# 데이터베이스의 원자성 쓰기 보장을 위한 파일시스템 및 SSD 개발

한규화 ○ 신동군

성균관대학교

hgh6877@skku.edu, dongkun@skku.edu

# Filesystem and SSD Implementation for Guaranteeing Atomic Write in Database

Kyuhwa Han○ Dongkun Shin

Sungkyunkwan University

**요 약**

최근 데이터베이스 관리 시스템에서는 이중 쓰기 버퍼 기법의 단점을 보완할 수 있는 원자성 쓰기 기법이 제시되었다. 그러나, 원자성 쓰기 기법을 지원하기 위해서는 파일시스템과 I/O 스케줄러 등의 I/O 서브시스템과 SSD의 원자성 쓰기의 보장이 필요하다. 본 연구에서는 16KB 연속 블록 할당 기법, 플래그 전달을 통한 I/O 명령 병합 방지 기법, 16KB 원자성 쓰기를 지원하는 SSD를 통해 MariaDB의 원자성 쓰기를 지원하는 연구를 진행하였다. Tpcc-mysql을 사용하여 본 연구의 기법과 MariaDB의 원자성 쓰기 기법의 성능을 평가한 결과, 데이터베이스의 처리량이 이중 쓰기 버퍼 기법을 사용한 MariaDB 대비 약 50%~110% 향상된 것을 확인하였다.

## 1. 서 론

스마트폰과 휴대용 전자기기의 사용이 증가함에 따라 클라우드 서비스와 사용자의 정보를 관리하는 서버의 사용도 증가하고 있다. 클라우드 시스템과 서버와 같은 커다란 정보를 저장하는 시스템은 데이터를 무결성 있게 관리하는 것이 중요하다. 이러한 시스템은 데이터베이스 관리 프로그램을 사용하여 데이터의 원자성, 일관성, 고립성, 영속성을 보장한다. 데이터베이스 관리 프로그램의 대표적인 예로는 MySQL과 MariaDB가 있다.

MySQL과 MariaDB는 데이터의 무결성을 보장하기 위하여 이중 쓰기 버퍼 기법을 사용한다[1][2]. 이 기법은 데이터 파일의 원자적인 변경을 위하여 이중 쓰기 버퍼 영역에 데이터의 쓰기가 완료된 후 파일의 원본 위치에 같은 데이터를 덮어쓴다. 이 기법을 사용하면 데이터의 원자성을 보존할 수 있으나, 같은 데이터를 서로 다른 두 영역에 써야 하기 때문에 쓰기 양이 2배로 증가한다.

이러한 이중 쓰기 버퍼 기법을 개선하기 위하여 원자성 쓰기(atomic write) 기법이 소개되었다[3]. 이 기법은 다이렉트 I/O를 사용하여 데이터를 원하는 위치에 바로 쓰는 기법이다. 이 기법을 사용하면 이중 쓰기 버퍼 기법에 비해 쓰기 양을 반으로 줄일 수 있지만, 데이터에 대한 원자성 쓰기를 지원해주는 저장 장치가 필요하다.

FusionIO에서는 이벤트 로깅 방식을 통하여 쓰기의 원자성을 보장하는 SSD를 제안하였고, DFS라는 독자적인 파일시스템을 통해 MariaDB에게 원자적인 쓰기를 제공하였다[3]. MariaDB는 DFS의 ioctl을 통해 파일을 원자성 쓰기 파일로 세팅할 수 있으며, DFS와 SSD가 원자성 쓰기 파일에 대해 원자적인 쓰기를 보장한다. 그러나, MariaDB의 원자성 쓰기는 FusionIO의 SSD와 DFS에서만 지원이 가능하다. 또한 FusionIO의 SSD와 DFS는 소스가 공개되지 않았으며, 관련 연구에서 기법에 대해 자세하게 서술하지 않았다. 본 연구에서는 리눅스의 고유 파일시스템인 EXT4 파일시스템[5], I/O 스케줄러, SSD를 수정하여, 16KB 데이터의 원자성 쓰기를 보장하는 기법을 제시한다.

본 논문은 2013년도 정부(교육부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임(2013R1A1A2A10013598)

## 2. 16KB 원자성 쓰기 보장 SSD

SSD는 MariaDB의 원자성 쓰기의 무결성을 보장하기 위해서 쓰기 명령을 원자적으로 처리하는 기능을 지원해야 한다. 그러나, SSD의 페이지 단위(4KB 또는 8KB)와 MariaDB의 데이터 단위(16KB)가 서로 다르기 때문에 MariaDB의 데이터는 여러 물리 페이지(2개 또는 4개)에 나누어져 저장된다. 이 때 여러 페이지들은 서로 다른 칩에 저장되며 동시에 처리된다. 만약 쓰기 명령을 처리하는 중간에 SSD의 전원이 갑작스럽게 차단될 경우, 모든 페이지의 저장이 완료됨을 보장할 수 없다. 이로 인해 16KB 데이터의 원자성을 보장하지 못한다.

SSD는 일반적으로 SPOR(Sudden Power Off Recovery)을 위하여 물리 페이지의 스페어 영역에 논리 페이지 번호(LPN; Logical Page Number)를 저장한다. 그리고, 갑작스럽게 전원이 차단된 경우 스페어 영역에 저장해둔 논리 페이지 번호를 이용하여 매핑 테이블을 복구한다. 그림 1(a)는 SSD의 SPOR 동작을 보여준다. 물리 페이지 108과 109는 쓰기가 완료되지 못하였으며, 물리 페이지 100~107은 매핑의 갱신이


그림 1 원자성 쓰기 보장 SSD

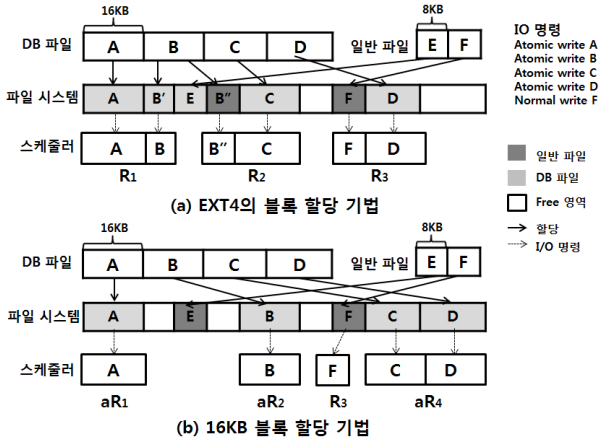


그림 2 16KB 블록할당 기법

적용되지 못하고 전원이 차단되어 SPOR의 대상이 된다. 이에 대한 복구 작업은 물리 페이지 100과 101에 각각 논리 페이지 300과 308의 데이터가 저장되었음을 확인하고, 매핑 테이블을 갱신한다. 또한, 논리 페이지 399~402의 매핑 정보도 같은 방법으로 갱신한다. 하지만, 그림 1의 ①과 같이 16KB 데이터가 완전히 쓰이지 않은 경우, $D_1$과 $D_2$의 데이터만 매핑 테이블 갱신이 진행되어 I/O 명령 aR(500, 4)에 대한 부분적인 쓰기가 발생한다.

기존에 트랜잭션 단위의 원자성을 보장하는 연구들이 진행하였다[3][4]. 하지만 두 연구 모두 트랜잭션을 관리하기 위한 오버헤드가 존재하며, 물리 페이지 별로 트랜잭션의 정보를 저장할 영역이 필요하였다. 본 연구에서는 SSD와 호스트 간의 인터페이스를 규정하여 매우 작은 오버헤드로 16KB 데이터의 원자성을 보장하도록 하였다.

본 연구에서 제안하는 16KB 원자성 보장 쓰기 명령의 경우, 데이터베이스의 특성(16KB 단위 쓰기)을 고려하여 16KB의 정렬된 시작 주소를 가진 쓰기 명령에 대해서만 원자성을 보장한다. 또한, 쓰기 명령의 크기가 16KB의 배수가 되어야 한다. 만약에 쓰기 명령이 16KB에 정렬되지 않으면, 16KB의 데이터는 여러 매핑 페이지에 저장될 수 있어 매핑 테이블의 원자적인 저장이 불가능하여 16KB 데이터의 매핑이 부분적으로 갱신될 수 있다. 이 경우 16KB 데이터의 원자적인 복구를 위해서는, 16KB 데이터의 트랜잭션 정보가 추가로 필요하게 되어 원자성 보장을 위한 오버헤드가 커진다. 본 연구에서 제안한 인터페이스는 논리 페이지의 할당에만 제약을 두고, SSD의 물리 페이지 할당에 영향을 주지 않기 때문에 SSD의 성능에 영향을 주지 않는다.

그림 1(b)는 16KB 원자성 쓰기 보장 SSD의 SPOR 모습을 보여준다. 기존 기법과 달리, 호스트는 16KB에 정렬된 주소로 aR(400, 4)라는 쓰기 명령어를 보내고 있다. 본 기법은 16KB 크기의 원자성 쓰기 요청과 일반 쓰기 요청을 서로 다른 방식으로 복구 작업을 진행한다. 두 가지 쓰기 요청을 구분하기 위하여 페이지의 스페어 영역에 쓰기 요청의 종류를 구분하는 Atomic 플래그를 추가하였다. 물리 페이지는 Atomic 플래그가 1로 세팅된 경우 16KB 크기의 원자성 쓰기 요청의 일부이고, 세팅되지 않은 경우 일반 쓰기 요청이다. 예를 들어 논리 페이지 300과 308은 일반 쓰기 요청이고, 논리 페이지 400~403은 16KB 크기의 원자성 쓰기 요청이다.

먼저 일반 쓰기 요청에 대해서는 기존의 복구 방법을 그대로 사용한다. 하지만, Atomic 플래그가 1로 세팅된 경우에는 16KB의 데이터 페이지가 모두 쓰기 완료 되었음을 확인한 후 매핑 정보를 갱신한다. 그림 1의 ②는 논리 페이지 400~403의 물리 페이지들이다. 해당 페이지들은 16KB의 데이터가 모두 적혔으므로 매핑 테이블을 갱신한다. 그림 1의 ③은 논리 페이지 500~503의 물리 페이지들이다. 물리 페이지 108과 109의 데이터가 모두 쓰여지지 않은 페이지이므로 논리 페이지 500~503의 매핑 테이블 갱신을 하지 않는다. 이를 통해 그림 1(a)와 같은 부분적 데이터 갱신이 발생하지 않도록 보장한다.

## 3. 리눅스 I/O 서브 시스템

### 3.1 기존 기법의 문제점

MariaDB는 데이터의 관리 단위로 16KB를 사용한다. 그림 2(a)는 EXT4의 블록 할당 기법을 보여준다. 데이터베이스에는 A, B, C, D라는 16KB 데이터가 4개 존재하며, MariaDB가 관리하지 않는 일반 파일에 E, F라는 8KB 데이터 2개가 존재한다고 가정한다. 데이터 B의 할당의 경우, 이미 할당된 일반 파일에 의해 분리된 2개의 영역을 할당받아 B′ 과 B″ 이라는 2개의 데이터로 분리되어 저장된다. 또한, 데이터 C의 경우 16KB에 정렬되지 않게 할당된다. 이 2가지 경우는 앞에서 정의한 SSD의 인터페이스에 맞지 않는 상황이다.

또한 데이터 B′ 과 B″ 은 서로 다른 물리 영역에 할당되었으므로 스케줄러에 의해 병합될 수 없으며, 2개의 I/O 명령으로 처리된다. 본 연구에서 제시한 SSD는 연속된 16KB 쓰기 명령의 원자성을 보장하지만 2개로 나누어진 I/O는 원자적으로 처리할 수 없다. 또한 일반 파일 데이터 F와 데이터베이스의 데이터 D는 스케줄러에 의해 병합되어 하나의 I/O 명령으로 SSD에 전달 된다. 이 경우 두 종류의 쓰기 요청이 병합되어 원자성 쓰기 요청으로 전송되지 못한다. 따라서 본 연구에서는 파일시스템과 스케줄러를

수정하여 본 연구에서 제시한 SSD가 원자성 쓰기를 보장할 수 있도록 수정하였다.

### 3.2 원자성 쓰기 지원 파일시스템

EXT4 파일시스템은 블록 할당 기법을 통해 할당할 영역의 시작 물리 주소와 크기를 결정한다. 본 연구에서는 EXT4의 블록 할당 기법을 수정하여 원자성 쓰기 파일의 경우에 시작 물리 주소가 16KB에 정렬되고, 크기는 16KB의 배수가 되도록 수정하였다. 그림 2(b)는 수정된 블록 할당 기법을 보여준다. 기존의 EXT4의 블록 할당 기법에서는 미리 할당된 블록에 연속된 블록이 다른 파일에 의해 할당된 경우, 데이터 B가 연속하지 않은 2개의 구간에 나누어져 저장되었지만, 본 연구에서 제안하는 기법에서는 데이터 B를 16KB에 정렬된 영역에 할당한다. 위의 2가지 기법을 통해 본 연구에서 제시한 SSD의 원자성 쓰기가 가능하도록 하였다.

### 3.3 원자성 쓰기 지원 I/O 스케줄러

그림 2(b)는 수정된 I/O 스케줄러 기법을 보여준다. 16KB 원자성 쓰기 요청의 경우 특정 플래그를 가진 채 I/O 스케줄러로 전달된다. 플래그를 가진 명령은 원자적 요청(그림 2의 **aR**)으로 처리되고, 일반 명령은 일반 요청(그림 2의 **R**)으로 처리된다. 스케줄러는 두 요청 간의 병합 처리를 하지 않는다. 예를 들어, C의 데이터는 일반 파일인 F와 주소가 연속되지만 병합 작업을 거치지 않고 독립적으로 처리된다. 또한 데이터 C와 D는 연속된 원자성 쓰기 명령이므로 병합을 통해 하나의 I/O 명령으로 처리된다.

## 4. 실 험

실험에서는 CPU가 Intel i5-4670 쿼드 코어, 메모리가 4GB인 호스트 PC를, 커널 버전 3.8.0의 우분투 12.10 OS를 사용하였으며, tpcc-mysql[6]을 사용하여 성능을 평가하였다. Tpcc-mysql은 percona에서 개발한 tpcc 워크로드를 테스트하는 벤치마킹 툴이다. 원자성 쓰기 지원 SSD가 특별한 성능 저하가 없으므로 기존 SSD(Samsung SSD 840 PRO)를 원자성 쓰기가 보장된다는 가정으로 사용하였다.

MariaDB는 버전 5.5.34를 사용하였고, 메모리에 1GB의 버퍼 영역을 할당하였으며, 트랜잭션의 정보를 저장하는 로그 파일의 크기는 128MB로, 실험 시간은 10분으로 설정하였다. 이중 쓰기 버퍼를 사용하는 *double*, 이중 쓰기 버퍼를 사용하지 않는 *no_double*, 그리고 원자성 쓰기를 사용하는 *atomic*에 대해 실험을 진행하였다.

그림 3은 성능 평가 결과를 보여준다. x축은 데이터베이스의 크기와 connection의 개수를, y축은 처리량의 상대값을 보여준다. 데이터베이스의 크기는 500MB에서 3GB까지, connection의 개수는 20, 50, 100까지 바꾸어가며 성능을 평가하였다. *no_double*과 *atomic*이 이중 쓰기 영역과 원본 위치에 2번의 쓰기를 하는 *double*에 비해 1.5~2.2배의 높은 성능을 보이는 것을 확인하였다. 이중 쓰기 버퍼가 *no_double*에 비해 쓰기량이 2배이지만 성능이 반으로 줄지 않는 원인은 임의 쓰기가 이중 쓰기 버퍼 영역에 대한

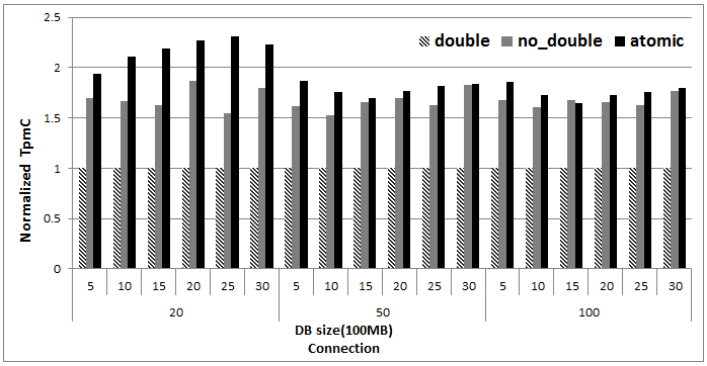


그림 3 TPC-c 실험 결과

연속적 쓰기로 변경되기 때문이다.

MariaDB의 원자성 쓰기를 사용하는 경우 *no_double* 보다 1.1~1.3배의 향상된 성능을 보인다. 이 성능 차이는 원자성 쓰기에서 사용하는 다이렉트 I/O로 인한 것이다. 다이렉트 I/O를 사용하면 일반 블록 계층과 페이지 캐시를 거치지 않고 디바이스로 바로 명령 전달이 가능하다. 또한 쓰기 명령이 페이지 캐시를 거치지 않음으로 인해 메모리 사용량 또한 줄일 수 있다. 추가적으로 *no_double*에서 보장하지 못하는 데이터의 무결성의 보장이 가능하다.

## 5. 결론 및 계획

데이터베이스에서 원자성 쓰기는 좋은 성능을 보이지만, 원자성 쓰기를 지원하는 공개된 파일시스템, SSD, I/O 스케줄러가 존재하지 않는다. 본 연구에서는 범용 파일시스템인 EXT4와 원자성 쓰기를 지원하는 SSD를 디자인하여 16KB 데이터의 원자성을 보장하도록 하였다.

MariaDB의 원자성 쓰기를 사용한 결과, 이중 쓰기 버퍼를 사용하는 것에 비해 좋은 성능을 보이는 것을 알 수 있었다. 본 연구에서는 SSD의 기법을 제시하였지만, 실제 SSD의 구현을 통해 실험을 하지는 못하였다. 향후에는 SSD 개발 환경인 OpenSSD[7]를 사용하여 16KB 데이터의 원자성을 보장하는 실제 하드웨어에서 MariaDB의 원자성 쓰기 성능을 평가할 것이다.

## Reference

[1] MySQL, http://www.mysql.com/.

[2] MariaDB, https://mariadb.org/.

[3] Xiangyong Ouyang, et al., “Beyond block I/O: Rethinking traditional storage primitives,” HPCA, 2011.

[4] Youyou Lu, et al., “LightTx: A lightweight transactional design in flash-based SSDs to support flexible transactions,” ICCD, 2013.

[5] Avantika Mathur, et al., “The new ext4 filesystem: current status and future plans,” Proceedings of the Linux Symposium, 2007.

[6] tpcc-mysql, http://www.percona.com/about-us/mysql-white-paper/clustrix-tpcc-mysql-benchmark.

[7] OpenSSD project, http://www.openssd-project.org.