블록 리매핑을 활용한 트랜잭션 처리 최적화 지원 로그 파일 시스템

진주영○, 박대준, 신동군
성균관대학교 정보통신대학
siennajjy@skku.edu, pdaejun@skku.edu, dongkun@skku.edu

Minimizing Transaction Overhead in Log-Structured File System with Block Remapping

Jhuyeong Jhin○, Daejun Park, Dongkun Shin
College of Information and Communication Engineering, Sungkyunkwan University

요 약

데이터베이스 관리 시스템은 데이터베이스 내의 일관성을 유지하기 위해 WAL(Write-ahead Logging) 방식을 주로 사용한다. 변경 데이터를 데이터베이스 파일에 바로 쓰지 않고 WAL 파일에 먼저 기록한다. WAL 파일이 일정 크기 이상으로 커지면 체크포인트를 발생시켜 WAL 파일에 기록된 변경 데이터들을 데이터베이스 파일에 복사하여 기록한다. 이러한 방식은 동일한 데이터에 대한 중복된 쓰기를 유발하여 쓰기 연산의 증가를 초래한다. 이와 같은 단점을 보완하기 위해, 파일시스템의 메타데이터가 가지고 있는 파일과 저장장치 사이의 매핑 정보를 수정하여 중복된 쓰기를 방지하는 Block Remap 기법이 EXT4 파일시스템을 위해 제안되었으나 WAL 파일의 단편화를 발생시켜 임의쓰기를 유발하는 단점이 있었다. 본 논문에서는 Block Remap 기법을 로그 구조 파일시스템에 적용하여 저장장치에 대한 쓰기 연산 없이 트랜잭션 체크포인트가 가능한 로그 파일 시스템을 제안한다. 제안한 기법을 통해 쓰기 양과 트랜잭션 처리 시간이 감소함을 확인하였다.

1. 서 론

NAND 플래시 저장장치는 저전력, 높은 내구성, 높은 임의 접근 성능 등의 장점을 가지고 있어 최근에 많이 사용되고 있다. 그러나 P/E(Program/Erase) 횟수가 제한되어 있어 플래시 메모리 블록이 제한된 P/E 횟수만큼 사용되면 해당 블록은 더 이상 쓸 수 없게 된다.

한편 데이터베이스 관리 시스템은 사용자가 데이터베이스 내의 데이터에 효율적으로 접근하게 하는 소프트웨어로, 스마트 디바이스 역시 어플리케이션의 데이터와 시스템 설정과 같은 정보들을 관리하기 위하여 데이터베이스 관리 시스템을 사용한다. 데이터베이스에 데이터를 기록하던 중 시스템에 장애가 발생하면 일관성이 훼손되는데, 데이터베이스 관리 시스템은 데이터베이스의 일관성을 유지하기 위해 저널 방식과 WAL[1] 방식을 사용한다.

WAL 방식은 데이터 변경 시에, 변경될 데이터를 WAL 파일에 우선 기록하고 일정량 기록되면 데이터베이스 체크포인트를 발생시킨다. 이때 WAL 파일에 기록된 변경 데이터를 데이터베이스 파일에 다시 기록한다. WAL 파일에 기록한 데이터를 데이터베이스 파일에 다시 한 번 기록하는 것은 결과적으로 동일한 데이터에 대한 중복 기록을 발생시키는 것이므로 쓰기 연산을 증가시키고, 스마트 디바이스에서 주로 사용하는 P/E 횟수에 제한이 있는 낸드 플래시 기반의 저장장치의 수명에 영향을 미친다.

이러한 문제를 해결하기 위해 Block Remap 기법[2]이 제안되었다. 이 기법은 중복된 쓰기를 수행하는 데이터베이스 체크포인트의 동작을 파일시스템에서 파일과 저장장치 사이의 블록 매핑을 수정하기 위해 파일시스템의 메타데이터만을 변경하는 기법이다. 이 기법을 통해 실제 데이터 쓰기 없이 DB 트랜잭션에 대한 체크포인트를 수행할 수 있게 한다.

해당 기법은 EXT4 파일시스템에 적용되었는데, 이때 데이터베이스 체크포인트를 수행할 때마다 연속적인 블록들에 순차적으로 매핑되어 있는 WAL 파일이 DB 파일의 블록들과 서로 교체되기 때문에 WAL 파일이 단편화된다. 이 때문에 체크포인트 수행 후 WAL 파일 로그 기록 시 임의쓰기가 유발되어 쓰기 연산 성능이 떨어지는 결과를 초래한다.

기존의 Block Remap 기법의 단점을 보완하기 위해 최근 WAL 파일 사전 할당 기법(EXT4 Block Remap with Preallocation, 이하 Block Remap+)[3]이 제안되었다. 해당 기법은 WAL 파일에 많은 연속적인 블록들을 미리 할당해두고, 체크포인트 발생 후 Block Remap을 마지막으로 수행한 WAL 파일 블록의 다음 블록부터 순차쓰기로 로그 기록을 수행한다. 그러나 WAL 파일은 데이터베이스의 개수만큼 생성되기 때문에 저장장치에 많은 영역을 차지하게 된다. 또 미리 할당한 블록을 모두 소진하면 단편화되어 있는 많은 블록들을 무효화시키고, 또다시 WAL 파일에 많은 블록들을 재 할당해야 하는 오버헤드가 있다.

본 논문에서는 Block Remap 기법을 로그 구조 파일시스템(LFS; Log-structured File System)[4]에 적용하여 저장장치에 대한 쓰기 연산 없이 트랜잭션 체크포인트가 가능한 로그 파일 시스템을 제안한다. 로그 구조 파일시스템은 데이터를 업데이트 할 때 원래 위치에

본 연구는 미래창조과학부 및 정보통신기술진흥센터의 서울어코드활성화지원사업의 연구결과로 수행되었음. (IITP-2015-R0613-15-1062)

덮어쓰지 않고 이전에 기록한 데이터는 무효화하여, 새로 쓰여지는 데이터는 다른 곳에 순차적으로 기록하는 특징을 가지고 있다. 따라서 LFS Block Remap 기법으로 데이터베이스 체크포인트를 수행하여 WAL 파일이 단편화되더라도, 다음 WAL 파일에 로그 기록 시 기존 단편화된 데이터들을 무효화시키고 다른 위치에 순차쓰기를 수행한다. 이러한 로그 구조 파일시스템의 특성의 영향으로 기존 WAL 기법과 Block Remap 기법, Block Remap+ 기법 대비 성능이 향상된다.

2. 관련 연구

데이터베이스는 갑작스런 이벤트로 인해 데이터베이스가 손상되는 것을 방지하고, 데이터의 변경을 이루는 여러 블록들에 대한 쓰기를 원자적으로 처리하기 위해 여러 가지 저널 방식을 제공하는데, 전통적으로 롤백 저널 방식을 사용하였다. 그러나 롤백 저널 방식에는 레코드 조회와 삽입을 동시에 수행할 수 없다는 단점이 있다.

그러나 WAL 기법은 원본 데이터베이스에 변경을 가하지 않고 모든 변경 데이터를 WAL 파일에 순차 쓰기 방식으로 기록한다. WAL 파일에 특정 크기 이상의 데이터가 기록되면 체크포인트가 발생하여 WAL 파일에 기록된 변경 데이터들이 데이터베이스 파일에 복사된다. WAL의 내용을 데이터베이스 파일에 전부 복사하기 전에 시스템에 장애가 생기면, 복구 과정에서 WAL의 내용을 데이터베이스에 적용하는 동작을 마저 수행하는 방식으로 데이터베이스의 일관성을 보장할 수 있다.

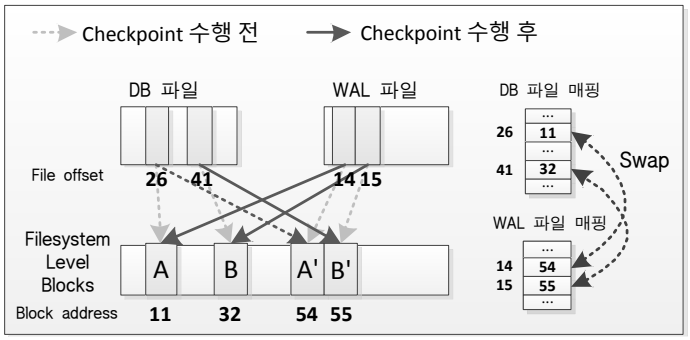

그림 1 Block Remap 동작 원리

기존 WAL 기법은 체크포인트 수행 시 WAL 파일에 먼저 기록해두었던 데이터들을 데이터베이스 파일로 복사한다. Block Remap 기법에서는 데이터를 실제 쓰기 없이 그림 1과 같이 파일시스템의 메타데이터가 가지고 있는 파일과 저장장치 사이의 블록 매핑 정보를 수정하여 WAL 파일에 기록된 데이터들을 데이터베이스 파일로 이동시킨다. 이러한 동작을 통해 체크포인트 수행 시 데이터에 대한 쓰기 연산은 발생하지 않고, 적은 양의 파일시스템 노드와 메타데이터에 대한 쓰기만 발생한다.

그러나 Block Remap 기법 적용 시, 체크포인트를 수행할 때마다 WAL 파일이 단편화되는 단점이 나타난다. 그림 1을 보면, 데이터베이스 체크포인트 수행 전 WAL 파일의 오프셋 14번, 15번은 블록 주소 54, 55에 연속적으로 매핑되어 있다. 그러나 체크포인트를 수행한 후, WAL 파일 오프셋 14번, 15번의 파일 블록에 대한 매핑이 블록 주소 11, 32로 각각 바뀐다. 연속적으로 매핑되어 순차쓰기가 가능했던 WAL 파일이 단편화되어 임의쓰기가 발생하게 되는 것이다. 이는 쓰기 성능의 하락을 초래한다.

Block Remap+ 기법은 Block Remap 기법의 단점을 보완하여 최근에 제안된 기법이다. Block Remap+는 WAL 파일을 위해 많은 연속적인 블록들을 미리 할당하여, 체크포인트 수행 이후에도 WAL 파일에 순차쓰기로 로그를 기록할 수 있게 하는 기법이다. 그러나 WAL 파일은 데이터베이스의 개수만큼 생성되기 때문에 저장장치에 많은 영역을 차지하게 된다. 또 미리 할당한 블록을 모두 쓰게 되면 단편화되어 있는 많은 블록들을 무효화시키고, 또다시 WAL 파일에 많은 블록들을 재 할당해야 하는 단점이 있다.

3. LFS Block Remap

로그 구조 파일시스템은 데이터 업데이트 시 원래 위치에 덮어쓰지 않고 무효화시킨 후 새로운 위치에 순차적으로 쓰는 특징을 가지고 있다. 이 때문에 기존 Block Remap 기법에서 나타났던 WAL 파일의 단편화로 인한 WAL 파일에 대한 임의쓰기 문제가 발생하지 않는다. 또한 Block Remap+ 기법과 달리 WAL 파일을 위해 저장장치에 많은 공간을 사전 할당하지 않아도 된다.

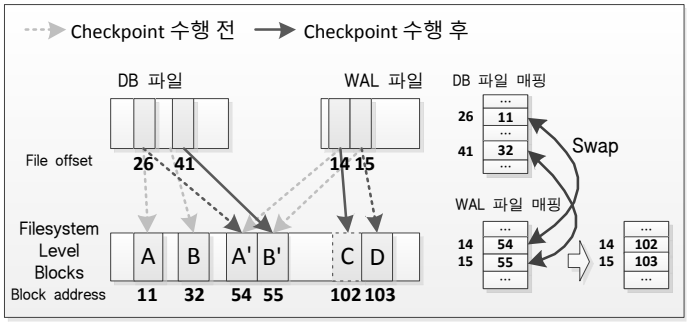

그림 2 LFS Block remap 동작 원리

그림 2에서 데이터베이스 체크포인트 수행 전 WAL 파일의 오프셋 14번, 15번은 블록 주소 54, 55에 순차적으로 매핑되어 있다. 체크포인트 수행 시 DB 파일과 WAL 파일의 매핑을 서로 바꾸어주면 WAL 파일 오프셋 14번, 15번의 파일 블록에 대한 매핑이 블록 주소 11, 32로 각각 바뀐다. 체크포인트 수행 완료 후 새로운 변경 데이터를 WAL 파일에 기록할 때, 블록 주소 11, 32인 블록들을 무효화시키고, 새로운 블록 주소 102, 103을 할당 받아 WAL 파일 오프셋 14번, 15번과 매핑시킨다. 단편화된 기존 영역을 그대로 사용하지 않고 무효화시킨 후, 새로운 연속적인 공간을 할당 받아 임의쓰기가 발생하지 않는다.

```
while( walIteratorNext(pIter, &iDbpage, &iFrame) == 0 ) {
        read(WAL FD, buffer, szPage, WAL file offset);
        write(DB FD, buffer, szPage, DB file offset);
}
```

그림 3 WAL 기법의 체크포인트 동작 코드

그림 3의 코드와 같이 기존 WAL 기법에서는 데이터베이스 관리 시스템에서는 체크포인트 함수에서 Read 시스템 콜을 호출하여 WAL 파일로부터 DB 파일에 복사할 데이터를 버퍼로 읽어 들이고, 이를 Write 시스템 콜을 호출하여 DB 파일에 쓴다.

Block Remap 기법에서는 파일시스템 레벨에서 블록 리매핑 동작을 수행하는 시스템 콜을 작성하였고, 이를 remap 시스템 콜이라 한다. 체크포인트 함수에서 Read/Write 시스템 콜을 호출하는 대신 remap 시스템 콜을 호출하여 체크포인트 기능을 수행한다.

```
while( walIteratorNext(pIter, &iDbpage, &iFrame) == 0 ) {
        offsets[count*2] = WAL file frame offset
        offsets[count*2+1]  = DB file offset
        count++
}
syscall(remap, WAL FD, DB FD, offsets, count);
```

그림 4 LFS Block remap 기법의 체크포인트 동작 코드

데이터베이스 관리 시스템의 체크포인트 함수에서 그림 4의 코드와 같이 파일 블록 매핑을 서로 바꾸어 줄 파일 오프셋 쌍들을 offsets 배열에 넣고, remap 시스템 콜을 호출하여 두 개의 파일의 메타데이터들이 가지고 있는 파일 블록 매핑 정보를 수정함으로써 실제 데이터 쓰기 없이 WAL 파일의 데이터를 DB 파일로 옮긴다. remap 시스템 콜의 매개변수는 WAL 파일 디스크립터, DB 파일 디스크립터, offsets 배열의 포인터와 파일 오프셋 쌍들의 개수이다.

4. 실 험

본 논문에서는 성능 측정을 위해 안드로이드 젤리빈 버전을 사용하는 상용 스마트폰 갤럭시 S4를 이용하였고, 파일시스템은 로그 구조 파일 시스템의 일종인 F2FS 파일시스템을 사용하였다. 쓰기 양과 수행 시간 측정을 위하여 blktrace와 SQLite에 대한 벤치마크 프로그램인 Mobibench를 이용하였다.

Mobibench 벤치마크 프로그램을 사용하여 트랜잭션의 수를 1000개부터 5000개까지 1000개 단위로 실험하였으며, 각각의 트랜잭션은 8KB 크기의 INSERT 요청이다. 실험을 통하여 F2FS Block Remap 기법과 EXT4 파일시스템에서의 기존 WAL 기법, F2FS 파일시스템에서의 기존 WAL 기법, EXT4 DB remap, EXT4 DB remap+ (L), (S)들을 비교하였다. EXT4 Block Remap+ (L)은 WAL 파일에 블록 사전할당을 150MB로 크게 할당하여 실험 중에 WAL 파일에 대한 재할당이 발생하지 않도록 했고, EXT4 Block Remap+ (S)는 WAL 파일의 크기를 40MB로 사전 할당하여 체크포인트 10번 수행할 때마다 블록 재할당이 발생한다.

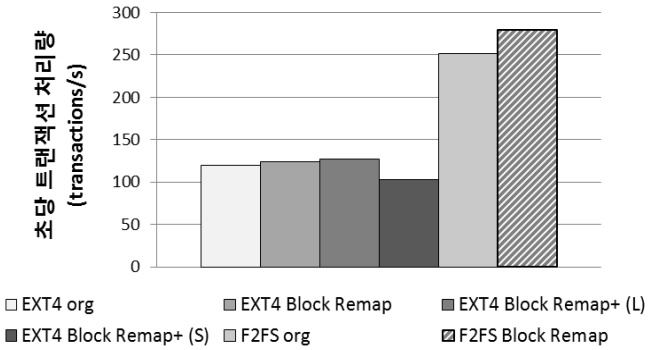

그림 5 초당 트랜잭션 처리량 비교 실험 결과

그림 5는 초당 트랜잭션 처리 개수의 비교를 나타낸다. EXT4 파일시스템에서의 기존 WAL 기법 대비, EXT4 Block Remap 기법은 초당 약 4%, EXT4 Block Remap+ 기법 (L)은 초당 약 7%의 트랜잭션을 더 많이 처리하고, EXT4 DB remap+ 기법 (S)는 초당 약 14% 트랜잭션을 더 적게 처리한다. 즉, WAL 파일에 사전할당을 작게 하면 오히려 성능이 저하된다. F2FS Block Remap 기법은 F2FS 파일시스템에서의 기존 WAL 기법 대비 초당 약 11%의 트랜잭션을 더 많이 처리한다. EXT4 파일시스템 대비 F2FS 파일시스템에서 더 많은 성능 향상을 보였다.

그림 6은 쓰기양의 비교를 나타낸다. EXT4 파일시스템에서의 기존 WAL 기법 대비, EXT4 Block Remap 기법은 약 15%, EXT4 Block Remap+ 기법 (L)은 약 26%, EXT4 Block Remap+ 기법 (S)는 약 25%의 쓰기 감소율을 보였다. F2FS Block Remap 기법은 F2FS 파일시스템에서의 기존 WAL 기법 대비 약 21% 더 적게 쓴다.

EXT4 Block Remap+ 기법 (L)는 EXT4 기존 WAL 기법과 EXT4 Block Remap 기법에 비해 성능이 향상되고, F2FS Block Remap 기법보다 더 높은 쓰기 감소율을 보이지만, 이는 WAL 파일에 사전할당을 크게 할당하여 블록 재할당을 발생시키지 않은 실험의 결과이다. WAL 파일은 데이터베이스 파일 하나 당 생성되기 때문에, 만약 어플리케이션 100개를 설치한다면 약 14.6GB의 저장공간이 WAL 파일들을 위해서 사용되어야 한다. EXT4 Block Remap+ 기법 (S)의 경우 체크포인트를 10번 수행할 때마다 블록 재할당이 발생하는데, 성능이 EXT4 기존 WAL 기법보다 떨어짐을 확인하였다.

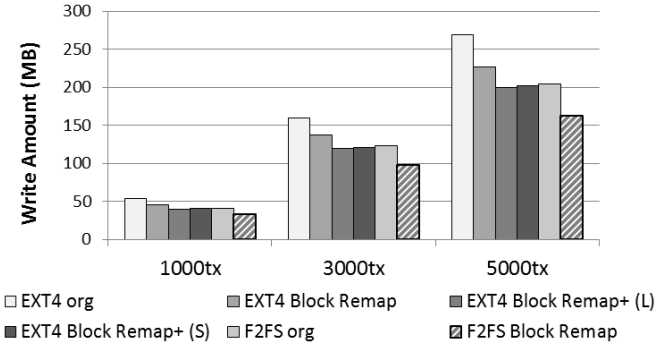

그림 6 쓰기 연산량 비교 실험 결과

5. 결 론

동일한 데이터에 대한 중복된 쓰기를 유발하는 데이터베이스의 WAL 기법은 낸드 플래시 기반의 저장장치의 수명에 영향을 미친다. 이를 보완하기 위해 Block Remap 기법이 제안되었으나 성능이 떨어지는 단점이 있었고, Block Remap+ 기법의 경우 WAL 파일이 저장공간을 많이 차지하는 단점이 있었다. 본 논문에서는 Block Remap 기법을 로그 구조 파일시스템에 적용하여 저장장치에 대한 쓰기 연산 없이 트랜잭션 체크포인트가 가능한 로그 파일 시스템을 제안하였다. 실험을 통하여 WAL 파일이 기존 WAL 기법과 같은 크기의 저장공간을 차지하면서도 21%의 쓰기 감소율과 11%의 성능 향상을 보였다.

참 고 문 헌

[1] http://www.sqlite.org/wal.html

[2] 박대준: 신동군, "파일시스템 메타데이터 수정을 활용한 SQLite WAL 체크포인트 기법 연구." 한국정보과학회 2014 한국컴퓨터종합학술대회 논문집, 1415-1417, 2014

[3] PARK, Daejun; SHIN, Dongkun. Removing duplicated writes at DB checkpointing with file system-level block remapping. In: Proceedings of the 12th ACM International Conference on Computing Frontiers. ACM, p. 37. 2015

[4] ROSENBLUM, Mendel; OUSTERHOUT, John K. The design and implementation of a log-structured file system. ACM Transactions on Computer Systems (TOCS), 10.1: 26-52. 1992