SQLite WAL mode multiple transaction

*이원건, 원유집
한양대학교 컴퓨터·소프트웨어 학과
e-mail : inamind@hanyang.ac.kr, yjwon@hanyang.ac.kr

Supplementation of Multiple Transaction Problem on SQLite WAL mode

*Won-Gun Lee, Youjip Won
School of Computer·Software
Hanyang University

Abstract

SQLite는 Android와 iOS의 탑재된 default standalone DBMS로써 그 사용 영역이 날로 증가하고 있다. 우리는 SQLite의 각 journal mode의 성능(tps)을 mobibench benchmark tool [1]로 측정하고 결과를 도표화 하였다. 그 결과 WAL journal mode 는 다른 rollback journal mode 에 비해 약 2,7x ~ 5x의 성능 우위를 보였다, 하지만 조사결과 WAL mode는 Android system에서 journal mode로 거의 선택되지 않고 있다. 우리는 이번 연구에서 WAL mode의 취약점인 multiple transaction 의 atomicity를 보장하기 위해 WAL mode 에서의 master jouranl(mj)을 정의하였다. 그리고 WAL mode 의 multiple transaction 의 처리순서와 각 단계에서 발생하는 system crash에 대해 recovery algorithm을 고안하였다.

I. 서론

과거, firefox, chrome 등 일부 PC application에서 사용되던 SQLite는 smart phone platform인 iOS와 Android의 default Database로 선택되면서, 그 사용이 크게 늘게 되었다. SQLite는 총 6개의 journal mode를 가지고 있다. [2] 우리는 각각의 journal mode 의 성능을 mobibench bench tool을 이용해 측정하였다. 그 결과 WAL mode 는 다른 journal mode 대비 5x의 성능을 보여주었다. 하지만 Android system에서 대부분의 App 들은 SQLite 의 default mode 인 persist mode를 사용하고 있다. 그 이유 중 하나는 WAL mode 는 다른 rollback journal mode 와 달리 multiple transaction 에 대해 atomicity를 보장하지 못하는데 있다. 우리는 WAL mode에서 multiple transaction을 보장하기 위해서 master journal(mj) file을 생성하고 그 구조를 정의하였다. 그리고 multiple transaction을 처리하는 단계와, 각 단계에서 발생하는 system crash 에 대한 recovery algorithm을 고안하였다. 이 기능으로 인해 WAL mode 는 multiple transaction 에 대한 atomicity를 보장하게 되었고, 이로 인해 WAL mode의 선택에 대한 제한요소를 제거하였다.

II. 본론

2.1 SQLite

SQLite 는 오늘날 smart phone 의 대명사인 Android 와 iPhone에 default DBMS로 적용되어 널리 쓰이고 있는 대표적인 standalone DBMS이다. PC 에

서도 Chrome, firefox web browser, Adobe Acrobat Read 등 많은 application 에서 사용 되고 있다. mobile app 이나 PC application 의 기능이 다양화 되고, 복잡도가 높아지면서 가볍고, 빠르며, 이식성 이 좋은 SQLite 에 대한 사용율이 크게 증가하였다[2]. 특히, Android system에서 SQLite 는 default DBMS 로써 많은 App 들에서 data를 저장하고, 관리하기 위해서 SQLite를 매우 빈번하게 사용하고 있고, 큰 비중의 IO traffic을 차지 하고 있다[3].

SQLite에는 5개의 rollback journal mode (OFF, MEMORY, DELETE, TRUNCATE, PERSIST) 와 WAL mode가 있다. 우리는 GalasyS5에서 Mobibench Benchmark Tool [1]을 이용하여 각각 성능을 측정하였다. SQLite는 fdatasync()를 지원하는 환경에서 fsync() 대신 fdatasync()를 사용하도록 되어 있다. 따라서 Andorid system에서 SQLite는 followed by fdatasync()를 사용[4]함을 알 수 있었고, 우리는 fdatasync()를 적용하여 GAlaxyS5에서 새롭게 성능을 측정하였다. 그림.1의 결과에 따르면 WAL mode 는 SQLite 의 default journal mode 인 PERSIST mode 보다 약 2.7x, DELETE, TRUNCATE mode 대비 약 5x 의 performance를 보이고 있다.

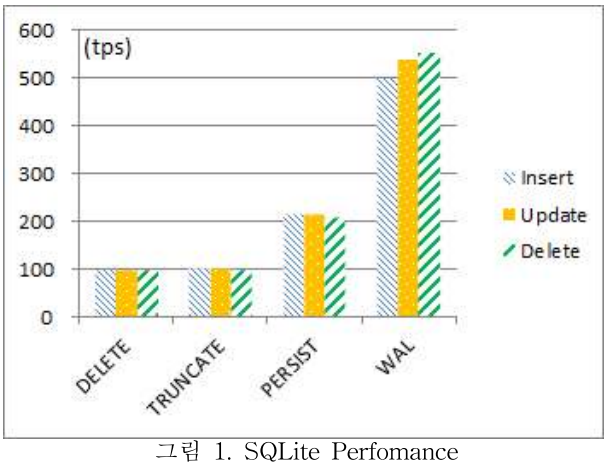

그림 1. SQLite Perfomance

우리는 Nexus5에서 SQLite를 사용하는 각 App 들이 선택한 journal mode를 조사하였다. 그림2. 의 결과에서 우리는 86%의 App 들이 default journal mode 인 PERSIST mode를 사용하고 있음을 알 수 있다. 성능이 가장 좋은 WAL mode 는 단지 5%밖에 사용되고 있지 않다. 이 결과에서 Android App 개발자들은 SQLite 의 journal mode를 직접 선택하지 않고, default journal mode를 사용함을 예측할 수 있다. WAL mode를 default journal mode 로 변경하면, 전체 SQLite 사용의 system overload를 크게 줄일 수 있다.

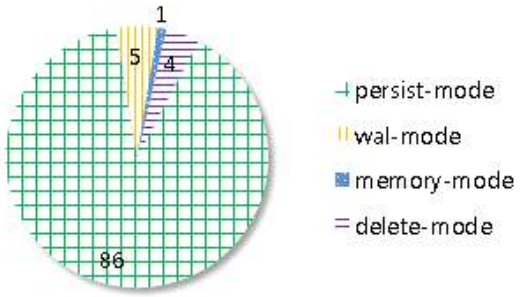

그림 2. SQlite journal mode 사용 비율(%) (Nexus5 Android 4.4)

2.2 Multiple Transaction of SQLite

SQLite 는 DB file 의 저장 및 이동 공간의 용의성을 위해서 ATTACH / DETACH 기능을 지원한다[5]. DETACH function을 통해 SQLite 는 connect 된 DB 의 일부 table을 별도의 .db 파일로 분리해서 extract 할 수 있다. 이렇게 분리 된 .db file 은 다른 storage 공간으로 이동하여 ATTACH를 통해 SQLite connection 에 연결 될 수 있다. SQLite 는 다수의 .db file을 하나의 SQLite connection으로 관리하게 되며, 이때에 발생하는 transaction 은 다수의 .db file에 존재하는 table에 영향을 미치게 된다. 이를 multiple transaction 이라고 명명한다[2].

Rollback journal mode에서 SQLite는 이러한 multiple transaction이 발생했을 때 transaction atomicity를 보장하기 위해서 .db-mj file을 사용한다. 해당 파일은 각 db 마다 생성된 journal file 들의 경로를 string 으로 저장하고 있고, multiple transaction에 관여한 각 journal file 들은 journal header에 .db-mj file 의 full path name을 저장하게 된다.

multiple transaction 의 수행 중 system crash 가 발생했을 때, SQLite 는 recovery를 위해서 .db-mj file을 열어서 안에 적혀있는 .db-journal file 의 list를 읽고, 해당 journal file 들에 저장된 old node 들을 각각의 .db file 에 rollback 한다. 이러한 동작으로 여러 .db file 에 발생된 transaction은 모든 .db 에 적용되지 않는 이상, 하나의 변경사항도 적용되지 않는 atomicity를 보장 할 수 있게 된다.

2.3 WAL mode 의 취약점

WAL mode에서 다른 .db file을 ATTACH 한 후 transaction을 발생시키면 RollBack journal mode 들과 마찬가지로 .db-mj file을 생성한다. 하지만 multiple

transaction이 발생했을 때, WAL file 과 .db-mj file 은 서로 상호 작용을 하지 않는다. .db-mj file 에는 db file name 에 .db-journal 의 postfix 가 붙은 path string 이 저장되지만, 해당경로의 journal file 은 실재하지 않는다. 따라서, WAL mode 상에서 발생하는 multiple transaction시에 system crash가 발생하게 되면 SQLite는 이를 완전히 rollback한 이후 recovery하지 못하는 취약점을 가지고 있다.

Ⅲ. 해결방안

3.1 WAL mode를 위한 mj file 운용 방법

우리는 WAL mode에서 multiple transaction 의 atomicity를 보장할 수 있는 방법을 고안하였다. multiple transaction은 다수의 db file 에 변경을 가하기 때문에, 해당 transaction이 완전히 완료되지 못했다면, 발생된 모든 log record 들은 rollback 되어야 한다. 그림3. 은 page size를 4KB 로 설정하였을 때 WAL file은 구조를 보여주고 있다. 각각의 wal frame 들은 db file 의 B-tree 의 node 에 해당하며, DataBase 의 page 에 해당한다. SQLite 는 WAL mode에서 변경된 page 들을 WAL file 에 logging 하게 된다.

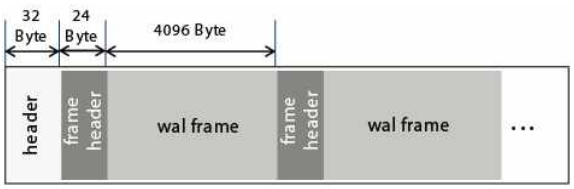

그림 3. WAL file 의 구조 (page size = 4KB)

multiple transaction이 발생하는 WAL file의 atomicity를 보장하기 위해서는 각 db 마다 실행된 transaction 의 수행동안 몇 개의 log 가 wal file 에 logging 되었는지를 기록할 필요가 있다. Rollback journal mode의 경우 변경될 page들 전부가 .db-journal file에 backup 된다. 하지만, WAL mode는 이전 checkpoint 이후에 발생한 모든 transaction의 log page 들이 WAL file 에 logging 되어 있기 때문에, 과거의 transaction 으로 logging 된 page들과 현재 transaction 으로 logging 된 page 들의 구분이 모호하게 된다. 따라서, system crush 이후 정확한 rollback을 위해서는 발생한 log 의 개수를 저장하는 것이 중요하다. 이를 위해 우리는 mj file 내부에 각 WAL file path를 적고, 뒤에 log 의 개수를 명시하도록 하였다.

3.2 Multiple Transaction Scenario

WAL mode에서 multiple transaction 수행 시 각 단계별 진행에 대해 매우 유의해야 한다. 이는 system crash 발생 시 적절한 roll back을 수행하기 위함이다. 고안된 WAL mode의 multiple transaction의 절차는 다음과 같다.

1) 각 wal file 에 transaction을 수행하여 log를 각각의 .db-wal file에 write 한다. 이때 wal-frame header 의 salt-1 값을 0으로 적는다.

Offset	Description
0	Page number
4	For commit records, the size of the database file in pages after commit
8	Salt-1: Copied from the WAL header
12	Salt-2: Copied from the WAL header
16	Checksum-1
20	Checksum-2

테이블 1. wal-frame header 의 구조 [6]

2) mj file을 생성하고, transaction 에 참여한 wal file 들의 path 와 각각의 wal file 에 logging 된 log 의 개수를 관련된 wal file path 뒤에 기록한다.

3) 각 wal file 에 logging 된 log 의 wal-frame header 의 salt-1 의 값을 wal-headr 의 salt-1 값으로 적는다.

4) 마지막으로, 생성된 mj-file을 삭제함으로써 transaction 이 완료 된다.

각, 단계에서 생성되는 file과 저장되는 data는 non-volatile storage에 persistent하게 저장되어야 한다.

3.3 system crash 에 대한 recovery

3.2 절에서 우리는 WAL mode를 위해 고안된 multiple transaction 의 각 단계에 대해서 설명하였다. 이 절에서는 각 단계에서 system crash가 발생하고, system 이 복구되었을 때, database를 transaction 발생 전 단계로 rollback 하는 알고리즘에 대해서 기술하고자 한다.

original SQLite 는 WAL mode 에서 system crash 가 발생하였을 때 SQLite 는 .db file 이 있는 directory에 .db-wal file 이 있는지 확인한다. 만약 그렇다면 wal file을 open 하고 저장된 log들을 read한 후 유효한 log 들을 database 에 checkpoint 하게 된다. 이때 각 log 들의 유효성을 판단하기 위해서 SQLite는 salt 값과 checksum 값을 이용한다 [6].

1) 단계에서 system crash 가 발생했을 때 wal file 에 logging 된 log 들은 salt-1 값이 0 이 된다. SQLite

는 Recovery 수행시 salt-1값이 맞지 않는 log 들은 db에 반영하지 않는다. 따라서, 이 단계에서 logging된 log 들은 database 에 실제 반영되지 않고 결과적으로 모두 rollback 된다.

2), 3) 단계에서 system crash 가 발생하였을 때 mj file 에는 transacton 에 참여한 각 wal file 의 path 와 각각의 wal file 에 logging 된 log 들의 개수가 적혀 있다. SQLite 는 transaction rollback을 위해서 mj file 에 적혀있는 path 의 wal file에서 새롭게 logging 된 log 들을 erase 하거나 salt-1 값을 다시 0으로 적는다. 이후 db 로의 checkpoint를 수행한다.

4) 단계에서는 이미 wal file 에 필요한 log 들이 안정적으로 쓰여진 상태이기 때문에 Recovery 에 문제가 되지 않는다.

위와 같은 절차를 통해서 SQLite는 안정적으로 multiple transaction을 모두 rollback 할 수 있다.

Ⅳ. 결론 및 향후 연구 방향

WAL mode 는 다른 rollback journal mode 에 비해 약 2.7x ~ 5x의 높은 성능을 보여주지만, 실제 Android system에서 단지 5%만의 process만이 WAL journal mode를 선택하고 있다. WAL mode를 SQLite 의 default journal mode 로 선택하기 위해서 우리는 WAL mode의 multiple transaction의 처리에 대한 취약점을 해결할 수 있는 방법을 찾았다. 이 연구를 바탕으로 mobile 환경에서 app과 middleware가 필요로 하는 기능과 성능을 고려한 SQlite의 사용에 대한 연구가 좀 더 진행되길 바란다.

참고문헌

[1] LEE, K. Mobile Benchmark Tool (MOBIBENCH). http:// github.com,/ESOS-Lab/Mobibench

[2] "SQLite" www.sqlite.org

[3] Sooman Jeong AndroStep: Android Storage Performance Analysis Tool (Software Engineering Workshops, 2013)

[4] Ham "Activation Pattern Analysis on Malicious Android Mobile Applications" (IEEE Computer Society, 2013)

[5] Owens “Introducing SQLite” (The Definitive Guide to SQLite, 2006)

[6] Haldar “SQLite Database System: Design and Implementation”