SQLite를 위한 데이터베이스 파일 기반의 멀티 데이터베이스 트랜잭션*

오준택○ 원유집
한양대학교 컴퓨터소프트웨어학부
na94jun@hanyang.ac.kr, yjwon@hanyang.ac.kr

Database file based Multi-database Transaction in SQLite

Joontaek Oh○ Youjip Won
School of Computer software, Hanyang University

요 약

본 논문은 SQLite에서 저널링 없이 데이터베이스 파일만을 사용하여 트랜잭션의 ACID 특성을 유지하는 기술들을 위한 멀티 데이터베이스 트랜잭션 기법을 제안한다. 기존 SQLite의 멀티 데이터베이스 트랜잭션은 저널 파일에 초점이 맞추어져 있기 때문에, 저널 파일을 사용하지 않는 기술들은 멀티 데이터베이스 트랜잭션을 수행할 수 없다. 본 논문은 데이터베이스 헤더의 예약 공간을 활용한 데이터베이스 파일 중심의 멀티 데이터베이스 트랜잭션 기법을 제안하고, LS-MVBT에 이를 적용하여 성능 향상을 보인다.

1. 서 론

SQLite [3]는serverless DBMS이다. SQLite는 따로 데이터베이스 서버가 존재하지 않고, 사용자 프로세스가 SQLite 라이브러리 함수를 통해 데이터베이스에 직접 접근하도록 한다. 또한, 데이터베이스는 사용자 어플리케이션이 접근할 수 있는 하나의 파일로써 관리된다. SQLite는 이러한 특성을 통해 사용자 어플리케이션이 자신의 데이터를 간단하게 관리하도록 한다. 또한, 데이터베이스를 갱신할 때, 데이터베이스 파일과 별개의 파일인 저널 파일에 갱신 정보를 로깅한 후, 데이터베이스를 갱신하는 저널링 기법을 사용하여 트랜잭션의 ACID 특성을 지킨다.

SQLite는 하나의 트랜잭션으로 다수의 데이터베이스를 갱신하는 멀티 데이터베이스 트랜잭션 기능을 제공한다. 사용자는 SQLite가 제공하는 멀티 데이터베이스 트랜잭션 기능을 사용하여 좀 더 유동적인 데이터베이스 관리를 할 수 있다.

하지만, 현재 SQLite의 멀티 데이터베이스 트랜잭션은 저널 파일에 초점이 맞추어져 있다. 많은 이전 연구들은 저널 파일 없이 트랜잭션의 ACID를 지킬 수 있는 메커니즘을 소개해왔다. 하지만, 현재 SQLite의 멀티 데이터베이스 트랜잭션이 저널 파일에 초점이 맞추어져 있어 LS-MVBT[1]와 같은 최신 기법에서는 멀티 데이터베이스 트랜잭션을 사용할 수 없는 한계가 있다.

본 논문에서는 저널 파일 없이 트랜잭션의 ACID 특성을 지키는 기법들을 위한 멀티 데이터베이스 트랜잭션 기법을 소개한다. 해당 기법을 위해서는 데이터베이스 헤더 내 예약 공간을 사용해야 한다. 그래서, 본 논문은 먼저 데이터베이스 헤더에 대한 이해를 위해 데이터베이스의 구조를 설명하고, SQLite의 멀티 데이터베이스 트랜잭션이 어떻게 동작하는지 설명한 후, 우리가 제안하는 기법에 대해 설명한다.

* 본 연구는 한국연구재단 기초연구실 지원사업(No. 2017R1A4A1015498), IITP 전문연구실 지원사업(2018-0-00549)의 지원을 받아 수행되었음.

2. 배 경

2.1. SQLite의 데이터베이스 구조

SQLite는 데이터베이스 하나를 하나의 파일로 관리한다. 이 파일을 데이터베이스 파일이라고 한다. 데이터베이스 파일은 B+ Tree로 구성된 4 KB 크기의 페이지의 집합이다. 각각의 페이지에는 cell의 형태로 데이터가 저장된다.

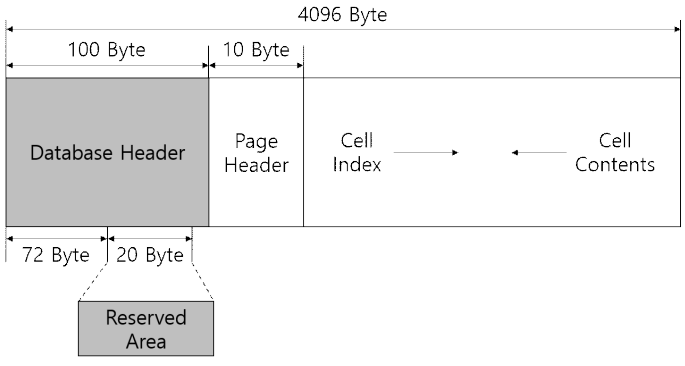

그림 1. 데이터베이스 헤더 페이지의 구조

B+ Tree에서 루트 페이지, 즉, 데이터베이스 파일 내에서 가장 첫번째 페이지는 데이터베이스 헤더 페이지다. 해당 페이지에는 사용자 데이터가 저장되지 않고, 데이터베이스 스키마가 저장된다. 또한, 상위 100 바이트에 데이터베이스 헤더를 저장한다. 그림 1은 데이터베이스 헤더 페이지의 구조를 보여준다. 데이터베이스 헤더는

데이터베이스 파일에 대한 각종 정보를 갖는다. 현재 SQLite의 데이터베이스 헤더는 20 바이트의 예약 공간을 갖는다.

2.2. SQLite의 멀티 데이터베이스 트랜잭션

SQLite는 멀티 데이터베이스 트랜잭션 기능을 제공한다. 멀티 데이터베이스 트랜잭션은 다수의 데이터베이스를 갱신하는 트랜잭션이다. SQLite에서 트랜잭션의 커밋 여부를 결정하는 것은 저널 파일의 유효 여부이다. 그래서, 특정 데이터베이스 파일을 열었을 때, 해당 데이터베이스 파일에 대한 저널 파일을 먼저 찾고, 해당 저널 파일 내 로그들이 유효한지 검사한 후, 로그 사용 여부를 결정한다.

멀티 데이터베이스 트랜잭션의 경우, 저널 파일이 유효한지를 판단한 후, 부가 동작이 필요하다. 같은 트랜잭션에 의해서 같이 갱신된 데이터베이스 파일을 찾는 동작과 해당 데이터베이스 파일들의 저널 파일들이 유효한지 검사하는 동작이다.

SQLite는 상기 동작을 위해서 멀티 데이터베이스 트랜잭션을 수행할 때, 추가 파일을 하나 더 생성한다. 이 추가 파일을 마스터 저널 파일이라고 한다. 마스터 저널 파일은 사용자가 하나의 저널 파일로부터 같은 트랜잭션에 의해서 같이 갱신된 다른 저널 파일들을 찾을 수 있도록 하는 파일이다.

SQLite는 가장 먼저 마스터 저널 파일을 생성하고, 멀티 데이터베이스 트랜잭션이 수정하는 저널 파일들의 이름을 마스터 저널 파일에 쓴다. 그리고, 각 저널 파일을 커밋하는데, 각 저널 파일에는 마스터 저널 파일의 이름이 추가로 쓰여진다. 크래시 상황에서, 특정 저널 파일의 유효성을 판단할 경우, 저널 파일에 저장된 마스터 저널 파일의 이름을 통해 마스터 저널 파일에 접근한 후, 다른 저널 파일들이 유효한지도 확인하여 저널 파일 내 로그의 사용 여부를 판단한다.

마스터 저널 파일에 저널 파일의 이름이 아닌 데이터베이스 파일의 이름을 쓸 수도 있다. 하지만, 데이터베이스 파일은 저널 파일과 달리 유효 여부를 확인할 수 있는 수단도 없고, 마스터 저널 파일의 이름을 쓸 공간도 없다.

3. Multi-database Transaction without journaling

우리는 두가지를 만족해야 한다. 먼저 데이터베이스 파일 내에 마스터 저널 파일의 이름을 쓸 공간을 확보해야 한다. 그리고, 데이터베이스 파일의 유효 여부를 판단할 수 있어야 한다.

우리는 데이터베이스 헤더 페이지를 사용하여 두 조건을 만족하게 했다. 데이터베이스 헤더 페이지는 Full sync 모드에서 트랜잭션이 수행될 때마다 변경되는 페이지이기 때문에, 트랜잭션 복구 정보를 저장하기에 알맞은 페이지다. 또한, 데이터베이스 헤더 페이지에는 데이터베이스의 스키마만 문자열의 형태로 저장되는데, 이러한 이유로 데이터베이스 헤더 페이지의 공간은 부족하지 않다. 먼저, 마스터 저널 파일의 이름은 스키마처럼 데이터베이스 헤더 페이지에 삽입한다. 멀티 데이터베이스 트랜잭션이 완료되면 마스터 저널 파일의 이름은 삭제되어, 추후 데이터베이스의 스키마가 변경되어도 영향을 주지 않는다. 또한, 그림 1에서처럼 cell이 차지할 수 있는 영역은 유동적이기 때문에 마스터 저널 파일의 이름의 길이에 상관없이 유동적으로 저장할 수 있다. 저장할 수 있는 마스터 저널 파일의 이름의 최대 길이는 4 KB - 데이터베이스 헤더의 크기(100 byte) - 페이지 헤더의 크기 (10 byte) - 스키마의 크기다. 즉, 약 3 KB다. 복구 과정에서 사용자는 데이터베이스 파일에서 마스터 저널 파일의 이름이 저장되어 있는지 판단하고, 마스터 저널 파일의 이름을 찾을 수 있어야 한다. 그리고, 상기한 바와 같이 해당 데이터베이스의 유효 여부를 판단할 수 있어야 한다. 우리는 마스터 저널 파일의 이름이 저장된 cell의 index, 4 바이트 난수 값, 데이터베이스 롤백 정보를 예약 공간에 저장한다. 데이터베이스 롤백 정보는 사용되는 기법에 따라 다르며, 크기는 최대 4 바이트가 되도록 한다. 난수는 멀티 데이터베이스 트랜잭션의 signature 값으로 사용된다.

멀티 데이터베이스 트랜잭션 수행 과정에서는 두 가지 동작이 추가된다. 가장 먼저, 4 바이트 난수를 생성한다. 이 때, 각 데이터베이스 헤더에 난수가 저장되어 있는지 확인하고, 해당 난수들과 다른 난수를 생성한다. 난수들은 복구 과정에서 0으로 초기화되지만, 복구 과정 없이 바로 멀티 데이터베이스 트랜잭션이 수행될 경우 난수 값은 0이 아니기 때문에 충돌을 막아야 한다. 난수 생성 후, 마스터 저널 파일을 생성하고 마스터 저널 파일에 각 데이터베이스 파일의 이름들과 난수를 쓴다. 그리고 보통 트랜잭션처럼 트랜잭션을 각 데이터베이스마다 수행한다. 각 데이터베이스의 데이터베이스 헤더를 수정할 때, 데이터베이스 헤더 페이지에 마스터 저널 파일의 이름을 삽입한다. 그리고, 마스터 저널 파일의 이름이 삽입된 cell의 index, 처음에 생성한 4 바이트 난수 값, 롤백 정보를 저장한다.

먼저 데이터베이스 헤더 페이지에 난수가0인지 확인한다. 0이 아니라면 데이터베이스 헤더 페이지에 저장된 cell index를 참조해서 마스터 저널 파일의 이름을 찾고, 마스터 저널 파일을 참조한다. 그리고, 마스터 저널 파일이 가리키는 데이터베이스 파일들에 저장된 난수들이 마스터 저널 파일에 저장된 난수와 같은지 확인한다. 만약 전부 같으면 멀티 데이터베이스 트랜잭션이 정상적으로 수행되었다고 판단하고 롤백을 하지 않고, 난수들을 0으로 변경한다. 만약 난수들이 전부 같지 않다면 멀티 데이터베이스 트랜잭션이 정상적으로 수행되지 않았다고 판단하고 마스터 저널 파일에 저장된 난수와 다른 난수를 갖는 데이터베이스들을 롤백한다.

데이터베이스를 롤백할 때 쓰이는 롤백 정보는 적용되는 기법마다 다르다. 본 논문에서는 저널링 없이 트랜잭션의 ACID 특성을 유지하는 기술 중, 대표적인 기술인 LS-MVBT[1]에 본 논문에서 제안한 기법을 적용한다.

LS-MVBT에서 모든 cell은 시작 버전과 종료 버전을 갖는다. 시작 버전은 해당 cell이 생성될 때의 데이터베이스의 현재 버전이고, 종료 버전은 해당 cell이 삭제될 때의 데이터베이스의 현재 버전이다. 데이터베이스의 현재 버전은 데이터베이스 헤더에 저장된다. LS-MVBT는

버전 정보들을 통해 데이터베이스의 일관성을 지킨다.

LS-MVBT에서 롤백 정보는 현재 버전 정보이다. 본 기법은 LS-MVBT에서 우선적으로 구현되었고LS-MVBT의 롤백 정보인 현재 버전 정보를 데이터베이스 헤더 페이지에 마스터 저널 파일의 이름, 난수와 같이 저장한다.

4. 성능 평가

우리는 LS-MVBT에 본 논문에서 제안한 기법을 적용했다. 그리고, 실험을 통해 기존 멀티 데이터베이스 트랜잭션과의 성능 차이를 측정했다. Samsung 840 Pro SSD를 EXT4로 포맷하여 실험에 사용하였고, 커널 버전은 4.18.0-rc6 버전을 사용하였다. 실험에 사용한 벤치마크 툴은 Mobibench [4]이다. 우리는 Mobibench를 수정하여 하나의 트랜잭션이 수정하는 데이터베이스의 개수를 조절하였다. 우리는 3,5,7,9개의 데이터베이스에 100 바이트 레코드를 삽입하는 트랜잭션을 10,000회 수행하고, 초당 트랜잭션 처리량을 측정했다. 그림 3은 실험의 결과를 보여준다.

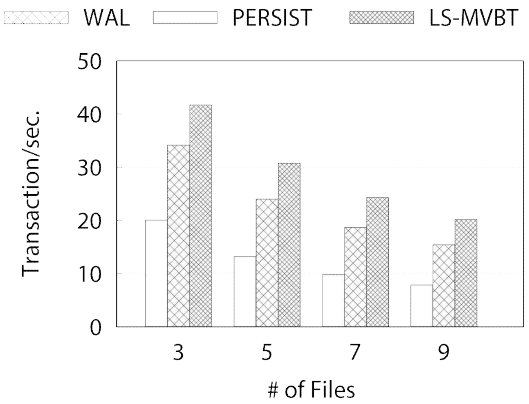

그림 2. 멀티 데이터베이스 트랜잭션의 성능

그림 3은 WAL 모드, PERSIST 모드, LS-MVBT 모드 간의 멀티 데이터베이스 트랜잭션 성능 차이를 보여준다. 9개의 데이터베이스에 100 바이트 레코드를 삽입하는 트랜잭션을 수행할 때, LS-MVBT가 PERSIST 모드에 비해서 2.6배 높은 성능을 보인다. 게다가 본 논문에서 제안한 기법이 적용되어 모든 트랜잭션이 ACID 특성을 유지한다.

5. 결 론

우리는 SQLite에서 저널링 없이 데이터베이스 파일만으로 트랜잭션의 ACID 특성을 유지하는 여러가지 최신 기법을 위한 멀티 데이터베이스 트랜잭션을 디자인하고, 해당 디자인을 LS-MVBT에 구현하여 성능을 측정했다. LS-MVBT는 PERSIST 모드에 비해서 2.6배 더 좋은 성능을 보였다. LS-MVBT는 데이터베이스를 롤백할 수 있는 기능을 가진 기술이기 때문에, 본 디자인이 적용될 수 있었다. 하지만, F2FS의 atomic write을 사용한 SQLite의 트랜잭션의 경우 [2], 데이터베이스를 롤백할 수 없기 때문에, 본 디자인을 적용할 수 없다. 이를 위해서는 새로운 기법을 디자인하거나 파일 시스템 지원을 받아야 할 것이다. 이에 대한 연구는 future work로 남겨놓도록 한다.

참고 문헌

[1] 김욱희, 남범석, 박동일, 원유집, “Resolving journaling of journal anomaly in Android I/O: Multi-version B-tree with lazy split.” In Proc. of the USENIX Conference on File and Storage Technologies (FAST) (2014).

[2] 김재극, “F2FS: support atomic_write feature for database.” https://lkml.org/lkml/2014/9/26/19.

[3] The SQLite http://www.sqlite.org

[4]정수만, 이기성, 황중우, 이성진, 원유집, “AndroStep: Android Storage Performance Analysis Tool,” Software Engineering (Workshops). Vol. 13. 2013.