SQLite기반 애플리케이션에서 롤백 저널 모드와 WAL 모드의 성능 비교 분석

김기호*, 오기환, 이상원
*성균관대학교 컴퓨터공학과
david9921@skku.edu

A Performance Analysis between Rollback Journal mode and WAL mode in SQLite-based Application

Ki-Ho Kim*, Gihwan Oh, Sang-Won Lee
*Dept of Computer Engineering, Sungkyunkwan University

요 약

현재 전 세계적으로 수많은 모바일 스마트 기기를 사용하고 있다. 스마트 기기 대부분 많은 양의 데이터를 처리하기 위해서 SQLite 데이터베이스를 기반으로 사용하고 있다. 본 논문에서는 SQLite가 제공하는 원자성 보장을 위한 두 가지 저널링 모드에 따른 성능 차이에 대하여 비교 분석한다. 안드로이드 모바일 기기에서 Mobibench 애플리케이션을 통해 롤백 저널 (delete 모드) 와 WAL 모드의 트랜잭션 처리 성능을 비교해 본 결과 insert시 각각 약 22TPS ,49TPS 로 WAL 방식이 약 2.5배 더 좋은 TPS를 확인할 수 있었고, 서로 다른 세 종류의 SD카드 환경에서 실제 카카오톡 과 gmail의 쿼리를 추출하여 롤백 저널 (delete 모드) 와 WAL 모드의 수행시간을 측정해본 결과, 모든 SD카드 환경에서 애플리케이션 종류와 상관없이 WAL 모드가 롤백 저널 모드보다 약 2배 빠른 것을 확인 할 수 있었다.

1. 서론

모바일 환경은 휴대성을 고려하기 위해서 한정된 자원을 사용해야 하는 많은 제약사항을 가지고 있다. 모바일 컴퓨팅 기기가 발전함에 따라서 기존과 같이 작은 용량을 가진 기기는 더 이상 찾아보기 힘들어졌으며, 메모리 기술이 발전함에 따라서 대용량을 지원하는 기기가 출현하게 되었다.

저용량의 메모리를 가지고 있는 기기의 경우에는 간단한 데이터 관리 시스템을 가지면 되었지만, 메모리가 커짐에 따라서 다루고자 하는 데이터가 많아지는 경우에는 안정성과 활용성을 고려한 데이터베이스 시스템이 사용되어야 한다. 이러한 모바일 환경상의 요구 사항을 만족시키기 위한 데이터베이스 솔루션으로 대표적인 것이 SQLite이다[1].

SQLite는 임베디드 데이터베이스로서 사용자 측면과 관리자 측면에서 모두 뛰어난 접근성을 가지고 있는 데이터베이스 관리 시스템이다. SQLite는 보통 트랜잭션의 원자성을 보장해주는 롤백 저널 모드를 사용한다. 하지만 롤백 저널 모드는 데이터를 갱신할 때 추가적인 쓰기 연산을 유발하게 되고 이는 디바이스의 성능을 저하 시키게 된다. 따라서 본 논문에서는 동시성을 제공하여 읽기 와 쓰기가 동시에 가능 하고 최소한의 로그만 저장하는 WAL 모드와의 성능을 비교 분석 하여 성능 향상에 대한 더 나은 방안을 찾아보고자 실험을 진행 하였다.

처음 실험에는 일반적인 상황에 대한 성능을 측정하기 위하여 SQLite 기반 안드로이드 기기 상에 Mobibench 벤치마크 애플리케이션을 사용하였다. Mobibench는 안드로이드 기반 기기의 데이터베이스 입출력과 File I/O 입출력 성능을 측정하기 위해 한양대학교에서 개발한 오픈소스 벤치마크이며 SQLite의 작업량에 대한 측정에 용이하고 write()+fsync()에 측정이 가능해 일관성이 보장된다. 또한 사용자가 지정하는 임의의 설정을 사용하여서 성능을 측정할 수 있으며 일반 운영체제 커널 뿐만 아니라 안드로이드의 스마트 폰에서도 측정이 가능한 장점을 가지고 있다[3].

다음으로 본 논문의 목표대로 실제 SQLite 기반의 두 종류의 애플리케이션에서 롤백 저널 모드와 WAL 모드를 비교하는 실험을 진행 하였다. 더 자세한 분석을 위하여 서로 다른 세 종류의 SD카드 환경에서 애플리케이션의 쿼리를 실행하고 수행시간에 대한 성능을 측정한 뒤 비교 분석하였다.

2. SQLite 저널 모드

SQLite는 운영체제의 비정상 종료나 갑작스러운 전원 차단이 발생했을 때, 사용자데이터 일관성을 유지하기 위해 롤백 저널을 제공한다. 롤백 저널 파일은 트랜잭션 과

정 동안 데이터베이스 파일 쓰기 상태 이전의 초기 값을 복원하는데 사용하기 위한 정보를 담는 파일이며, 각각의 데이터베이스 파일마다 롤백 저널 파일을 유지한다.

롤백 저널은 데이터베이스 파일과 같은 디렉터리에 위치하며 데이터베이스 파일 이름 뒤에 "-journal"이 붙는다. 관련된 롤백 저널 파일은 데이터베이스 파일 당 하나만 존재하기 때문에 한 번에 하나의 쓰기 트랜잭션이 가능하다. 응용프로그램이나 운영체제의 비정상 종료, 디바이스 전원 차단과 같은 상황에 의해 트랜잭션이 완료되지 못하게 되면, 데이터베이스는 불일치 상태에 놓이게 되고 SQLite는 다음번에 데이터베이스가 연결될 때 롤백 저널 파일이 있는 경우 완료되지 않은 트랜잭션의 시작 상태로 데이터베이스를 되돌린다. 롤백 저널 은 저널 파일이 존재하며 정상적인 헤더를 포함하고 있을 때에 비로소 세 가지 방식의 저널 모드(DELETE, TRUNCATE, PERSIST) 중 하나로 스토리지에 저널을 반영한다[2].

SQLite에는 표1과 같은 5가지(OFF, MEMORY, DELETE, TRUNCATE, PERSIST)의 롤백 저널 모드 와 WAL 모드가 있다.

<표 1> SQLite의 저널 모드 설명

모드	설명
롤백 저널 모드 : DELETE	기본 모드이며 트랜잭션 종료 시 저널 파일을 삭제한다.
TRUNCATE	종료 시 저널 파일을 지우는 대신 size를 0으로 만든다.
PERSIST	저널 파일을 지우지 않고 트랜잭션마다 덮어 써서 사용한다.
MEMORY	저널 파일을 생성하지 않고 메모리에 기록한다.
OFF	롤백 저널을 사용하지 않는다.
WAL 모드	롤백 저널 대신 write-ahead log 방법을 사용한다.

WAL(Write-Ahead Logging) 저널링 모드는 SQLite version 3.7.0부터 제공하기 시작한 방식으로 롤백 저널 대신 write-ahead log 방법을 사용한다. OFF 저널링 모드는 롤백 저널을 사용하지 않는 것으로 롤백이 동작하지 않을 것이고 트랜잭션 시 충돌이 나면 데이터베이스가 붕괴될 수 있다.

주 비교대상이 될 WAL 모드는 트랜잭션이 종료되더라도 모든 변경사항을 데이터베이스에 기록하는 대신 최소한의 로그만 기록하는 것이다. 다시 말해서 데이터베이스의 일관성을 유지하기 위한 최소한의 로그만 저장하는 방식이다. 동일한 레코드를 수정하더라도 새로운 페이지에 기록하여 히스토리를 남긴다. 특정 개수 이상의 페이지가 생성되면, 체크포인트 이벤트를 발생하여 최신 데이터를 데이터베이스에 반영한다. 이에 따라 롤백 저널 모드에서 동시접근 시 발생하는 데이터베이스 락(lock) 문제가 해결된다. WAL 모드는 동시성을 제공하기 때문에 롤백 저널에 비해 읽기 와 쓰기를 동시에 진행 할 수 있다는 장점이 있다. [1]

3. 실험 및 결과 분석

<표 2> 실험 구현 환경

안드로이드 스마트 폰	기종 : 갤럭시 노트
	운영 체제: Ice Cream Sandwich 4.0.4
	SQLite3 version: 3.8.2
PC	운영 체제 : Ubuntu 14.04
	SQLite3 version: 3.8.2
SD카드	Samsung SD카드 : 32GB
	Sandisk SD카드 : 8GB
	Sandisk micro SD카드 : 16GB

먼저 SQLite 기반 모바일 기기라는 일반적인 상황에서 SQLite의 롤백 저널 모드 와 WAL 모드 의 트랜잭션 처리 성능을 측정하고 비교 분석 해보기 위해서 갤럭시 노트 안드로이드 스마트 폰을 사용하였고 측정 도구로서 Mobibench 을 사용하여 측정 하였다. 기기 상에서의 성능을 측정하기 위해서 갤럭시 노트 기기에 Mobibench 애플리케이션 버전을 설치하였다. 설치 후 안드로이드 기기의 /data 파티션에 쓰기 파일 크기 10MB ,IO 크기 4KB , 트랜잭션 1000 으로 설정하여 임의의 데이터베이스 파일을 생성하고 저널 모드만을 변경해 보면서 트랜잭션 처리 성능을 측정하였다. 저널 모드 중 OFF, MEMORY 모드를 제외한 DELETE , TRUNCATE , PERSIST , WAL 모드만을 같은 환경에서 측정하고 비교하였다.

<표 3> Mobibench 실험 결과

Journaling 모드	TPS(transaction per second)
DELETE	21TPS
TRUNCATE	23TPS
PERSIST	18TPS
WAL	48TPS

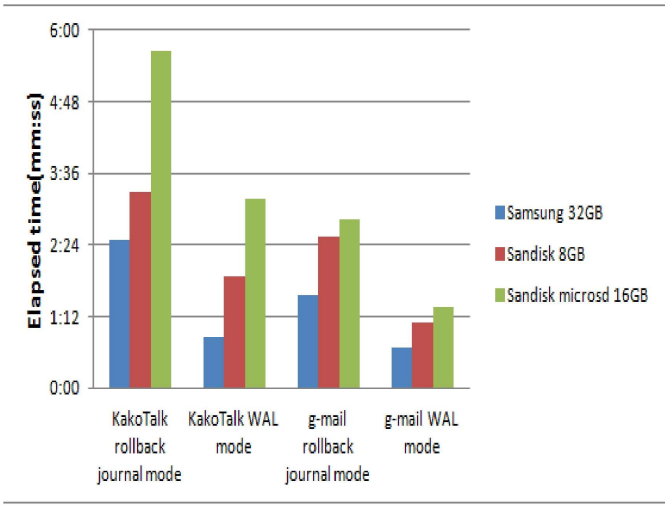
측정 결과 update 와 delete를 실행 할 경우에는 4가지 모드의 성능이 비슷하였고 insert를 실행 할 경우에 성능 차이가 나는 것을 알 수 있었다. insert를 할 경우 롤백 저널 모드 인 DELETE, TRUNCATE, PERSIST 는 평균 20 TPS 를 보였고, WAL 모드는 48 TPS로 약 2.5배 정도 WAL 모드의 성능이 좋다는 것을 알 수 있게 되었다.

(그림 1) 실험 순서

다음으로 본 논문의 목표대로 안드로이드 모바일 기기의 특정 애플리케이션에 대하여 성능을 측정 하였다. 특정 애플리케이션으로는 현재 우리나라에서 가장 많이 쓰이고 있는 메신져 애플리케이션인 카카오톡 과 구글의 메일 애플리케이션인 gmail로 선정하였다.

실제 사용하던 카카오톡 과 gmail 애플리케이션의 쿼리를 안드로이드 기기에서 추출하고 롤백 저널 모드의 기본 모드인 delete 모드 와 WAL 모드 로 변경하였다. 페이지 크기는 4KB (page size=4096)로 동일하게 설정한 뒤 쿼리를 각각 생성하였다.

더 자세한 실험 과 측정을 위해 서로 다른 종류 의 플래시 메모리 환경에서 실험을 진행하고자 했고, 세 개의 SD카드에 실험을 진행 하였다. 서로 다른 두 개의 SD카드와 한 개의 micro SD카드를 사용하였다. 각 각 우분투 14.04 리눅스 환경에 마운트하여 ext4 파일시스템으로 변경 시킨 후 카카오톡 과 gmail 쿼리를 실행시키고 time명령어를 통해 수행시간을 측정 하였다.



(그림 2) SD카드 종류 별 롤백 저널 모드 와 WAL 모드의 수행시간(Elapsed time) 실험 결과

측정 결과 그림 2 와 같은 결과가 나왔다. Samsung 32GB SD카드 환경이 가장 빠른 수행시간을 보여줬고 Sandisk 8GB SD카드는 Samsung 32GB SD카드보다는 느렸지만 용량이 더 크더라도 SD카드의 축소판인 micro SD 카드 보다는 빠른 수행시간을 나타냈다.

결과적으로 SD카드의 종류에 따라 속도차이를 나타냈지만 모든 환경에서 WAL 모드가 롤백 저널 모드 보다 약 2배 빠른 수행시간을 보여줬다.

4. 결론 및 향후 연구

실험을 통해 일반적인 모바일 기기 상에서 TPS를 측정해본 결과 WAL 모드 가 롤백 저널 모드 보다 약 2.5배 좋은 성능을 보여줬고, 모바일 기기 상에서 실제 사용하던 카카오톡 과 gmail 애플리케이션의 쿼리를 추출하여 WAL 모드 와 롤백 저널 모드로 각 각 변경하여 좀 더 구체적인 측정값을 얻기 위해 서로 다른 세 종류의 SD카드 환경에서 수행시간을 측정해본 결과 값도 WAL 모드가 약 2배 빠른 성능을 보여줬다.

결론과 같이 WAL 모드가 더 좋은 성능을 보여주지만 대부분의 모바일 기기나 애플리케이션은 롤백 저널 모드를 사용하고 있다. 그 이유 와 단점을 보완하여 WAL 모드를 적용시킬 수 있는 방안에 대하여 향후 연구해볼 계획이다.

사사표기

1) 이 논문은 2015년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (R0126-15-1108, FlashSQL:비휘발성 메모리 기반 개방형 고성능 DBMS 개발)
2) 이 논문은 2014년도 정부(산업통상자원부)의 재원으로 산업기술 기술혁신사업의 지원을 받아 수행된 연구임 (10049445, 모바일 저장장치용 UFS 2.0 제어기 SoC 및 임베디드 SW 개발)

참고문헌

[1] SQLite , http://www.sqlite.org/index.html

[2] Won Young Lee, "SQLite Optimization Strategy on Tizen" , skku, 2013

[3] Sooman Jeong, Kisung Lee, Jungwoo Hwang, Seongjin Lee, Youjip Won, "AndroStep: Android Storage Performance Analysis Tool", The 1st European Workshop on Mobile Engineering (ME'13), Feb. 26, 2013, Aachen, Germany

[4] Mobibench, https://github.com/ESOS-Lab/Mobibench