계에서 이 순서를 이용해 여러 런으로부터 한꺼번에 데이터 블록을 읽어 I/O 시간을 대폭 줄인다.

키워드: 외부정렬, 합병정렬, 플래시 메모리, SSD

Abstract Mergesort is one of the most widely-known external sorting algorithms, which is used when input data is larger than the amount of available memory. There were several attempts to improve mergesort by reducing I/O time, with an assumption that sorting takes place on HDDs. FlashSSDs are emerging as next generation storage devices and becoming alternatives to HDDs. FlashSSDs outperform HDDs in access latency, because they have no mechanical heads to move. In addition, flashSSDs benefit from its internal structure by exploiting internal parallelism, resulting in high I/O bandwidth. In this paper, we propose an external mergesort algorithm for flashSSDs called Flash mergesort. Flash mergesort computes the block consumption sequence in the run generation phase, which is the order of blocks that are needed in the merge phase. With this sequence, multiple blocks from multiple runs are read simultaneously into main memory in the merge phase, to reduce I/O time dramatically.

Keywords: external sorting, mergesort, flash memory, SSD

· 이 논문은 2013년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(NRF-2012R1A2A1A01010775)

† 학생회원 : 연세대학교 컴퓨터과학과
joonnc@cs.yonsei.ac.kr
†† 비 회 원 : 연세대학교 컴퓨터과학과
fallsmal@cs.yonsei.ac.kr
jngwnmk@cs.yonsei.ac.kr
††† 종신회원 : 연세대학교 컴퓨터과학과 교수
sanghyun@cs.yonsei.ac.kr
(Corresponding author임)

논문접수 : 2013년 7월 10일
심사완료 : 2013년 10월 14일

정보과학회논문지: 데이타베이스 제41권 제1호(2014.2)

1. 서 론

최근 몇 년 사이에 플래시SSD는 차세대 저장장치로 부각되었다. 여러 개의 플래시 패키지가 병렬적으로 연결된 플래시SSD는 높은 I/O 대역폭(bandwidth)과 짧은 접근시간(access latency)으로 여러 분야에서 채택되었다.

합병정렬은 가장 널리 알려진 외부정렬 알고리즘으로 정렬하고자 하는 전체 데이터를 메모리에 올리지 못할 때 사용된다[1]. 합병정렬은 여러 개의 정렬된 런(run)을 생성하는 런 생성 단계와, 생성된 런을 하나로 합치는 합병 단계로 구성된다. 일반적인 합병정렬은 합병단계(merge phase)에서 특정 런에 할당된 입력버퍼의 블록이 다 소모되어야 해당 런의 다음 블록을 읽는다. 그러나 이러한 방법은 한번에 읽는 블록의 수가 제한적이

므로 플래시SSD의 I/O 대역폭(bandwidth)을 활용할 수 없다. 플래시SSD의 I/O 대역폭을 최대한 사용하기 위해서는 플래시SSD만이 갖는 특징인 내부 병렬성(internal parallelism)을 활용해야 한다.

플래시 메모리 상에서 실행되는 합병정렬에 대한 연구가 없었던 것은 아니다. [2]은 플래시 메모리가 DBMS의 저장장치로서 갖는 잠재력을 설명하면서 핵심적인 데이터베이스 작업인 외부 정렬을 하나의 예로 들었다. 외부 정렬의 런 생성 단계에서는 순차 쓰기(Sequential write)가 수행되어 하드디스크와 플래시 메모리간의 성능 차이가 크지 않으나, 합병 단계에서 임의 읽기(Random read)가 반복적으로 나타날 때, 접근시간이 짧은 플래시 메모리가 하드디스크보다 더 유리하다는 것이다. 또한 Yang Liu는 부분적으로 정렬된 데이터(partially sorted data)가 갖는 특성과 플래시 메모리 만의 특성을 결합하여 런 생성 단계의 쓰기 횟수를 감소시키는 방법을 제시하였다[3]. 그러나 전체 성능에 가장 핵심적인 합병 단계의 성능을 직접 개선한 연구는 없었다.

본 논문은 플래시SSD에 적합한 합병정렬인 플래시 합병정렬을 제안한다. 플래시 합병정렬은 플래시SSD 상에서 합병 단계의 성능을 향상시키는데 중점을 둔 최초의 합병정렬이다. 플래시 합병정렬은 런 생성 단계에서 각 블록의 최대키 값을 따로 저장한 후, 이것을 정렬함으로써 합병 단계에서 필요할 블록의 순서를 미리 계산한다. 합병 단계에서는 이 순서를 참조하여 동시에 여러 런으로부터 가능한 한 많은 블록을 읽는다. 이 방법은 여러 런으로부터 블록을 읽을 때, 임의 읽기가 발생하므로 각각의 블록을 읽을 때마다 물리적인 디스크 암이 움직여야 하는 하드디스크에서는 부적합하다. 반면 플래시SSD는 물리적인 움직임이 없어 접근 시간이 훨씬 짧고, 내부 병렬성을 최대한 활용하여 플래시SSD의 최대 대역폭을 사용할 수 있어, 전체 I/O 시간을 크게 줄일 수 있다.

본 논문의 구성은 다음과 같다. 1장의 서론에 이어 다음 장에서는 플래시SSD의 특징과 합병정렬을 개선하기 위한 기존의 연구를 소개한다. 3장에서는 플래시 합병정렬의 알고리즘을 소개하고 4장에서는 실험결과에 대해 분석한다. 마지막으로 5장에서 결론을 도출한다.

2. 배경 및 관련연구

2.1 플래시SSD의 특징

플래시SSD는 그림 1과 같이 호스트 인터페이스, 컨트롤러, 여러 개의 데이터 채널과 플래시 메모리 패키지로 이루어진다. n개의 플래시 메모리 패키지가 하나의 채널에 병렬적으로 연결되고 m개의 채널이 SSD 컨트롤러에 연결된다. SSD 컨트롤러는 CPU와 램 버퍼로 구성된다.

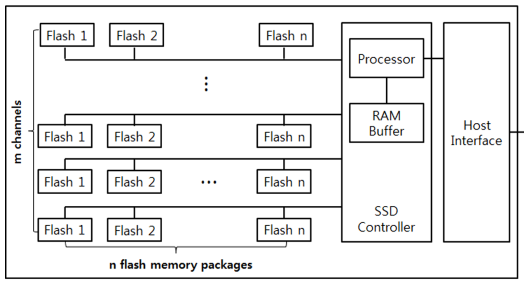


그림 1 플래시SSD의 내부 아키텍처
Fig. 1 Internal architecture of flashSSDs

이러한 구조 때문에 플래시SSD는 하드디스크에 존재하지 않는 내부 병렬성[4]이라는 특징을 갖는다. 내부 병렬성을 최대한 활용하기 위해서는 여러 개의 I/O 요청이 m 개의 채널에 전달되어야 하고(채널 레벨 병렬성) 각각의 채널이 I/O 요청을 n 개의 플래시 메모리 패키지에 분배해야 한다(패키지 레벨 병렬성).

최근 플래시SSD의 내부 병렬성을 최대한 활용하기 위한 새로운 I/O 요청 인터페이스, Psync I/O가 등장하였다[5]. Psync I/O는 리눅스 libaio API[6]로 구현된 모듈로 여러 개의 랜덤 I/O 요청을 묶어서 플래시SSD로 전달하고 플래시SSD는 요청된 I/O를 한꺼번에 처리한다.

2.2 하드디스크에서 합병정렬을 개선한 사례

LuoQuan은 하드디스크에서 합병정렬을 개선하기 위해 교차배치(interleaved layout)와 읽기 스케줄링(read scheduling)과 같은 방법을 제안하였다[7].

교차배치는 각 런의 블록들을 디스크에 교차시켜 저장함으로써 합병 단계에서 디스크 암(disk arm)의 움직임을 줄인다. 런 생성 시 각 런의 첫 번째 블록들을 연속된 공간에 저장하고 이어서 두 번째 블록들을 연속되게 저장하는 방식이다. 런 생성 단계에서 런들을 디스크에 쓰는데 더 많은 시간이 걸리지만, 합병 단계에서 특정 블록이 필요할 때 그 블록이 현재 디스크 암의 위치와 가까운 곳에 있을 확률이 높으므로 탐색시간을 줄일 수 있다.

읽기 스케줄링은 합병 단계에서 필요한 블록들의 순서인 블록 소모 순서(block consumption sequence)를 계산한다. 이후, 정렬이 어긋나지 않는 한도 내에서 디스크 탐색 횟수가 최소화되도록 이 순서를 수정하는데 하드디스크는 같은 실린더 내의 여러 블록에 디스크 헤드를 움직이지 않고 접근할 수 있기 때문이다. 이렇게 수정된 순서를 읽기 순서(read sequence)라고 하며, 합병단계에서 이 순서대로 블록들을 읽는다.

Graefe는 [8]에서 하드디스크와 주 메모리(RAM) 사이에 플래시 메모리를 두어 3개의 레벨로 이루어진 메모리 계층도(memory hierarchy)를 구성하였다. 이 방식은 외부 합병정렬의 알고리즘 자체를 수정한 것은 아

니지만, 플래시 메모리가 계층 중간에서 캐시 역할을 하여 메모리를 증폭시키는 효과를 낸다. 따라서 적은 메모리에서도 비교적 큰 파일을 빠른 시간 내에 정렬할 수 있으며, 메모리 사용이 제한적인 DBMS의 쿼리 프로세싱에도 적합하다.

3. 플래시 합병정렬

기존의 합병정렬과 같이, 플래시 합병정렬 역시 두 단계로 구성된다.

플래시 합병정렬은 [7]에서 제시된 읽기 스케줄링과 같이 런 생성 단계에서 블록 소모 순서를 계산하나 이 순서를 수정하지는 않는다. 플래시SSD는 하드디스크와 달리 내부 병렬성을 이용해 여러 개의 비연속적인 블록들을 동시에 읽을 수 있기 때문이다.

3.1 런 생성 단계(Run Formation Phase)

플래시 합병정렬은 다음과 같이 런을 생성한다. 먼저 입력버퍼에 데이터를 가득 채운다(알고리즘1의 2줄). 이후 각 튜플의 키, 버퍼내의 위치만을 따로 배열 등에 저장하여 정렬하고(3-5줄) 첫 번째 튜플부터 출력버퍼에 복사한다(7줄). 출력버퍼로 튜플을 복사한 후 출력버퍼의 블록이 가득 차면 해당 튜플은 그 블록 내에서 최대 키 값을 갖게 되므로 튜플의 키와 런 번호(몇번째 런인지)를 블록 소모 순서에 저장한다(8-9줄). 출력버퍼의 모든 블록이 다 찼으면 출력버퍼의 내용을 디스크에 쓰고 여분의 블록이 있으면 다음 튜플부터 그 블록에 저장한다(10-12줄). 루프가 반복될 때 마다 하나의 정렬된 런(sorted run)이 생성되고 루프문이 종료된 후 블록 소모 순서를 최대키 값으로 정렬한다(13줄).

Algorithm 1: Run Formation Phase of Flash mergesort

```
1:  while there are more blocks to read do
2:      Read data into the input buffer
3:      for each tuple in the input buffer do
4:          Store tuple's key, position pair in an array
5:      Sort the array by the keys
6:      for each pair in the array do
7:          Copy a tuple from a position to the output buffer
8:          if the current output buffer block is full
9:              Store the tuple's key and run number into
                the block consumption sequence
10:             if all output buffer blocks are full
11:                 Flush output buffer to a disk
12:                 else Set next block of current output buffer
                    block be a current output buffer block
13: Sort the block consumption sequence
```

플래시 합병정렬은 블록 소모 순서를 계산하고 저장하는데 추가적인 메모리를 필요로 한다. 그러나 이 추가적인 메모리는 전체 데이터 크기에 비해 아주 적으며(한 블록 당 8byte 내외), 블록 크기에 반비례한다.

3.2 합병 단계(Merge Phase)

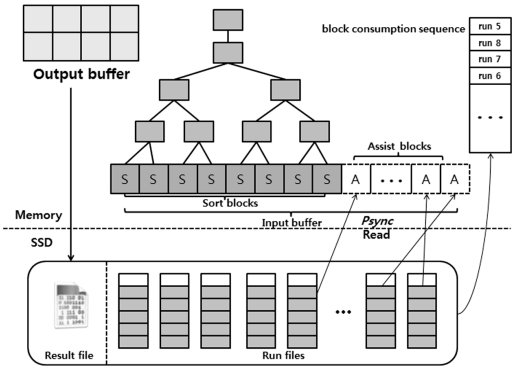
플래시 합병정렬의 합병단계는 알고리즘 2와 같다. 먼저 일반적인 합병정렬과 같이 모든 런의 첫 번째 블록이 입력 버퍼에 올라간다(알고리즘 2의 1줄). 이 블록들은 곧바로 정렬에 참여하므로 정렬블록(Sort Block) 이라 칭한다. 이 후 그림 2(a)와 같이 정렬블록을 채우고 남은 입력 버퍼의 공간에 정렬과정에서 정렬블록이 소모됨에 따라 필요한 다음 블록들을 블록 소모 순서를 참조하여 동시에 읽어온다(4줄). 이러한 블록들을 보조블록(Assist Block)이라 한다. 동시에 여러 개의 비연속적인 블록들을 입력버퍼에 올리기 위해 앞에서 언급한 PSync I/O를 사용한다. 정렬블록과 보조블록은 입력버퍼 상에서 고정되지 않고 정렬이 진행됨에 따라 위치가 유동적으로 바뀐다. 보조블록들은 정렬과정에 당장 사용되지는 않지만, 정렬블록의 특정 블록이 소모될 때마다 차례대로 정렬블록이 되기 때문이다(8-11줄). 이 때 소모된 정렬블록은 그림 2(b)와 같이 빈 블록이 된다. 정렬이 진행됨에 따라 입력버퍼의 보조블록이 모두 없어지면 다시 블록 소모 순서를 참조하여 여러 런으로부터 블록들을 동시에 읽어와 빈 블록들을 채운다(4줄). 이때 채워진 블록들이 다시 보조블록이 된다.

Algorithm 2: Merge Phase of Flash mergesort

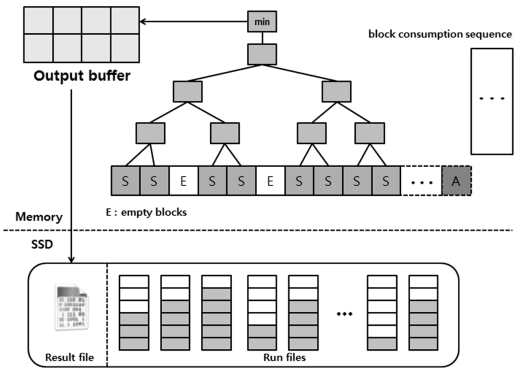
```
1:  Read the first block of each run into the input buffer
    and mark them sort block
2:  while the number of sort block ≠ 0 do
3:      if there is no assist block in the input buffer
4:          PSync_Read next blocks into the empty block
            by referencing block consumption sequence
5:      Retreive a minimum tuple among the sort blocks
        and Copy it into the output buffer
6:      if all output buffer blocks are full
7:          Flush output buffer to a disk
8:      if a sort block having a minimum tuple is exhausted
9:          the sort block becomes empty block
10:         if assist block exists in the input buffer
11:             first assist block becomes sort block
12:         else
13:             The number of sort blocks--
```

실제 정렬은 패자트리(Tree of Loser) 안에서 이루어진다. 패자트리는 트리의 최하단(leaf)에서 노드의 삽입이 일어나고, 모든 노드가 토너먼트를 진행하여 트리의 루트에 최소 노드가 저장된다. 또한 새로운 노드가 트리에 삽입될 때, 기존에 남겨진 트리 구조를 활용하여 log(n)의 비교만으로 최소 노드를 찾을 수 있다. 루트로 올라온 최소 튜플은 출력버퍼에 복사되고 출력버퍼가 찰 때마다 디스크에 쓰는 작업이 수행된다(5-7줄).

정렬이 끝나가는 시점에는, 더 이상 읽을 블록이 없어 입력버퍼에 보조블록이 들어오지 않게 된다. 이후에는



(a) before the actual sorting begins



(b) after some sorting is done

그림 2 합병 단계에서의 메모리 상태

Fig. 2 Memory state during merge phase

정렬블록이 소모될 때마다 정렬블록의 수가 감소하고(13줄), 정렬블록이 모두 소모되면서 정렬이 완료된다(2줄).

4. 실험결과

플래시 합병정렬과 기존의 합병정렬을 비교하는 실험을 수행하였다. 기존의 합병정렬은 단일블록 합병정렬(Single-block mergesort)과 복수블록 합병정렬(Multi-block mergesort)으로 구현하였는데 두 알고리즘의 런 생성 단계는 동일하다. 다만 단일블록 합병정렬은 전체 입력버퍼의 크기와 상관없이 합병 단계에서 한 런당 한 블록만을 입력버퍼로 할당하는 반면 복수블록 합병정렬은 가능한 한 많은 블록을 각 런의 입력버퍼로 할당한다. 예를 들어, 전체 입력버퍼로 사용가능한 메모리가 400블록이고 런의 개수가 200개 생성된 경우 단일블록 합병정렬은 합병 시 각 런당 1개의 블록만을 입력버퍼로 할당하지만, 복수블록 합병정렬은 2개의 블록을 할당한다. 플래시 합병정렬은 단일블록 합병정렬과 같이 한 런당 하나의 블록을 입력버퍼로 사용하고, 여분의 블록은 모두 보조블록으로 사용한다.

4.1 실험환경

오픈소스 DBMS인 PostgreSQL 소스를 수정하여 데이터 접근 모듈을 구현하고, 이를 사용해 데이터를 로드한 후 합병정렬의 성능을 측정하였다. 데이터 파일은 공식적인 TPC-H[9] 데이터베이스 생성 도구인 DBGen을 사용하여 PostgreSQL 데이터 파일 형식으로 생성하였다. 여러 개의 TPC-H 테이블 중 가변 필드를 포함하여 9개의 필드로 구성된 ORDERS 테이블을 선택하고, ORDERDATE 필드를 정렬키(Sort Key)로 지정하였다. 표 1은 DBGen의 scale factor를 변경해가며 생성된 3개의 ORDERS 테이블 파일을 나타낸다.

표 1 실험에서 사용한 데이터 파일

Table 1 Data files used in experiment

	Data size	# of tuples	minimum input buffer size for single-pass merge
Data A	220MB	1,500,000	168 blocks(≒1.3MB)
Data B	1.1GB	7,500,000	384 blocks(=3MB)
Data C	2.2GB	15,000,000	538 blocks(≒4.2MB)

실험에서 I/O의 단위(블록 크기)는 실제 DBMS의 페이지(page) 크기인 8KB로 고정하였다. 전체 입력버퍼의 크기는 표 1과 같이 합병 단계가 단일 패스(Single-pass)로 완료될 수 있는 최소 크기에서부터 점차 증가시켰다. 입력버퍼는 각 알고리즘의 두 단계에서 모두 사용되고 도중에 크기가 변하지 않으므로 런 생성과 합병에 사용되는 메모리양은 거의 동일하다. 입력버퍼와 달리 출력버퍼의 크기는 1MB로 고정하였다. 출력버퍼가 1MB 이상일 때 실험 성능에 거의 영향을 미치지 않았기 때문이다.

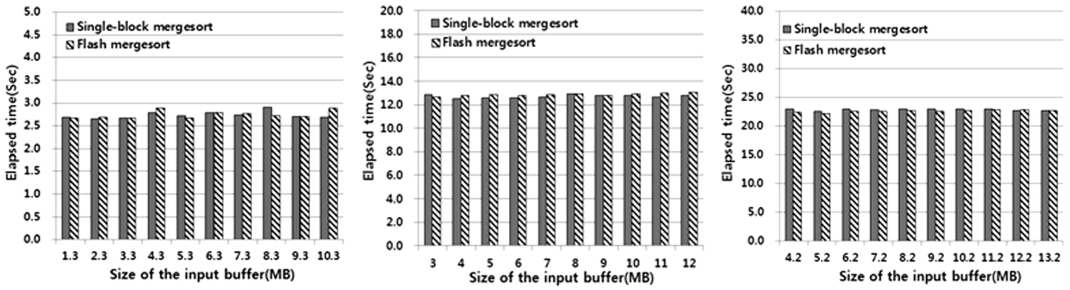
또한 실험의 모든 I/O는 직접 I/O(direct I/O) 모드로 수행되어 데이터 접근 모듈이 OS 캐시를 거치지 않고 플래시SSD와 직접 데이터를 주고 받았으며, 보다 정확한 시간 측정을 위해 동기 I/O(Synchronous I/O)를 사용하여 I/O와 CPU시간은 겹쳐지지 않았다. 따라서 선반입(prefetching)이나 이중 버퍼링 기법도 적용되지 않았다.

마지막으로 본 실험에서 사용할 플래시SSD로 마이크론 사의 P300[10]과 Fusion-io 사의 ioDrive[11]를 선택하였다.

4.2 런 생성 단계

플래시 합병정렬의 블록 소모 순서를 계산하는 오버헤드가 얼마나 큰지 알아보기 위해 런 생성 단계의 소요시간을 측정해보았다. 그림 3의 그래프는 P300에서 두 알고리즘의 런 생성 단계에 소요되는 시간이다. 단일블록 합병정렬과 복수블록 합병정렬의 런 생성 단계는 동일하므로 복수블록 합병정렬은 이 실험에서 제외하였다.

직관적으로 플래시 합병정렬은 런 생성 단계에서 블



(a) Data A (b) Data B (c) Data C

그림 3 P300에서 실험한 런 생성 단계

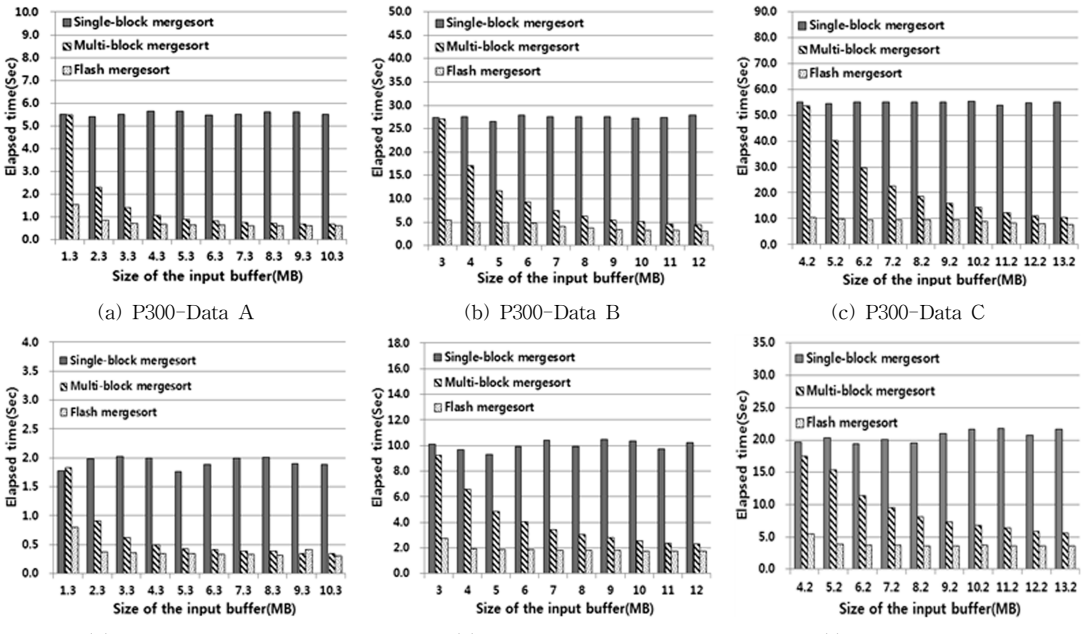
Fig. 3 Elapsed time of run formation phase on P300

록 소모 순서를 계산하기 때문에 더 많은 시간이 걸릴 것으로 예상된다. 그러나 블록 소모 순서를 계산하는데 별도의 I/O 없이 CPU 계산만이 추가되어 전체 런 생성 시간에 아주 미미한 영향만을 끼쳤다. 또한 런 생성 단계의 소요시간은 전체 입력버퍼의 크기와 크게 상관없이 일정하였다. 따라서 플래시 합병정렬이 갖는 오버헤드는 런 생성 단계의 성능에 거의 영향을 주지 않는다고 말할 수 있다.

4.3 합병 단계

다음으로 앞서 소개한 세 가지 알고리즘의 합병 단계에서의 소요되는 전체 읽기시간(Total read time of merge phase)을 측정하였다. 플래시 합병정렬의 주된 목표가 이 시간을 줄이는 데 있기 때문이다.

그림 4는 두 개의 플래시SSD에서 측정한 실험결과이고 표 2는 Data A의 합병과정에서 발생하는 읽기 작업(read operation)에 대한 정보이다. 단일블록 합병정렬은 전체 입력버퍼의 크기와 상관없이 일정한 읽기시간을 보여주었다. 단일블록 합병정렬은 한 번에 한 블록만을 읽고 전체 데이터의 총 블록 수는 입력버퍼의 크기와 상관없이 거의 일정하기 때문이다. 이는 표 2의 단일블록 합병정렬 항목에서도 확인할 수 있는데 전체 읽기 횟수가 완전히 동일하지 않은 이유는 입력버퍼의 크기



(a) P300-Data A (b) P300-Data B (c) P300-Data C

(d) ioDrive-Data A (e) ioDrive-Data B (f) ioDrive-Data C

그림 4 합병 단계에서 소요되는 전체 읽기시간

Fig. 4 Total read time of merge phase

표 2 Data A의 합병단계에서 수행되는 읽기작업
Table 2 Read operation of merge phase on Data A
(a) average number of blocks per one read operation

input buffer	Single-block mergesort	Multi-block mergesort	Flash mergesort
1.3MB	1	1.05	8
2.3MB	1	3.25	235
3.3MB	1	6.63	360
4.3MB	1	11.27	503
5.3MB	1	17.00	640
6.3MB	1	23.76	774
7.3MB	1	32.28	907
8.3MB	1	40.92	1038
9.3MB	1	51.83	1169
10.3MB	1	62.86	1299

(b) total number of read operation

input buffer	Single-block mergesort	Multi-block mergesort	Flash mergesort
1.3MB	26948	26276	3350
2.3MB	26879	8408	132
3.3MB	26853	4086	76
4.3MB	26852	2423	55
5.3MB	26854	1615	43
6.3MB	26852	1142	36
7.3MB	26845	851	31
8.3MB	26843	660	27
9.3MB	26846	521	24
10.3MB	26845	429	22

에 따라 생성되는 런의 개수가 다르고, 각 런의 마지막 블록에 빈 공간이 발생하기 때문이다.

복수블록 합병정렬의 읽기시간은 전체 입력버퍼의 크기가 작을 때는 단일블록 합병정렬의 읽기시간과 거의 동일하였으나 입력버퍼의 크기가 커질수록 급격히 감소하였다. 전체 입력버퍼의 크기가 커질수록 각 런의 입력버퍼로 할당되는 블록의 수가 증가하고, 이에 따라 I/O의 단위가 8KB의 배수로 커졌기 때문이다. I/O 단위가 커지면서 플래시SSD의 내부 병렬성 활용이 증가되어 복수블록 합병정렬이 단일블록 합병정렬보다 훨씬 높은 성능을 보였다.

마지막으로 플래시 합병정렬은 어떤 실험환경에서든 다른 두 알고리즘보다 우수한 성능을 보였다. 플래시 합병정렬은 합병에 곧 필요한 블록들만을 읽음으로써 플래시SSD의 대역폭을 불필요한 블록을 읽는데 낭비하지 않고 입력버퍼의 캐싱효과를 높일 수 있었다. 또한 각 런의 입력버퍼로 오직 한 블록을 할당하고 입력버퍼 대부분의 영역을 보조블록으로 채울 수 있어, 전체 입력버퍼를 각 런에 균등하게 배분하는 복수블록 합병정렬에 비해 한 번의 읽기로 불러오는 평균 블록의 개수가 급격하게 증가하였고(표 2(a)) 이는 총 읽기 횟수의 감소로 이어졌다(표 2(b)). 그런데 플래시 합병정렬의 읽기시간은 전체 입력버퍼의 크기가 커짐에 따라 처음엔 감소하였으나 특정 구간 이후로는 일정한 성능을 보였다. 플래시 합병정렬은 여러 개의 런에 흩어져 있는 블록들을 PSync I/O를 사용해 동시에 읽는데 블록들의 수가 점점 많아져 플래시 SSD의 대역폭을 최대로 사용하게 된 시점부터는 병렬성의 효과가 떨어졌기 때문이다.

4.4 전체 성능향상

마지막으로 표 3과 같이 플래시 합병정렬의 전체적인 성능을 다른 두 알고리즘과 비교하였다. 런 생성 단계와 합병 단계를 모두 포함했을 때 어느 정도 성능이 향상되는지 보기 위해서이다. 표 3에서 Spd-S는 단일블록 합병정렬에 비해 플래시 합병정렬이 몇 배 더 빠른지, Spd-M은 복수블록 합병정렬에 비해 몇 배 더 빠른지를 나타낸다.

전체 입력버퍼의 크기가 커질수록 런 생성 단계에서 생성되는 런의 개수가 작아지는데 단일블록 합병정렬은 합병 시 하나의 런에 대해 한 블록만을 입력버퍼로 할당하므로, 점점 사용하지 않는 메모리가 많아진다. 반면 플래시 합병정렬은 남는 메모리를 보조블록으로 활용하므로, Spd-S는 입력버퍼의 크기 비례하여 증가하는 경향을 보였다.

Spd-M은 전체 입력버퍼의 크기가 커짐에 따라 감소하는 경향을 띄었다. 입력버퍼가 커질수록 복수블록 합병정렬에서 한 런에 할당되는 블록의 수가 많아졌기 때문이다. 한 런당 할당되는 블록의 개수가 많아질수록 순차읽기의 I/O 단위가 점차 커져 복수블록 합병정렬의 성능이 플래시 합병정렬의 성능에 근접하였다.

추가적으로 Spd-S와 Spd-M 모두 데이터 크기에 비례하여 증가하는 경향을 보였다. 종합적으로 플래시 합병정렬은 정렬하려는 데이터의 크기가 크고, 사용가능한 메모리가 제한적일 때 기존 알고리즘에 비해 성능향상이 커지는 것을 알 수 있었다.

5. 결 론

본 논문은 플래시SSD에 적합한 합병정렬인 플래시 합병정렬을 제안하였다. 플래시 합병정렬은 합병단계에서 동시에 여러 런으로부터 많은 블록을 읽음으로써 플래시SSD의 높은 임의 I/O 대역폭을 최대한 활용한다.

실험에서 플래시 합병정렬의 합병 단계 읽기시간은 단일블록 합병정렬과 복수블록 합병정렬의 읽기시간에 비해 최대 9.4배, 5배 더 빠르게 수행되었고, 전체적으로 최대 2.2배, 2.05배 더 빠르게 수행되었다. 또한 실험결과에 의하면 플래시 합병정렬은 데이터 크기가 크고 가용 메모리가 적을 때 다른 알고리즘보다 우수한 성능을 보였다.

표 3 플래시 합병정렬의 성능 향상도

Table 3 Speedup factor of Flash mergesort

(a) P300-Data A

input buffer	# of run	Spd-S	Spd-M
1.3MB	160	1.764	1.779
2.3MB	91	2.031	1.357
3.3MB	64	2.161	1.200
4.3MB	49	2.109	1.091
5.3MB	40	2.225	1.094
6.3MB	34	2.131	1.052
7.3MB	29	2.155	1.045
8.3MB	26	2.198	1.054
9.3MB	23	2.221	1.016
10.3MB	21	2.087	0.978

(b) P300-Data B

input buffer	# of run	Spd-S	Spd-M
3MB	349	1.973	1.966
4MB	322	2.019	1.564
5MB	299	1.962	1.332
6MB	280	2.085	1.241
7MB	262	2.084	1.183
8MB	247	2.139	1.162
9MB	233	2.186	1.128
10MB	221	2.186	1.110
11MB	210	2.157	1.072
12MB	200	2.168	1.056

(c) P300-Data C

input buffer	# of run	Spd-S	SPd-M
4.2MB	419	2.074	2.052
5.2MB	399	2.141	1.813
6.2MB	381	2.133	1.534
7.2MB	364	2.136	1.379
8.2MB	349	2.145	1.292
9.2MB	335	2.143	1.227
10.2MB	322	2.173	1.180
11.2MB	311	2.131	1.126
12.2MB	299	2.156	1.105
13.2MB	289	2.205	1.107

(d) ioDrive-Data A

input buffer	# of run	Spd-S	SPd-M
1.3MB	160	1.468	1.520
2.3MB	91	1.994	1.388
3.3MB	64	2.034	1.220
4.3MB	49	2.030	1.150
5.3MB	40	1.854	1.075
6.3MB	34	1.954	1.102
7.3MB	29	2.024	1.059
8.3MB	26	2.058	1.067
9.3MB	23	1.908	1.002
10.3MB	21	1.960	1.036

(e) ioDrive-Data B

input buffer	# of run	Spd-S	SPd-M
3MB	349	1.869	1.763
4MB	322	1.914	1.617
5MB	299	1.813	1.373
6MB	280	1.971	1.335
7MB	262	2.031	1.241
8MB	247	1.951	1.195
9MB	233	2.001	1.146
10MB	221	2.015	1.144
11MB	210	1.951	1.118
12MB	200	1.972	1.084

(f) ioDrive-Data C

input buffer	# of run	Spd-S	SPd-M
4.2MB	419	1.827	1.683
5.2MB	399	1.986	1.713
6.2MB	381	1.925	1.491
7.2MB	364	1.983	1.409
8.2MB	349	1.925	1.311
9.2MB	335	2.049	1.288
10.2MB	322	2.070	1.241
11.2MB	311	2.080	1.226
12.2MB	299	2.011	1.207
13.2MB	289	2.092	1.190

DBMS에서 합병정렬은 조인, 중복제거, 교집합 연산 등을 수행할 때 사용된다[12]. DBMS의 특성상 여러 개의 연산이 파이프라인 방식으로 수행되고 따라서 각각의 연산을 수행하는 세션은 제한된 메모리만을 할당받는다. 플래시 합병정렬은 가용 메모리가 적고 데이터의 크기가 클 때 뛰어난 성능을 내므로, DBMS의 외부 정렬 알고리즘으로도 적합할 것이다.

References

[1] S. Dasgupta, C. Papadimitriou, and U. Vazirani, "Algorithms," 2008.

[2] S. Lee, B. Moon, C. Park, J. Kim, S. Kim, "A Case for Flash Memory SSD in Enterprise Database Applications," SIGMOD '08.

[3] Y. Liu, Z. He, Y. P. Chen and T. Nguyen, "External Sorting on Flash Memory Via Natural Page Run Generation," The Computer Journal Advance Access published June 2, 2011.

[4] F. Chen, R. Lee, and X. Zhang, "Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data processing," In High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on, pages 266-277. IEEE, 2011.

[5] H. Roh, S. Park, S. Kim, M. Shin, and S.-W. Lee, "B+-tree index optimization by exploiting internal parallelism of flash-based solid state drives," VLDB 2012.

[6] libaio, http://lse.sourceforge.net/io/aio.html.

[7] L. Zheng and P. Larson, "Speeding Up External Mergesort," IEEE Transactions on Knowledge and Data Engineering, vol.8, no.2, Apr. 1996.

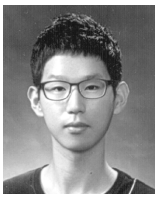
[8] G. Graefe, "Sorting in a Memory Hierarchy with Flash Memory," Datenbank Spektrum 2011.

[9] Transaction Processing Performance Council, TPC BENCHMARK™ H Standard Specification Revision 2.14.3, 2011.

[10] Micron, P300. http://www.micron.com/~/media/Documents/ Products/Data%20Sheet/SSD/p300_2_5.pdf.

[11] Fusion-Io, ioDrive, http://www.fusionio.com/load/-media-/1ufytn/docsLibrary/FIO_DS_ioDrive.pdf.

[12] R. Elmasri, and S. B.Navathe, "Fundamentals of DATABASE SYSTEMS 5th edition," 2007.



이 준 희

2013년 연세대학교 컴퓨터과학과(학사) 2013년~현재 연세대학교 컴퓨터과학과 석사과정. 관심분야는 플래시SSD, 분산처리 시스템

노 홍 찬

2006년 연세대학교 컴퓨터과학과(학사) 2008년 연세대학교 컴퓨터과학과(공학석사). 2008년~현재 연세대학교 컴퓨터과학과 박사과정. 관심분야는 데이터베이스 시스템, 플래시SSD

정 원 묵

2011년 연세대학교 컴퓨터과학과(학사) 2011년~2013년 연세대학교 컴퓨터과학과 석사과정. 관심분야는 클라우드 컴퓨팅, 정보 검색

박 상 현

1989년 서울대학교 컴퓨터공학과(학사) 1991년 서울대학교 컴퓨터공학과(공학석사). 2001년 UCLA 대학원 컴퓨터과학과(공학박사). 1991년~1996년 대우통신 연구원. 2001년~2002년 IBM.J.Watson Research Center Post-Doctoral Fellow 2002년~2003년 포항공과대학교 컴퓨터공학과 조교수. 2003년~2006년 연세대학교 컴퓨터과학과 조교수. 2006년~2011년 연세대학교 컴퓨터과학과 부교수. 2011년~현재 연세대학교 컴퓨터과학과 교수. 관심분야는 데이터베이스, 데이터마이닝, 바이오인포매틱스, 적응적 저장장치 시스템, 플래시 메모리 인덱스, SSD