Atomic Write를 활용한 SQLite 최적화

김형득*

*삼성전자

SQLite Optimization with Atomic Write

Hyung-deuk Kim*

*Samsung Electronics

E-mail : hd3.kim@samsung.com

요 약

여러 연구에 따르면 임베디드 디바이스에서 프로세서 및 네트워크의 속도는 사용자의 요구사항을 충족시킬 만큼 빠른데 반해 IO 속도가 성능의 주요 병목으로 밝혀진 바 있다. 또한 이런 IO 병목 현상의 70% 이상이 SQLite 데이터베이스와 관련된 현상으로 밝혀졌다. 이를 해결하기 위한 SQLite 성능 최적화 관련 연구들은 쓰기 IO에 최적화된 저널 방식인 WAL 방식 중심의 연구들이 다수를 이루고 있다. 본 논문에서는 Android와 Tizen에서 주로 사용되는 Rollback 저널 방식 환경 하에서 성능 문제 해결을 위한 SQLite Atomic Write 기법을 제안한다. 제안한 기법을 통해 파일 쓰기, 동기화 작업을 줄임으로써 쓰기 성능(300%)과 메모리 사용량(80%)이 향상 된 것을 확인할 수 있었고 JOJ(저널링 파일에 대한 저널링) 현상을 막고 플래시 메모리의 수명을 늘릴 수 있다.

ABSTRACT

According to researches, while the speed of processor and network in embedded devices is fast enough to meet user requirement, the IO speed is recognized as the main performance bottleneck. Meanwhile it is known that more than 70 percent of IOs are issued from SQLite database. Many researches related SQLite performance optimization is based on WAL mode because WAL mode optimized for write IO performance. In this paper, I propose to optimize SQLite with Atomic Write in the Rollback Journal Mode, which is mainly used in Android and Tizen. I have observed that Atomic Write have a significant write performance improvement(300%) by reducing write, file sync operation and memory usage improvement(80%). Additionally it can block JOJ(Journaling of Journal) and extend the life of the flash memory.

키워드

SQLite, Atomic Write, 저널, 최적화

Ⅰ. 서 론

스마트폰과 같은 임베디드 기기는 매우 보편적이고 사용자의 요구사항을 충족시키기 위해 빠르게 발전하고 있다. 이전에는 느린 네트워크 속도와 처리 속도가 모바일 장치의 성능 병목 현상의 주요 원인으로 간주되어 왔으나 요즘 모바일 장치의 네트워크 및 프로세서 속도는 사용자 요구 사항을 충족시킬 만큼 충분히 빨라졌고 사용자에게 매우 높은 성능을 제공하게 되었다.

그러나 프로세서와 네트워크의 속도가 충분히 빨라졌음에도 불구하고 사용자 애플리케이션의 실행 속도는 기기 개발 속도만큼 빠르게 증가하지 않았다. 이는 프로세서의 개발 속도와 달리 입출력 성능 개선이 미흡하기 때문에 전체 시스템의 성능 저하가 나타나기 때문이다. 여러 연구에 의해 IO 속도가 성능 병목 현상의 주요 원인임이 밝혀졌고[1][2] 이러한 IO의 70% 이상이 SQLite 데이터베이스와 연관되어 있음이 밝혀진바 있다.[3][4]

SQLite는 작은 라이브러리를 제공하는 오픈 소스 데이터베이스 엔진이다. SQLite는 크로스 플랫폼, 적은 메모리 사용, 롤백 저널 등과 같은 많은 장점을 가지고 있기 때문에 Android, IOS, Tizen과

같은 임베디드 시스템에서 널리 사용되었습니다. 사용자에게 시스템 안정성을 제공하기 위해 SQLite는 저널링 기술을 제공합니다. 이 기술은 안정성을 제공하기 위해 필수적이지만, 많은 부가적인 IO 및 추가 과정을 생성하고, 성능에 부정적인 영향을 미칩니다.

본 논문에서는 추가 IO를 줄이고 성능 및 메모리 사용을 향상시켜 SQLite의 병목 현상을 최소화할 수 있는 Atomic Write를 이용한 최적화 기법을 제안한다.

Ⅱ. 배 경

SQLite는 크게 롤백 저널 모드(Rollback Jounral Mode)와 WAL 모드 2가지의 저널(journal) 모드를 제공하고 있다. 다른 데이터베이스 시스템과 마찬가지로 SQLite는 이러한 저널 모드를 통해 일관성과 안정성을 관리하고 갑작스러운 오류로 인한 데이터베이스 손상을 방지한다.

SQLite 롤백 저널 모드와 WAL 모드는 최신 페이지 버전과 물리적 로그를 다른 공간에 저장하는 등 IO 시스템에 차이가 있다. 롤백 저널 모드는 페이지가 페이지 캐시에서 제거 될 때 안정적인 데이터베이스에 최신 페이지 버전을 저장하고 저장소에 대한 임의 쓰기(random write)를 일으킨다. 반면, SQLite WAL 모드는 페이지가 페이지 캐시에서 제거 될 때 최신 페이지 버전을 안정된 로그 영역에 저장한다. 이때 임의 쓰기보다 빠른 순차 쓰기(sequential write)가 발생하며 임의 쓰기는 CHECKPOINT에서만 발생하게 된다. 이러한 모드들 간의 IO 패턴의 차이는 성능에 중요한 영향을 미치게 되고 WAL 모드가 롤백 저널 모드보다 IO 성능에 최적화 되어 있음을 나타낸다[5]. 이러한 이유로 SQLite 성능 최적화와 관련된 최근 연구들은 상당수 WAL 모드에 기반하고 있다. [6][7][8].

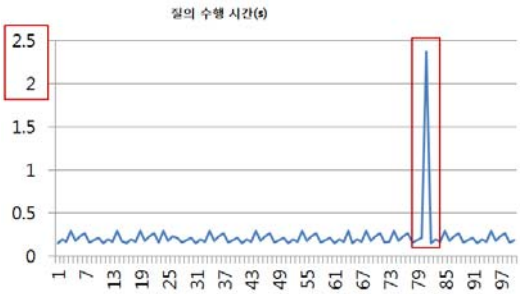

그림 1. WAL 모드의 성능 튐 현상

그러나 WAL 모드 사용 시 그림1과 같이 CHECKPOINT에 따른 성능 튐 현상이 예측하지 못한 시점에 발생할 수 있고 여러 데이터베이스를 함께 사용 시 트랜잭션 보장이 안 되는 등 WAL 모드 사용에 따른 단점도 존재한다.

이처럼 각 모드에는 장단점이 있고 Nexus5 (Android 4.4)의 86% 어플리케이션이 롤백 저널 모드를 사용하고 있고 Tizen에서는 모든 어플리케이션과 시스템 서비스가 롤백 저널 모드를 사용할 만큼 많은 경우 롤백 저널 모드를 사용하고 있기에 롤백 저널 모드에 대한 SQLite 최적화가 필요한 상태이다.

Ⅲ. ATOMIC WRITE

롤백 저널 모드에서는 그림 2의 과정을 통해 삽입(Insert) 질의(query)가 수행되는데 이때 기본적으로 3~5 번의 fsync 혹은 fdatasync가 발생하고 10번 이상의 쓰기(write) 작업이 이루어진다. 이 과정에서 저널 파일에 데이터 쓰기, 저널 파일 삭제 및 디렉토리 동기화와 같은 작업은 쓰기 작업이 Atomic이면 수행되지 않을 수 있는 과정이다. (그림 2의 박스 과정)

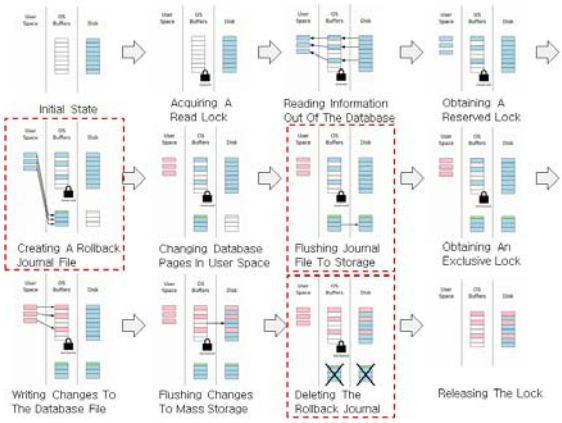

그림 2. Atomic commit을 위한 SQLite 수행 단계

SQLite에서는 Atomic write를 위한 컴파일 옵션(SQLITE_ENABLE_ATOMIC_WRITE)[10]을 제공한다.

```
static int unixDeviceCharacteristics(sqlite3_file *id){
  unixFile *p = (unixFile*)id;
  int rc = 0;
#ifdef __QNXNTO__
  if( p->sectorSize==0 ) unixSectorSize(id);
  rc = p->deviceCharacteristics;
#endif

  rc |= SQLITE_IOCAP_ATOMIC4K;

  if( p->ctrlFlags & UNIXFILE_PSOW ){
    rc |= SQLITE_IOCAP_POWERSAFE_OVERWRITE;
  }
  return rc;
}
```

그림 3. Atomic write를 위한 설정 예시

SQLite의 Atomic write 기능 활성화를 위해서는 상기 컴파일 옵션을 선언하고 파일 시스템이

Atomic write를 지원함을 선언하기 위해 그림3과 같이 xDeviceCharacteristics 메서드에서 SQLITE_IOCAP_ATOMIC 비트를 선언해 주어야 한다.

이와 같이 Atomic write 컴파일 옵션을 사용하고 파일 시스템이 Atomic write를 지원하며 트랜잭션(transaction)이 데이터베이스 파일의 단일 페이지만 변경하면 트랜잭션은 단일 쓰기 요청으로 수행되고 롤백 저널은 작성되거나 기록되지 않는다. 롤백 저널과 관련된 일련의 과정이 일어나지 않기 때문에 이러한 최적화는 쓰기 작업을 크게 향상시킬 수 있다.

롤백 저널 모드 중 가장 많이 사용되는 PERSIST 모드 하에서 삽입 질의 실행 시 IO횟수, 성능, 메모리 사용량 관점에서 Atomic Write 사용하는 케이스와 사용하지 않는 케이스를 비교 검사했다. 표 1, 2 및 그림 4은 그 결과를 보여준다.

표 1. 삽입 질의 시 I/O 횟수 비교

I/O	Persist	Persist + Atomic Write
Read	3	2
Write	11	3
Open	2	0
Close	2	0
Unlink	1	0
_llseek	12	3
Fdatasync	4	1
Stat64	4	3
Fstat64	4	3
Geteuid32	1	0
Fchown32	1	0
Fcntl64	9	9
Total	54	24

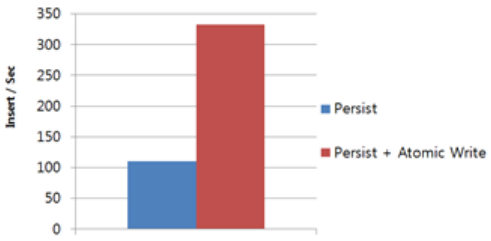

그림 4. 삽입 질의 성능 비교

표 2. 삽입 질의 시 메모리 사용량 비교

I/O (Unit : Byte / Cnt)	Persist	Persist + Atomic Write
Memory Used	502848 (max 505000)	101528 (max 113608)
Largest Allocation	425600	64000
Successful lookaside attempts	99	231
Pager Heap Usage	13176	26408
Schema Heap Usage	904	1128

일부 Atomic write를 지원하지 않는 파일 시스템들이 있어 기본적으로 이 컴파일 옵션은 꺼져 있다. 그러나 가장 널리 사용되는 파일 시스템인 EXT4(Android 및 Tizen에서 사용됨)와 APFS(Apple File System) 및 상당수의 최신 파일 시스템은 Atomic write를 지원한다. 그리고 쓰기 작업의 70 % 가 4KB (페이지 크기)로서 [2] [3] [11] 상당수의 임베디드 시스템에서 SQLite의 Atomic Write를 적용하여 효과를 볼 수 있다.

Ⅳ. 결 론

IO 속도는 주요 성능 병목 현상으로 인식되고 IO의 70 % 이상은 SQLite 데이터베이스에서 발행되며 임베디드 환경 하에서 상당수의 SQLite 데이터베이스는 롤백 저널 모드를 사용 중에 있다. 이 글에서는 롤백 저널 모드 중 Android와 Tizen에서 가장 많이 사용되는 PERSIST 모드 데이터베이스를 최적화를 위해 SQLite에 Atomic Write를 적용함으로서 SQLite에서 300 %의 쓰기 성능 향상을 확인하였고 쓰기 작업 시 메모리 사용량이 1/5로 줄어드는 것을 확인하였다.

참고문헌

[1] H. Kim, N. Agrawal, and C. Ungureanu. Revisiting Storage for Smartphones.ACM Trans.Storage, 8(4):14:1–14:25, Dec. 2012.

[2] K. Lee and Y. Won. Smart Layers and Dumb Result: IO Characterization of an Android-based. Smartphone. Proc. of EMSOFT2012, Tampere, Finland, Oct 2012.

[3] S. Jeong, K. Lee, S. Lee, S. Son, and Y. Won, I/O stack optimization for smartphones, in Proc. of USENIX ATC 2013 Annual Technical Conference,Berkeley, CA, USA, 2013.

[4] M. Kim, S. Lee, and Y. Won, Io workload characterization of tizen based consumer electronics, in Proc. of IEEE ISCE 2014 International Symposium on Consumer Electronics. IEEE, 2014.

[5] J. Sim, D. Shin, W. Kang, S. Lee, Performance Evaluation of SQLite Logging Algorithms, KIPS 2012.

[6] W. Lee, K. Lee, H. Son, W.-H. Kim, B. Nam, and Y. Won, Waldio: Eliminating the filesystem journaling in resolving the journaling of journal anomaly, Proc. of USENIX ATC 2015 Annual Technical Conference, Santa Clara, CA, USA, 2015.

[7] W. Lee, Y. Won, Supplementation of Multiple Transaction Problem on SQLite WAL mode, THE INSTITUTE OF ELECTRONICS ENGINEERS OF KOREA, May 2015

[8] D. Park, D. Shin, DB Remap: Remapping WAL for Checkpointing by Filesystem Metadata modification for SQLite databases, KOREA INFORMATION SCIENCE SOCIETY, June 2014

[9] Atomic Commit In SQLite. http://www.sqlite.org/atomiccommit.html

[10] Atomic Write In SQLite. http://www.sqlite.org/compile.html

[11] Kim, M., Lee, S., and Won, Y, Io workload characterization of tizen based consumer electronics, Proc. of IEEE ISCE 2014 International Symposium on Consumer Electronics, 2014.